Institut für Organisation und Management von Informationssystemen

An Automation-based Approach for Reproducible Evaluations of Distributed DBMS on Elastic Infrastructures

Dissertation

zur Erlangung des Doktorgrades Dr. rer. nat.
der Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
der Universität Ulm

vorgelegt von

Daniel Seybold
aus Crailsheim

2020

This thesis has been written by Daniel Seybold in partial fulfilment of the requirements for a doctoral degree of the faculty of Engineering, Computer Science and Psychology at Ulm University. It has been submitted on June 30, 2020.

Amtierender Dekan:	Prof. Dr.-Ing. Maurits Ortmanns
Erstgutachter:	Prof. Dr.-Ing. Dr. h.c. Stefan Wesner
Zweitgutachter:	Prof. Dr.-Ing. Samuel Kounev
Tag der Promotion:	08.03.2021

Contact

Daniel Seybold
mail: daniel.seybold@uni-ulm.de
www: https://www.uni-ulm.de/in/omi/institut/persons/daniel-seybold/

Institute of Information Resource Management
Faculty of Engineering, Computer Sciences and Psychology
University of Ulm
Albert-Einstein-Allee 43
89081 Ulm, Germany

Typesetting

This document was set by the author using the XeLaTeX typesetting system, the Meta font family, and the Latin Modern font family. My sincere apologies to those people whose names are not typeset correctly due to limitations of the fonts.

Layout

Many thanks to Jörg Domaschka for sharing the template for this thesis.

Abstract

Driven by the data-intensive applications of the Web, Big Data and Internet of Things, Database Management Systems (DBMSs) and their operation have significantly changed over the last decade. Besides relational DBMSs, manifold NoSQL and NewSQL DBMSs evolved, promising a set of non-functional features that are key requirements for each data-intensive application: *high performance*, *horizontal scalability*, *elasticity* and *high-availability*. In order to take full advantage of these non-functional features, the operation of DBMSs is moving towards elastic infrastructures such as the cloud. Cloud computing enables scalability and elasticity on the resource level. Therefore, the storage backend of data-intensive applications is commonly implemented by distributed DBMSs operated on cloud resources.

But the sheer number of heterogeneous DBMSs, cloud resource offers and the resulting number of combinations make the selection and operation of DBMSs a very challenging task. Therefore, supportive analyses of the non-functional DBMS features are essential. But the analyses design and execution is a complex process that involves detailed domain knowledge of multiple domains. First, the multitude of DBMSs technologies with their respective runtime parameters need to be considered. Secondly, the tremendous number of resource offers including their volatile characteristics need to be taken into account. Thirdly, the application-specific workload has to be created by suitable DBMS benchmarks. While supportive DBMS benchmarks only focus on DBMS performance, the evaluation design and execution for advanced non-functional features such as scalability, elasticity and availability becomes even more challenging.

This thesis enables the holistic evaluation of distributed DBMS on elastic infrastructures by defining a supportive methodology that determines the domain-specific impact factors for designing comprehensive DBMS evaluations and establishes a set of evaluation principles to ensure significant results. Moreover, reproducible evaluation processes for the non-functional features performance, scalability, elasticity and availability are established. Based on these concepts results the novel DBMS evaluation framework *Mowgli*. It supports the design and automated execution of performance and scalability evaluation processes. Therefore, Mowgli manages cloud resources, DBMS deployment, workload execution and result processing based on evaluation scenarios, which expose configurable domain-specific parameters. Mowgli follows the established evaluation principles with a dedicated focus on the automated and reproducible evaluation execution. Mowgli is extended with the *Kaa* framework that automates the DBMS elasticity evaluation process by enabling DBMS and workload adaptations. The *King Louie* framework builds upon these features and enables availability evaluations by providing an extensive failure injection framework.

The extensive automation capabilities of Mowgli, Kaa and King Louie ensure reproducible DBMSs evaluations on elastic infrastructures. This enables comparable and novel insights in the non-functional features of distributed DBMSs. Moreover, the automation capabilities facilitate the determination of the the elastic resource impact on the non-functional DBMS features.

In conclusion, this thesis provides a novel DBMS evaluation framework based on the Mowgli, Kaa and King Louie frameworks, enabling comprehensive DBMS evaluations on elastic infrastructures with a dedicated focus on advanced non-functional features as well as automated and reproducible evaluation processes.

Zusammenfassung

Angetrieben durch die datenintensiven Anwendungen des Web, Big Data und Internet der Dinge, haben sich die Datenbankmanagementsysteme (DBMS) und ihr Betrieb in den letzten zehn Jahren erheblich verändert. Neben relationalen DBMS haben sich vielfältige NoSQL- und NewSQL-DBMS entwickelt, welche die Kernanforderungen von datenintensiven Anwendungen versprechen: *Performanz*, *horizontale Skalierbarkeit*, *Elastizität* und *Hochverfügbarkeit*. Um diese nicht-funktionalen Eigenschaften voll auszunutzen, werden elastische Infrastrukturen wie Cloud Computing für den Betrieb von DBMS herangezogen, um Skalierbarkeit und Elastizität auch auf der Ressourcenebene zu ermöglichen. Daher werden moderne Speicherdienste datenintensiver Anwendungen durch verteilte DBMS implementiert, die auf Cloud-Ressourcen betrieben werden.

Doch die bloße Anzahl heterogener DBMS, Cloud-Ressourcenangebote und die daraus resultierenden Kombinationen machen die Auswahl und den Betrieb von DBMS zu einer komplexen Herausforderung. Daher sind unterstützende Analysen der nicht-funktionalen DBMS-Eigenschaften unerlässlich. Jedoch sind Design und Ausführung solcher Analysen komplexe Prozesse, die mehrschichtiges Domänenwissen erfordern. Zunächst müssen die DBMS mit ihren Laufzeitparametern betrachtet werden. Weiter muss die enorme Anzahl von Ressourcenangeboten mit ihren flüchtigen Eigenschaften berücksichtigt werden. Abschließend muss die Anwendungslast durch geeignete DBMS-Benchmarks erzeugt werden. Bestehende DBMS-Benchmarks unterstützen hierbei nur die Erzeugung der Anwendungslast. Zudem zielen sie primär auf die DBMS-Performanz ab, während die Analyse von Skalierbarkeit, Elastizität und Verfügbarkeit außen vor bleibt.

Diese Thesis ermöglicht die ganzheitliche Analyse von DBMS auf elastischen Infrastrukturen durch die Definition einer unterstützenden Methodik. Diese bestimmt die domänenspezifischen Einflussfaktoren für das Design umfassender DBMS-Analysen und definiert Evaluationsprinzipien um signifikante Ergebnisse zu gewährleisten. Zudem werden reproduzierbare Analyseprozesse für die nicht-funktionalen Eigenschaften Performanz, Skalierbarkeit, Elastizität und Verfügbarkeit definiert. Basierend auf dieser Methodik, wird das neuartige DBMS-Evaluations-Framework *Mowgli* bereitgestellt, das den Evaluationsprozess für Performanz und Skalierbarkeit automatisiert. *Mowgli* verwaltet Cloud-Ressourcen, DBMS-Bereitstellung, Lasterzeugung und die Ergebnisverarbeitung auf Basis von konfigurierbaren Evaluationsszenarien. *Mowgli* folgt den Evaluationsprinzipien mit Fokus auf automatisierte und reproduzierbare Evaluationen. *Mowgli* wird durch das *Kaa* Framework erweitert, das den Elastizitätsbewertungsprozess automatisiert, indem es DBMS- und Lastanpassungen automatisiert. Das *King Louie* Framework baut auf diesen Merkmalen auf und ermöglicht die DBMS-Verfügbarkeitsbewertung, indem es ein umfangreiches Fehlerinjektions-Framework bereitstellt

Mowglis umfangreiche Automatisierungskonzepte sowie die Erweiterungen Kaa und King Louie gewährleisten reproduzierbare DBMS-Evaluationen auf elastischen Infrastrukturen, die neuartige und vergleichbare Ergebnisse der nicht-funktionalen DBMS-Eigenschaften ermöglichen. Darüber hinaus erleichtern sie die Bestimmung des Einflusses elastischer Ressourcen auf die nicht-funktionalen DBMS-Eigenschaften.

Zusammenfassend stellt diese Thesis ein neuartiges DBMS-Evaluations-Framework vor, das ganzheitliche DBMS-Evaluationen auf elastischen Infrastrukturen ermöglicht, mit einem speziellen Fokus auf fortgeschrittene nicht-funktionale Merkmale sowie automatisierte und reproduzierbare Evaluationsprozesse.

Acknowledgements

The time as a doctoral researcher has become one of the most challenging, but also most exciting parts of my life. In this time, I was able to meet a lot of inspiring people that have contributed to this thesis in their own way. I am truly thankful to all of these people. Especially, I want to thank my supervisor Stefan Wesner for offering me the position as doctoral researcher, introducing me to the research field of cloud computing, guiding me through the whole journey of his thesis and numerous research projects while always giving me enough freedom to develop own ideas. Many thanks also to my second thesis reviewer Samuel Kounev and for the inspiring discussions on performance engineering.

I want to thank all of my colleagues I have met over the time at the OMI. It was a great experience to develop new research ideas together, working on projects, writing proposals but also all additional activities from coffee breaks to barbecues, hiking and skiing trips. Many of you not only were great colleagues but also became friends. In particular, I want to thank Christopher Hauser for being such a great office mate and a friend for over five years. I want to thank my former colleague Frank Griesinger for the cooperation in different European research projects, including several memorable project travels. I want to thank Volker Foth, Florian Held, Diana Denk for dispersing table soccer matches during the lunch breaks. Thanks also to Daniel Baur for the numerous discussions on cloud orchestration and how to get things done in projects. I want to thank Mark Leznik for the inspiring discussions on how machine learning can be applied for my field of research. I want to thank Simon Volpert for all the technical support at any time and for always introducing a brand new technology. A special thanks goes to my colleague Jörg Domaschka who encouraged me to pursue this journey in the first place. Moreover, I want to thank Jörg for his guidance, patience, critical but always constructive feedback on research ideas and continuous support throughout my thesis, even when he was already overloaded with his own work.

I also want to thank all the people from all over world that I have met on conferences and project around the world. You have enriched my life research- and cultural-wise.

I also want to thank all of my friends in Ulm and Crailsheim for many dispersing adventures during the time of my thesis that helped to free my mind and gather focus for new research activities.

Finally, I want to thank my whole family and especially my parents Petra and Friedrich for their continuous and unsolicited support over my whole life, without you I would have never made it to this point. And of course I want to thank my girlfriend Juliane Eder for her love, support and for encouraging me to start this journey despite the knowledge that this means many days apart from each other. Without all of you, I would never have been able to write this thesis.

List of Publications

This cumulative dissertation is a consolidated report of the research results obtained during the author's doctoral studies. The detailed results have been published in the following peer-reviewed publications that are included in Part II of this thesis. In addition, supplemental data sets and software artefacts have been published.

Core Publications

Journal Publications

[core1] Somnath Mazumdar, **Daniel Seybold**, Kyriakos Kritikos, and Yiannis Verginadis. "A survey on data storage and placement methodologies for Cloud-Big Data ecosystem". In: *Journal of Big Data* 6.1 (Feb. 2019), p. 15. ISSN: 2196-1115. DOI: 10.1186/s40537-019-0178-3.

Conference Publications

[core2] Daniel Baur, **Daniel Seybold**, Frank Griesinger, Athanasios Tsitsipas, Christopher B Hauser, and Jörg Domaschka. "Cloud orchestration features: Are tools fit for purpose?" In: *Utility and Cloud Computing (UCC), 2015 IEEE/ACM 8th International Conference on*. IEEE. 2015, pp. 95–101. DOI: 10.1109/UCC.2015.25.

[core3] **Daniel Seybold** and Jörg Domaschka. "Is Distributed Database Evaluation Cloud-Ready?" In: *European Conference on Advances in Databases and Information Systems (ADBIS) - New Trends in Databases and Information Systems (Short Papers)*. Cham: Springer International Publishing, 2017, pp. 100–108. ISBN: 978-3-319-67162-8. DOI: 10.1007/978-3-319-67162-8_12.

[core4] **Daniel Seybold**, Christopher B. Hauser, Simon Volpert, and Jörg Domaschka. "Gibbon: An Availability Evaluation Framework for Distributed Databases". In: *On the Move to Meaningful Internet Systems. OTM 2017 Conferences*. Cham: Springer International Publishing, 2017, pp. 31–49. ISBN: 978-3-319-69459-7. DOI: 10.1007/978-3-319-69459-7_3.

[core5] **Daniel Seybold**, Moritz Keppler, Daniel Gründler, and Jörg Domaschka. "Mowgli: Finding Your Way in the DBMS Jungle". In: *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*. ICPE '19. Mumbai, India: ACM, 2019, pp. 321–332. ISBN: 978-1-4503-6239-9. DOI: 10.1145/3297663.3310303.

[core6] **Daniel Seybold**, Simon Volpert, Stefan Wesner, André Bauer, Nikolas Herbst, and Jörg Domaschka. "Kaa: Evaluating Elasticity of Cloud-Hosted DBMS". In: *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. Dec. 2019, pp. 54–61. DOI: 10.1109/CloudCom.2019.00020.

[core7] **Daniel Seybold**, Stefan Wesner, and Jörg Domaschka. "King Louie: Reproducible Availability Benchmarking of Cloud-hosted DBMS". In: *35th ACM/SIGAPP Symposium on Applied Computing (SAC '20), March 30-April 3, 2020, Brno, Czech Republic*. Apr. 2020, pp. 144–153. DOI: 10.1145/3341105.3373968.

Workshop Publications

[core8] **Daniel Seybold**, Nicolas Wagner, Benjamin Erb, and Jörg Domaschka. "Is Elasticity of Scalable Databases a Myth?" In: *2016 IEEE International Conference on Big Data (Big Data)*. Dec. 2016, pp. 2827–2836. DOI: 10.1109/BigData.2016.7840931.

[core9] **Daniel Seybold**. "Towards a Framework for Orchestrated Distributed Database Evaluation in the Cloud". In: *Proceedings of the 18th Doctoral Symposium of the 18th International Middleware Conference*. Middleware '17. Las Vegas, Nevada: ACM, 2017, pp. 13–14. ISBN: 978-1-4503-5199-7. DOI: 10.1145/3152688.3152693.

[core10] **Daniel Seybold**, Christopher B. Hauser, Georg Eisenhart, Simon Volpert, and Jörg Domaschka. "The Impact of the Storage Tier: A Baseline Performance Analysis of Containerized DBMS". In: *Euro-Par 2018: Parallel Processing Workshops*. Cham: Springer International Publishing, 2018, pp. 93–105. ISBN: 978-3-030-10549-5. DOI: 10.1007/978-3-030-10549-5_8.

[core11] Jörg Domaschka and **Daniel Seybold**. "Towards Understanding the Performance of Distributed Database Management Systems in Volatile Environments". In: *Symposium on Software Performance*. Vol. 39. Gesellschaft für Informatik. 2019, pp. 11–13. URL: https://pi.informatik.uni-siegen.de/stt/39_4/01_Fachgruppenberichte/SSP2019/SSP2019_Domaschka.pdf.

Software Artefacts & Data Sets

[data1] **Daniel Seybold** and Jörg Domaschka. *Mowgli: DBMS Elasticity Evaluation Data Sets*. Zenodo. Aug. 2019. DOI: 10.5281/zenodo.3362279.

[data2] **Daniel Seybold** and Jörg Domaschka. *Mowgli: DBMS Performance & Scalability Evaluation Data Sets*. Zenodo. Oct. 2019. DOI: 10.5281/zenodo.3518786.

[data3] **Daniel Seybold** and Jörg Domaschka. *Mowgli: Finding Your Way in the DBMS Jungle*. Version 0.1. Zenodo. July 2019. DOI: 10.5281/zenodo.3341512.

[data4] **Daniel Seybold**, Stefan Wesner, and Jörg Domaschka. *King Louie: DBMS Availability Evaluation Data Sets*. Sept. 2019. DOI: 10.5281/zenodo.3459604.

[data5] **Daniel Seybold** and Jörg Domaschka. *Cloud-hosted DBMS Performance, Scalability and Availability Evaluation Data*. June 2020. DOI: 10.5281/zenodo.3901428.

[data6] **Daniel Seybold**, Christopher B. Hauser, Georg Eisenhart, Simon Volpert, and Jörg Domaschka. *Performance Results of a Containerized MongoDB DBMS*. June 2020. DOI: 10.5281/zenodo.3894855.

Additional Publications

During the author's doctoral studies, he has been involved in the following additional publications that partially influenced this dissertation:

Journal Publications

[add1] Frank Griesinger et al. "BPaaS in Multi-cloud Environments - The CloudSocket Approach". In: *European Space Projects: Developments, Implementations and Impacts in a Changing World - Volume 1: EPS Porto 2017,* INSTICC. SciTePress, 2017, pp. 50–74. ISBN: 978-989-758-311-7. DOI: 10.5220/0007901700500074.

[add2] Achilleas P. Achilleos et al. "The cloud application modelling and execution language". In: *Journal of Cloud Computing* 8.1 (Dec. 2019), p. 20. ISSN: 2192-113X. DOI: 10.1186/s13677-019-0138-7.

[add3] Kyriakos Kritikos, Chrysostomos Zeginis, Joaquin Iranzo, Roman Sosa Gonzalez, **Daniel Seybold**, Frank Griesinger, and Jörg Domaschka. "Multi-cloud provisioning of business processes". In: *Journal of Cloud Computing* 8.1 (Nov. 2019), p. 18. ISSN: 2192-113X. DOI: 10.1186/s13677-019-0143-x.

Conference Publications

[add4] **Daniel Seybold**, Jörg Domaschka, Alessandro Rossini, Christopher B Hauser, Frank Griesinger, and Athanasios Tsitsipas. "Experiences of models run-time with EMF and CDO". In: *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*. SLE 2016. ACM. Amsterdam, Netherlands, 2016, pp. 46–56. DOI: 10.1145/2997364.2997380.

[add5] Jörg Domaschka, Frank Griesinger, **Daniel Seybold**, and Stefan Wesner. "A Cloud-driven View on Business Process as a Service". In: *CLOSER*. INSTICC. SciTePress, 2017, pp. 739–746. ISBN: 978-989-758-243-1. DOI: 10.5220/0006393107670774.

[add6] Kyriakos Kritikos, Chrysostomos Zeginis, Frank Griesinger, **Daniel Seybold**, and Jörg Domaschka. "A cross-layer bpaas adaptation framework". In: *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE. 2017, pp. 241–248. DOI: 10.1109/FiCloud.2017.12.

[add7] Daniel Baur, **Daniel Seybold**, Frank Griesinger, Hynek Masata, and Jörg Domaschka. "A provider-agnostic approach to multi-cloud orchestration using a constraint language". In: *Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE. 2018, pp. 173–182. DOI: 10.1109/CCGRID.2018.00032.

[add8] Mark Leznik, Simon Volpert, Frank Griesinger, **Daniel Seybold**, and Jörg Domaschka. "Done yet? A critical introspective of the cloud management toolbox". In: *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. IEEE. 2018, pp. 1–8. DOI: 10.1109/ICE.2018.8436348.

Workshop Publications

[add9] Jörg Domaschka, Daniel Baur, **Daniel Seybold**, and Frank Griesinger. "Cloudiator: a cross-cloud, multi-tenant deployment and runtime engine". In: *9th Symposium and Summer School on Service-Oriented Computing*. 2015. URL: https://domino.research.ibm.com/library/cyberdig.nsf/papers/656B934403848E8A85257F1D00695A63/$File/rc25564.pdf.

[add10] Jörg Domaschka, **Daniel Seybold**, Frank Griesinger, and Daniel Baur. "Axe: A novel approach for generic, flexible, and comprehensive monitoring and adaptation of cross-cloud applications". In: *European Conference on Service-Oriented and Cloud Computing*. Springer. 2015, pp. 184–196. ISBN: 978-3-319-33313-7. DOI: 10.1007/978-3-319-33313-7_14.

[add11] Frank Griesinger, **Daniel Seybold**, Jörg Domaschka, Kyriakos Kritikos, and Robert Woitsch. "A DMN-Based Approach for Dynamic Deployment Modelling of Cloud Applications". In: *European Conference on Service-Oriented and Cloud Computing*. Springer. 2016, pp. 104–111. ISBN: 978-3-319-72125-5. DOI: 10.1007/978-3-319-72125-5_8.

[add12] **Daniel Seybold**, Robert Woitsch, Jörg Domaschka, and Stefan Wesner. "BPaaS Execution in CloudSocket". In: *Advances in Service-Oriented and Cloud Computing (ESOCC 2016)* (2018), pp. 292–293. URL: https://link.springer.com/content/pdf/bbm%3A978-3-319-72125-5%2F1.pdf.

Table of Contents

Table of Contents xv

List of Figures xvii

List of Tables xix

List of Listings xxi

I Thesis 1

1 Introduction 3
- 1.1 Problem Statement 4
- 1.2 Research Objectives 8
- 1.3 Research Contributions 9
- 1.4 Thesis Outline 10

2 Background 11
- 2.1 Elastic Infrastructures 11
  - 2.1.1 Virtualization 11
  - 2.1.2 Cloud Computing 12
  - 2.1.3 Fog and Edge Computing 13
  - 2.1.4 Application Orchestration on Elastic infrastructures 14
- 2.2 Distributed DBMS 14
  - 2.2.1 Data Models 14
  - 2.2.2 Data Distribution Techniques 16
  - 2.2.3 Non-Functional Features 19
  - 2.2.4 Boundaries: CAP and PACELC 20
- 2.3 Operational Models of Distributed DBMS 21
- 2.4 Summary 23

3 Related Work 25
- 3.1 Terminology 25
- 3.2 DBMS Benchmarking 26
  - 3.2.1 OLTP 27
  - 3.2.2 OLAP 28
  - 3.2.3 HTAP 28

3.3 Elastic Infrastructure Benchmarks 28
3.3.1 Cloud Benchmarks 29
3.3.2 Edge and Fog Benchmarks 29
3.4 Advanced Evaluation Frameworks 30
3.5 Summary 30

4 Methodological DBMS Evaluation 33
4.1 Evaluation Impact Factors 34
4.2 Cross-Domain Evaluation Principles 38
4.3 Evaluation Design 40
4.3.1 Performance Evaluation Design 41
4.3.2 Scalability Evaluation Design 42
4.3.3 Elasticity Evaluation Design 43
4.3.4 Availability Evaluation Design 44
4.4 Summary 48

5 Methods for the Automated Evaluation of Non-functional DBMS Features 49
5.1 DBMS Evaluation Templates 49
5.1.1 Deployment Template 51
5.1.2 Workload Template 52
5.1.3 Adaptation Templates 54
5.1.4 Implementation 55
5.2 Mowgli Framework 55
5.2.1 Automation Concepts 56
5.2.2 Framework Architecture 56
5.2.3 Implementation 58
5.3 Mowgli for Higher-Level Evaluation Objectives 58
5.3.1 Elasticity: Kaa Framework 58
5.3.2 Availability: King Louie Framework 60
5.4 Evaluation Data Collection 61
5.5 Summary 63

6 Validation 65
6.1 Case Studies 65
6.1.1 CS1 - Performance Impact of Elastic Resources 65
6.1.2 CS2 - Performance and Scalability 67
6.1.3 CS3 - Elasticity 68
6.1.4 CS4 - Availability 68
6.1.5 Case Study Discussion 69
6.2 Support for Evaluation Principles 70
6.3 Reflections on Evaluating distributed DBMS on Elastic Infrastructures 72
6.4 Summary 74

7 Conclusion and Future Work 75
7.1 Contribution 76
7.2 Future Research 77

Acronyms 79

Bibliography 81

II Publications 101

8 [core1] A survey on data storage and placement methodologies for Cloud-Big Data ecosystem 103

9 [core2] Cloud orchestration features: Are tools fit for purpose? 141

10 [core3] Is Distributed Database Evaluation Cloud-Ready? 149

11 [core4] Gibbon: An Availability Evaluation Framework for Distributed Databases 159

12 [core5] Mowgli: Finding Your Way in the DBMS Jungle 179

13 [core6] Kaa: Evaluating Elasticity of Cloud-Hosted DBMS 193

14 [core7] King Louie: Reproducible Availability Benchmarking of Cloud-hosted DBMS 203

15 [core8] Is elasticity of scalable databases a Myth? 215

16 [core9] Towards a Framework for Orchestrated Distributed Database Evaluation in the Cloud 227

17 [core10] The Impact of the Storage Tier: A Baseline Performance Analysis of Containerized DBMS 231

18 [core11] Towards Understanding the Performance of Distributed DatabaseManagement Systems in Volatile Environments 245

List of Figures

1.1 Determining the DBMS operational model on elastic infrastructures 5
1.2 Thesis scope — reproducible evaluation design and execution for higher-level objectives 7

2.1 Data distribution techniques overview 16
2.2 DBMS operational models on cloud, edge and fog resources 21

3.1 Related research concepts 32

4.1 Evaluation impact factors 34
4.2 DBMS impact factors 35
4.3 Resource impact factors 37
4.4 Workload impact factors 37
4.5 Performance evaluation process 41
4.6 Scalability evaluation process 44
4.7 Elasticity evaluation process 46
4.8 Availability evaluation process 47

5.1 Mowgli architecture 57
5.2 Kaa and King Louie framework architecture 59

List of Tables

4.1 Performance evaluation tasks 42
4.2 Scalability evaluation tasks 43
4.3 Elasticity evaluation tasks 45
4.4 Availability evaluation tasks 47

6.1 Case study metrics 66

List of Listings

5.1 Evaluation scenario template 50
5.2 Exemplary evaluation API 50
5.3 Cloud deployment template 51
5.4 Workload template YCSB 53
5.5 Elasticity Template 54
5.6 Resource failure template 55

Part I

Thesis

Chapter 1

Introduction

Information Technology (IT) has undergone significant advances within the last two decades, especially when it comes to processing and storing of data. The adoptions of large-scale web applications has resulted in a constantly growing number of users. These users, in turn, produce a growing amount of data that needs to be stored and processed. Consequently, the predicted compound annual growth rate of the data management market is over 10% per year [162]. Moreover, its revenue will increase from $89 billion in 2017 to over $140 billion in 2022 [162]. According to 451 Research, this growth is mainly driven by two key technologies of the data management market: distributed data processing frameworks and *distributed Database Management Systems (DBMSs)* [162]. This thesis focuses on distributed DBMS technologies that are the storage backend of many data processing frameworks.

Emerging application domains, such as the Internet of Things (IoT) and Big Data, further increase the amount of data, resulting in a strong need in DBMSs which ensure high throughput and low latency request processing, independent of the data size. Moreover, these data-intensive application domains impose new challenges to DBMS: (geo-) distributed data storage, variety of data, dynamic workload patterns and the need for high-available data [127].

Traditionally, the DBMS landscape has been shaped by Relational Database Management Systems (RDBMSs), which used to be the common solution to persist data since the 1970s. Yet, already in mid of the 2000s the end of their predominance has been predicted [22, 27]. This prediction has become true over the last decade. The DBMS landscape moved towards application-specific data models and distributed architectures to address the requirements of the emerging data-intensive application domains. Driven by new DBMS technologies such as Amazon's Dynamo [25], Google's Big Table [30] or Facebook's Cassandra [42], a new class of DBMSs has been established, the NoSQL DBMSs [51]. NoSQL DBMSs are characterized by providing flexible, non-relational data models and building upon a shared-nothing architecture that eases a distributed architecture, including replication and sharding mechanisms [51, 59]. By the end of 2019, the number of existing NoSQL DBMS has grown to over 220[1].

With the widespread adoption of distributed NoSQL DBMSs [47], in 2012 the DBMS landscape has been extended with a further class of distributed DBMSs, the NewSQL DBMSs [72]. NewSQL seek to provide the same shared-nothing architecture, sharding and replication mechanisms as NoSQL DBMSs while still maintaining the relational data model and strict consistency guarantees for transactions [137] as RDBMS. By the end of 2019, the number of established NewSQL DBMSs has grown to over 20[1].

NoSQL and NewSQL DBMSs promise a set of non-functional features that are key requirements for each data-intensive application domain: *high performance* in terms of low latency and high throughput request processing [9, 105] as well as *horizontal scalability* to be able to deal with growing workloads and data sizes [40,

[1] http://nosql-database.org/

46], *elasticity* to cope with workload fluctuations at runtime [46] and *high-availability* to overcome failures [33, 41].

In order to fully benefit from these non-functional features, the operation of distributed DBMSs has to evolve and move towards elastic resources that provide scalability and elasticity also on the resource level [105, 127]. Elastic infrastructures have become a formative topic in both academia and industry with the opening of the first commercial cloud service by Amazon Web Servicess (AWSs)[2]. Nowadays, the number of cloud resource offers has already grown to over 20,000 in the public cloud provider market[3], not including the manifold private cloud resources. The main advantage of elastic infrastructures is the delivery of the necessary mechanisms to enable scalability on the resource level by providing virtually unlimited resources in an elastic manner [56]. In consequence, elastic resources have become the preferred resources to operate distributed DBMSs [211]. In this context, Gartner predicts that until 2022, 75% of the DBMSs will be operated in the cloud [193].

1.1 Problem Statement

The introduced advantages of the DBMS and resource domain offer extensive technical possibilities to tackle the challenges of modern data-intensive applications. But these advantages also bring a new set of challenges regarding the *selection* and *operation* of DBMSs. In particular, cloud computing and distributed DBMS have driven the disappearance of the 'one size fits all' strategy, where single node RDBMS operated on dedicated hardware appliances have been the predominant solution for most applications [22]. Nowadays, a system architect needs to thoroughly analyse a plethora of modern DBMS and elastic resource offers in the process of selecting and operating the optimal data storage backend [89].

The focus of such analyses also depends on the industry grade of the application. On the one hand, *greenfield applications* require the selection of the optimal elastic resource and DBMS combination from all offers, fulfilling the application-driven functional and non-functional features. On the other hand, there are *enterprise applications* that are building upon existing data structures. If these applications move towards elastic infrastructures, there is the need to evaluate modern DBMS technologies that provide the required data model, identify the optimal elastic resources and refine the DBMS runtime configurations to ensure the efficiency of the non-functional DBMS features on elastic infrastructures.

In consequence, the determination of the *DBMS operational model* for data-intensive applications has become considerably more extensive and challenging [105, 107, 211]. Within this thesis, we follow established definitions of the *DBMS operational model* as the combination of the *DBMS*, its *runtime configuration* and the applied *elastic resources* [105, 107, 211]. In consequence, the operational model of distributed DBMSs on elastic infrastructures needs to be determined from RDBMS, NoSQL DBMS, NewSQL DBMS offers [127] and heterogeneous elastic resource offers [89, 107]. The runtime configurations comprise heterogeneous DBMS- and distribution-specific options [86, 150].

To highlight these challenges, Figure 1.1 depicts an exemplary decision process to determine the *DBMS operational model* for a data-intensive application. The decision process consists of three analytical steps: (i) *functional feature evaluation*; (ii) *non-functional feature evaluation* and (iii) *operational model selection*.

[2] https://www.computerworlduk.com/galleries/cloud-computing/aws-12-defining-moments-for-the-cloud-giant-3636947/

[3] https://cloudharmony.com/

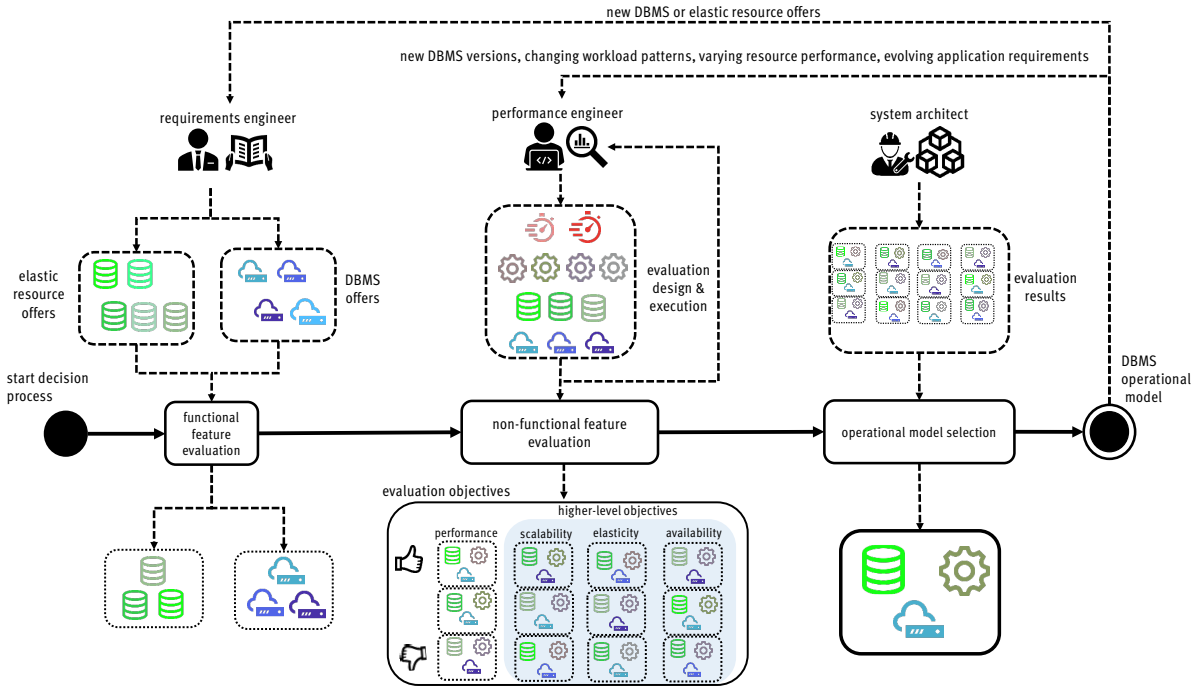


Figure 1.1: Determining the DBMS operational model on elastic infrastructures

Functional feature evaluation First, the *functional feature evaluation* is carried out by the requirements engineer with the goal to determine a set of eligible DBMS and elastic resource offers. Therefore, the application-specific functional requirements are compared with the functional features of the DBMS and elastic resource offers. The step is carried out with the objective to reduce the number of potentially suitable DBMS and elastic resource offers. Depending on the industry grade of the considered DBMSs, this step is supported by DBMS-specific documentations, industrial experience reports and academic decision guidelines [51, 54, 150, 203] providing comparative studies of the functional features and qualitative recommendations.

Non-functional feature evaluation Secondly, the *non-functional feature evaluation* is a continuous process that is carried out by the performance engineer, analysing the non-functional DBMS features for a set of selected DBMSs in experiment-driven scenarios. Based on the eligible DBMS technologies and their non-functional features, the performance engineer needs to consider multiple *evaluation objectives* in this step as depicted in Figure 1.1. For DBMSs in general, *performance* is the most important non-functional feature [9]. Yet, data-intensive applications drive the need to evaluate *higher-level non-functional features* of distributed DBMS that go beyond performance. *Horizontal scalability* enables a distributed DBMS to handle arbitrary workload sizes by operating the DBMS cluster accordingly, exploiting the ”unlimited” capacity of elastic resources [46, 105]. By building upon horizontal scalability, *elasticity* is a required prerequisite non-functional DBMS feature to cope with sudden workload fluctuations by adapting the DBMS cluster at runtime without service disruption where elastic resources provide the elasticity on the resource level [46, 105]. Besides the advantages of elastic resources, they come with a higher risk of resource failures [38, 132]. In consequence,

DBMS *availability* is becoming an important non-functional feature for operating distributed DBMSs on elastic infrastructures [98]. Especially, as the availability of a distribution storage system is always a trade-off between consistency and partition tolerance as defined by the CAP theorem [15]. In consequence, the scope of the evaluation objectives that need to be considered by the performance engineer has significantly increased.

In order to evaluate these evaluation objectives, realistic workloads are required that reflect the characteristics of the target application domain. The requirements of Web, Big Data and IoT applications are leading to a growing number of DBMS benchmarks that enable the emulation of different workload types such as Online Transaction Processing (OLTP) [40, 82, 146], Online Analytical Processing (OLAP) [67, 111] and Hybrid Transaction-Analytical Processing (HTAP) [149].

In consequence, the comprehensive design and execution of DBMS evaluations includes the DBMS, elastic resource and workload domain. Hence, the performance engineer requires detailed domain knowledge for each of these domains.

Operational model selection Thirdly, the results of the non-functional feature evaluation represent the input for the system architect who needs to select the optimal DBMS operational model. Therefore, the quantified evaluation results of the non-functional evaluation step and optional business requirements are included in this decision to determine the DBMS operational model. However, the revision of the selected operational model may become necessary due to the rapidly evolving IT landscape. The revision of the DBMS operational model can be initiated by various events as depicted in Figure 1.1. Events such as new DBMSs and elastic resource offers appearing on the market require the execution of both decision steps. More frequent events such as the release of new DBMS versions, changing DBMS workload patterns, varying elastic resource performance or evolving application requirements demand for the continuous execution of the non-functional feature evaluation to ensure the consistent efficiency of the operational model.

Challenges of non-functional feature evaluations Each step within the decision process comes with its own challenges, but the non-functional feature evaluation represents the most frequently executed steps as depicted in Figure 1.1. Its continuous characteristic [6] increases its importance when new DBMSs and elastic resource offers appear on the market, new DBMS versions are released, elastic resource providers revise their offers [190] and their resource performance changes [44, 135]. Therefore, this thesis focuses on the challenges of the *evaluation design*, *evaluation execution* and *higher-level objectives* which concern the performance engineer who evaluates the non-functional features of DBMSs on elastic infrastructures as depicted in Figure 1.2. In this regard, a particular focus lies on the domain-specific challenges of the involved *DBMS*, *elastic resource* and *workload* domains.

The *DBMS domain* requires knowledge in the DBMS technologies and their runtime configuration options. Especially distributed DBMS provide heterogeneous runtime configurations as the CAP theorem restricts a distributed DBMS to only provide two out of the three properties consistency, availability and tolerance to (network) partitions. The evolvement of distributed DBMSs has shown that these constraints are not binary decisions, but there is a wide range of trade-offs [65]. Consequently, distributed DBMSs differ significantly in their architectures and distribution mechanisms, which results in in a heterogeneous set of distribution configurations. These configurations offer a wide range of possibilities to optimize towards consistency, availability and partition tolerance [150, 165]. While these runtime configurations have a direct impact on the efficiency of the non-functional features, their impact degree is hard to quantify.

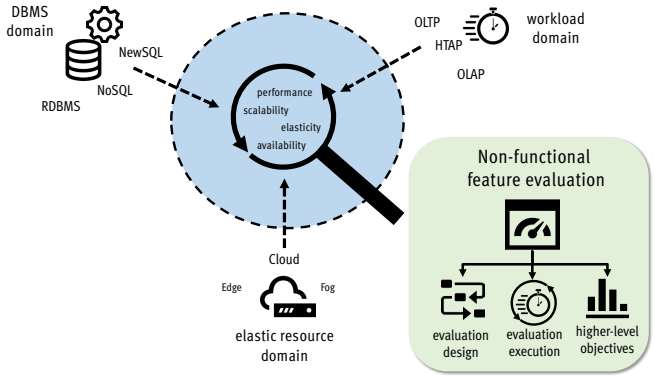


Figure 1.2: Thesis scope — reproducible evaluation design and execution for higher-level objectives

The *elastic resource domain* requires the consideration of a multitude of resource providers with heterogeneous resource types [114, 158] and varying runtime characteristics [44, 135, 101, 169]. Consequently, the applied elastic resources and their volatile runtime characteristics have a significant impact on the results of the non-functional feature evaluation [103]. While covering the extent of available resource types is challenging, analysing the impact of resource variances is an even more complex task [100, 148].

The *workload domain* requires knowledge in emulating realistic workloads that reflect the characteristics of the targeted application domain. For this purpose, the performance engineer can refer to a multitude of DBMS benchmarks [40, 82, 99, 154, 149, 146]. Yet, current DBMS benchmarks only consider the workload domain as dynamic, while assuming the DBMS and resource domain (*i.e.* the DBMS operational model) as static. This fact limits the design and execution of comprehensive evaluations as well as the evaluation of higher-level evaluation objectives.

In particular, a performance engineer requires extensive domain knowledge of all involved domains to design and execute the required non-functional feature evaluations. Yet, there is no tool support that realizes a *comprehensive evaluation methodology* by considering the domain-specific properties of the elastic resource domain, DBMS domain and workload domain. Similarly, the evaluation of higher-level non-functional DBMS features such as scalability, elasticity or availability requires concepts that go beyond the sole performance evaluation concepts of current DBMS benchmarks.

The evaluation of higher-level non-functional DBMS features demands for the execution of large-scale experiments that include runtime adaptations on DBMS and elastic resource level. While such capabilities can be provided by Cloud Orchestration Tools (COTs) [157, 207], they are not adopted for DBMS evaluations and performance engineers need to manually conduct and execute such experiments without supportive tools. This leads to unnecessarily complex, time consuming and error prone evaluation executions due to missing *evaluation automation*.

Furthermore, the missing support for evaluation automation limits the evaluation *reproducibility* that are key requirements in cloud service benchmarking [148, 201]. That is especially the case as only reproducibility enables users to keep up with the constantly evolving domains and allows the analysis of elastic resource characteristics. Yet, existing evaluations barely address the reproducibility of evaluation on elastic infrastructures [201].

1.2 Research Objectives

This thesis addresses the identified limitations of current state-of-the-art approaches for evaluating the non-functional features of distributed DBMS on elastic infrastructures, by pursuing two overarching Research Objectives (ROs), each accompanied by specific Research Questions (RQs). Within this thesis, each of these RQs is answered to achieve the respective ROs.

Establishing a comprehensive DBMS evaluation methodology that enables the reproducible evaluation of high-level non-functional DBMS features (scalability, elasticity, availability) by explicitly considering elastic resource characteristics.

We address the first objective by answering the following challenges divided into the following four RQs.

RQ-A.1: What are relevant impact factors of the distributed DBMS, elastic resource and workload domain that need to be considered in a comprehensive evaluation methodology?

RQ-A.2: Which evaluation principles of the DBMS and elastic resource domains need to be adopted by a comprehensive DBMS evaluation methodology?

RQ-A.3: What are significant metrics to evaluate higher-level non-functional DBMS features scalability, elasticity, availability?

RQ-A.4: What are the required tasks that define the non-functional, feature-specific evaluation process and enable the measurement of higher-level metrics resulting from RQ-A.3?

Research Objective B (RO-B): Providing the technical evaluation methods that enable the automated and reproducible execution of the comprehensive evaluation methodology established in Research Objective A.

We address the second objective by building upon the results of RO-A and answering the resulting challenges that are divided into the following four dedicated RQs.

RQ-B.1: How should comprehensive DBMS evaluations be specified to enable the automated execution while ensuring reproducibility and portability?

RQ-B.2: What technical concepts are required to enable the evaluation automation across the elastic resource, DBMS and workload domain, ensuring reproducibility and portability?

RQ-B.3: Which adaptation concepts are required by supportive evaluation methods to enable the evaluation of the higher-level non-functional features elasticity and availability?

RQ-B.4: How can the automation methods ensure significant results and reduce the efforts in the knowledge discovery of the non-functional DBMS features?

1.3 Research Contributions

Based on the two overarching research objectives and the accompanying research questions, this section summarizes the two core contributions of this thesis. Both contributions address the respective research questions as part of research objective A and B. All contributions build upon each other, resulting in the set of novel and highly integrated DBMS evaluation frameworks Mowgli, Kaa and King Louie. In particular, contribution I presents the results to establish the comprehensive DBMS evaluation methodology; contribution II builds upon these results and presents the novel methods for the automated performance, scalability, elasticity and availability evaluation of distributed DBMS, resulting in the Mowgli, Kaa and King Louie frameworks.

Contribution I To address RQ-A.1, a set of comprehensive analyses is carried to identify the relevant technology- and evaluation-specific impact factors [core11]. These analyses comprise the latest advances in the DBMS [core1], elastic resource [core10] and DBMS workload domain [core3]. The identified impact factors build the foundation for the methodological DBMS evaluation on elastic infrastructures. The contributions to RQ-A.2 extend this methodology by analysing evaluation principles of the separate research areas DBMS benchmarking and elastic infrastructure benchmarking. The results are applied to specify a cross-domain set of evaluation principles for evaluating DBMS on elastic infrastructures. In addition, these principles are extended with the principles orchestration and automation [core2, core9, core3] to enable the higher-level DBMS evaluation objectives scalability, elasticity and availability . The comprehensive DBMS evaluation methodology is completed by the results that address RQ-A.3 and RQ-A.4 in defining a set of objective-specific evaluation processes. Each evaluation process is defined by the required evaluation tasks and significant metrics. In consequence, four comprehensive evaluation processes are defined: performance [core10, core5], scalability [core5], elasticity [core8, core6] and availability [core4, core7].

Contribution II In the context of RO-B, RQ-B.1 is approached by defining a domain-specific evaluation model that comprises all relevant DBMS, elastic resource and workload properties as a unified Evaluation Scenario Template (EST) [core5]. An EST comprises a set of mandatory sub-templates and additional objective-specific adaptation templates to enable the specification of elasticity [core6] and availability [core7] evaluation scenarios. ESTs provide the foundation of automated DBMS evaluations and consequently, they represent the input to the evaluation automation methods that are required to address RQ-B.2 and RQ-B.3. To address RQ-B.2, this thesis provides the novel DBMS evaluation framework Mowgli [core5, data3] that fully automates the DBMS evaluation on elastic infrastructures for the evaluation objectives performance and scalability. By building upon Mowgli's automation capabilities, RQ-B.3 is approached by the frameworks Kaa [core6] and King Louie [core7]. Kaa enables the automated elasticity evaluation based on DBMS and workload adaptations during the evaluation runtime [core6]. King Louie enables the availability evaluation under consideration of elastic resource failures on different levels [core7]. Besides evaluation automation, Mowgli, Kaa and King Louie enable the reproducibility of the evaluation scenarios by ensuring the deterministic execution of the evaluation process. In order to approach RQ-B.4, Mowgli, Kaa and King Louie provide extensive data sets for each evaluation objective. These data sets contain raw performance metrics, processed higher-level objective-specific metrics and comprehensive metadata to enable advanced post-processing [data6, data2, data1, data4, data5]. The validation of Mowgli, Kaa and King Louie is based on industry-driven case studies that comprise 310 objective-specific evaluation scenarios with 1426 resulting data sets.

1.4 Thesis Outline

The remainder of this cumulative doctoral thesis is structured into two parts.

Part I summarizes and conflates the findings of the accompanying publications as follows:

Chapter 2 introduces the technical foundations of this thesis to provide a common understanding on elastic infrastructures, distributed DBMS and emerging operation models of distributed DBMS.

Chapter 3 presents and discusses related work on DBMS benchmarking, elastic infrastructure benchmarking and advanced evaluation frameworks. Moreover, an overview of related approaches for application orchestration on elastic infrastructures is provided.

Chapter 4 presents the approach to address RO-A by defining a novel DBMS evaluation methodology. It builds upon three concepts: (i) the identification of technology- and evaluation-specific evaluation impact factors of the DBMS, resource and workload domain; (ii) DBMS and elastic infrastructure benchmarking principles that are combined into a set of cross-domain evaluation principles for evaluating distributed DBMS on elastic infrastructures and (iii) the comprehensive design of objective-specific evaluation processes.

Chapter 5 pursues RO-B by implementing novel evaluation methods that build upon the established evaluation methodology of RO-A. These evaluation methods comprise a domain-specific language that enables the specification of comprehensive and reproducible ESTs. These ESTs represent the input for the novel DBMS evaluation framework Mowgli that fully automates the DBMS evaluation of elastic infrastructure for the objectives performance and scalability. The Kaa framework extends Mowgli with DBMS and workload adaptations at runtime to enable the automated elasticity evaluation of distributed DBMSs. The King Louie framework builds upon these frameworks and enables the DBMS availability evaluation in the context of elastic resource failures. These integrated frameworks provide comprehensive evaluation data sets that comprise higher-level objective-specific metrics and supportive metadata.

Chapter 6 validates the introduced frameworks by a series of objective-specific case studies in an industrial context. Moreover, the frameworks are validated against established cross-domain evaluation principles.

Chapter 7 summarizes the thesis with a conclusion and provides an outlook into future research directions in evaluating and operating distributed DBMSs on elastic infrastructures.

Part II contains the publications associated with this thesis in Chapter 8—18.

Chapter 2

Background

This chapter introduces the fundamental technical concepts on which the thesis is based on. It covers the concepts of elastic infrastructures, distributed DBMSs and operational models of distributed DBMSs on elastic infrastructures.

2.1 Elastic Infrastructures

Over the last decades, the provisioning of compute and storage resources has tremendously changed by moving from static physical resources to elastic and virtualized resources. Since the beginning of the cloud computing era in 2005 [38], resources are offered in service based manner. Moreover, resources are distributed on different levels, connected via the network. Recently, the cloud computing paradigm gets extended by the subsequent paradigms edge [178] and fog computing [125, 147]. Consequently, elastic resources have a significant impact on the operation of modern Web, Big Data and IoT applications as these applications emphasize distributed architectures and the dynamic adaptation of the application topologies to cope with varying workloads. Consequently, these resources are not only applied for stateless applications, but also for stateful applications such as distributed DBMSs [105].

A key concept of these elastic resource paradigms is the usage of virtualization technologies [56, 125]. In the following, first the background on virtualization technologies is introduced, before describing the cloud, edge and fog computing paradigm in detail. The operation of distributed DBMS on these elastic infrastructures is presented in the subsequent Section 2.3.

2.1.1 Virtualization

Virtualisation enables the abstraction of logical resources from the underlying physical resources. This allows the sharing of resources amongst multiple tenants by providing multiple logical resource entities that are mapped to the underlying physical resources. In the era of elastic infrastructures, it is distinguished between two virtualization concepts for compute resources, *Hardware Virtualization (HWV)* and *Operating System Virtualization (OSV)* [139].

For HWV, resource entities are termed Virtual Machines (VMs) and manged by a hypervisor that allocates the resources and handles the operating state. Hypervisors are classified into two categories. The *type-1 hypervisors* such as Xen [19] operate directly on the top of the host's hardware while type-2 hypervisors such as KVM [26] operate on top of the host's OS [2]. These hypervisors provide standalone VMs that are independent and isolated from the host system. This enables the operation of Windows-based VMs on top of Linux

based hosts and vice versa. The resulting trade-offs are that the VM encapsulates a full Operating System (OS) and the VM images are accordingly large in disk size. Besides, the emulation of the virtual hardware device introduces additional levels of abstraction and potential overhead [118].

OSV provides a more lightweight level of abstraction in terms of virtualization and isolation compared to HWV. OSV uses operating system features to create lightweight isolated environments, commonly known as containers. Container engines, such as the established Docker engine[1], allocate compute and storage resources. Besides, they provide access to container-specific networking services. Therefore, containers run on the same shared operating system kernel of the underlying host and multiple processes can be run within one container. By sharing the host kernel and operating system libraries, advantages of containers are reduced container image sizes compared to VM images as well as faster start-up times compared to VMs [142]. The strongest advantage of OSV is the lower performance overhead compared to HWV. On the other hand container provide weaker resource isolation capabilities compared to VMs [139].

As both virtualization concepts are established technologies, a multitude of comparative studies investigate into the performance overhead, isolation capabilities and operational aspects of multiple HWV andOSV offers [118, 139, 185, 152]. These studies primarily focus on using containers only for stateless applications while stateful applications are operated on physical or VM-based resources.

2.1.2 Cloud Computing

With the beginning of the cloud computing era in 2005 [38], cloud computing has become an important research topic and it has been adopted for operating large-scale enterprise applications [164].

Cloud service models are classified into four deployment models [56]: a *private cloud* is operated and used within a single organization, enabling full control over the physical and virtual resources; a *public cloud* is operated by an organization that offers the cloud services to public customers together with provider-specific resources and usage constraints; a *community cloud* is provisioned for an exclusive user group and operated by organizations that have shared concerns (*e.g.* research institutes); a *hybrid cloud* is a composition of at least two clouds of different deployment models that remain unique entities but enabling service and data portability via standardized technologies.

According to the National Institute of Standards and Technology (NIST) definition of cloud computing [56], cloud services share five common characteristics: *on-demand self-service* enables users to acquire cloud services directly without requiring human interaction; *broad network access* defines that cloud services are accessible over the network; *resource pooling* defines that cloud resources are pooled to enable multi-tenancy by assigning different physical and virtual resources dynamically according to the tenant demands; *rapid elasticity* enables the dynamically provisioning and releasing of cloud resources while the resource capabilities often appear 'virtually' unlimited to the user; *measured service* defines that cloud resource usage can be monitored, controlled, and reported, in order to provide transparency for the provider and the user.

Cloud providers offer heterogenous services, from compute resources to software-based services. Therefore, cloud services are classified into the three general cloud service models *Infrastructure as a Service (IaaS)*, *Plattform as a Service (PaaS)* and *SaaS* [56]. Moreover, more fine-grained classifications of cloud service models have been established [87]. In the following, a brief overview of IaaS, PaaS, Software as a Service (SaaS) and additional cloud service models is provided.

[1] `https://www.docker.com`

IaaS IaaS clouds provide compute, storage, and network resources to run arbitrary software. While modern IaaS clouds can provide physical and virtual resources to the users, virtualized resources are the common approach to increase the physical resource utilization and the number of parallel users. Hereby, VMs and containers are the common resource entities. The location of IaaS resources can be classified into geographical locations (regions), data centres inside a region (availability zones), racks inside a data centre and physical hosts inside a rack.

PaaS PaaS provides a ready-to-use application runtime environment to the user, supporting a provider-specific set of programming languages, libraries, services, and tools. With the adoption of cloud computing, the heterogeneity of additional services offered by PaaS providers has increased substantially, nowadays also comprising services such as data processing frameworks, message queues, file systems or storage systems. PaaS clouds are often operated on top of IaaS resources and in consequence, the same geographical locations can be applied to PaaS as to IaaS.

SaaS SaaS provides complete applications to the user that are accessible via client devices, *i.e.* thin client interfaces or programmatic Application Programming Interfaces (APIs). SaaS offers the lowest level of user control, as SaaS services typically only provide a limited set of application-specific configuration options. SaaS offers are often built upon IaaS or PaaS resources.

DBaaS The continuous evolvement of cloud computing is creating a variety of additional service models that are classified between IaaS, PaaS and SaaS [60, 87]. One of the noteworthy results of these evolvements is the trend to provide DBMSs as cloud services, resulting in the cloud service model Database as a Service (DBaaS) [87, 105]. While the concept of DBaaS has already been introduced before the cloud era in 2002 [18], its adoption has tremendously increased with the cloud computing evolvement. The DBaaS concept defines that the service provider is responsible for managing the DBMS and its required computational and storage resources while the DBMS is accessible to the user by DBMS-specific client interfaces. Consequently, cloud resources have become the preferred resources to operate DBaaS, as their advantages on-demand resource allocation, elasticity and 'virtually' unlimited resources are required to provide service guarantees to the customers [105].

2.1.3 Fog and Edge Computing

The previously introduced benefits of cloud computing and their service models are the preferred option for operating enterprise Web and Big Data applications. Recent IoT applications also have the demand for operating them closer to the user. This paradigm is commonly known as *edge* [140] or *fog computing* [64, 175] and it is an extension to the cloud computing paradigm.

As these paradigms are still in their early stages, their definitions are still evolving and it depends on the source, whether fog computing is either the same as edge computing [140] or it is the consolidation of cloud and edge resources [141]. In this thesis, we follow the latter definition and consider fog computing as the consolidation of cloud and edge computing [147]. Future advances of these paradigms might revise this definition. Moreover, we distinguish *cloud resources* and *edge resources*, where cloud resources always remain

in a cloud data centre, while edge resources always remain outside a cloud data centre. Similar to cloud resources, virtualization is also a key concept of edge resources [178].

According to NIST [175], fog computing services are classified in the same traditional service models as cloud computing (IaaS, PaaS, SaaS) and their deployment models are also identical, *i.e.* public, private, hybrid and community (*cf.* Section 2.1.2).

Yet, there are also distinctions from cloud computing [175, 147, 141]: *geographical distribution* as resources and service deployments are more widely distributed compared to the more centralized cloud data centre deployments; *heterogeneity* as resources are more diverse in terms of computational and storage power; *dynamic* as changes in the resource and service topology are a key concept of fog computing. Consequently, fog computing applications need to be able to exploit these dynamics and in return, they need to cope with more frequent resource outages and service interruptions [147].

2.1.4 Application Orchestration on Elastic infrastructures

The introduced elastic infrastructures raise the need for supportive tools that orchestrate distributed applications on elastic infrastructures. Resulting requirements of such tools are discussed in [core2]. They comprise features such as the support of multiple resource provider APIs to enable *multi-cloud* support; the support for cloud Domain Specific Languages (DSLs) [163] such as *e.g.* TOSCA [95] or CAMEL[add2]; and the management of the full application lifecycle including resource allocations, the deployment of the application components and runtime adaptations.

In this context numerous COTs have been established over the recent decade that focus on dedicated aspects of the aforementioned characteristics. Scientific approaches are represented by OpenTosca [79], Roboconf [120], Apache Brooklyn [130], the Orchestrator Conversation framework [184] and Cloudiator[add9, add7]. As these COTs focus on VM-based resources, container-based orchestrators including Borg, Omega and Kubernetes are presented by [123, 129]. Further COTs are discussed in [157, 191, 207].

2.2 Distributed DBMS

This section summarizes the recent advances of DBMS technologies with the focus on distributed DBMSs. Therefore, first the different data models are introduced, secondly the distribution techniques are described and thirdly, the non-functional features of distributed DBMSs are presented. Finally, the boundaries of distributed DBMSs are discussed.

2.2.1 Data Models

The evolvement of the distributed DBMSs leads to an increasing heterogeneity in DBMS data models. The data models of modern DBMS are classified into three top-level categories, ordered by their appearance on the DBMS landscape: *relational*, *NoSQL* and *NewSQL* [51, 86, 165, core1]

Relational The *relational data model* was established in the 1970s [3] and stores data as tuples forming an ordered set of attributes; which can be extended to extract more meaningful information. A relation forms

a table and tables are defined as a static, normalized data schema. RDBMS provide the standardized SQL interface [1] as generic data definition, manipulation and query language for relational data.

NoSQL Starting in the late 2000s, the *NoSQL data model* evolved as a superclass for different data models, such as key-value, document-oriented, column-oriented and graph-based [51, 86, 165]. Recently, the NoSQL data models have been extended with the time series [151, 145] and multi-model data models [198]. Common characteristics of the NoSQL data models are the adoption of flexible data models that can be schemaless and the fact that data may need to be interpreted at the application level. Consequently, there is no common query interface such as SQL for the relational data model but DBMS-specific interfaces with heterogenous query capabilities. Yet, these schemaless data models ease the distribution of data and thus they are the preferred data models for distributed DBMS. Hereafter, the NoSQL data models are briefly introduced.

The *key-value data* model relates to the hash tables of programming languages. The data records are tuples consisting of key-value pairs. While the key uniquely identifies an entry, the value is an arbitrary chunk of data. Operations are usually limited to simple CRUD (Create, Read, Update, Delete) operations.

The *document data model* extends the key-value data model by defining a structure on the values in certain formats, such as XML or JSON. These values are termed documents, but usually without fixed schema definitions. Compared to key-value stores, the document data model allows for more complex queries as document properties can be used for indexing and querying.

The *column-oriented data model* stores data by columns rather than by rows. It enables both storing large amounts of data in bulk and efficiently querying over very large structured data sets. A column-oriented data model does not rely on a fixed schema. It provides nestable, map-like structures for data items which improve flexibility over fixed schema [30].

The *graph* data model builds upon graph structures, usually including elements like nodes and edges, for data modelling. Nodes are often used for the main data entities, while edges between nodes are used to describe relationships between entities. Querying is typically executed by traversing the graph. Due to the direct connections between records, the graph data model does not facilitate distributed DBMS in contrast to the other NoSQL data models.

The *time-series* data model typically builds upon existing relational or NoSQL data models (preferably key-value or column-oriented), and adds a dedicated time-series data model on top [151, 145]. The time-series data model consists of data points which comprise a time stamp, an associated numeric value and customizable metadata. Query interfaces for the time-series data model typically offer analytical queries, covering statistical functions and aggregations.

The *multi-model* data model addresses the problem of polyglot persistence [70] which signifies that each of the NoSQL data models addresses a specific use case, including data model or even DBMS-specific query interface. Hence, multi-model NoSQL DBMSs combine different data models into a single DBMS while building upon one storage backend to improve flexibility, *e.g.* providing the document and graph data model via a unified query interface [198].

NewSQL As conventional RDBMSs provide limited data partitioning support, in the beginning of the 2010s, the *NewSQL* data model emerged [47, 72]. NewSQL DBMSs [47, 72] aim at addressing these limitations by building upon the relational data model but emphasizing a distributed architecture and supporting data distribution. Therefore, NewSQL DBMSs build upon the relational data model but mitigate relational features in

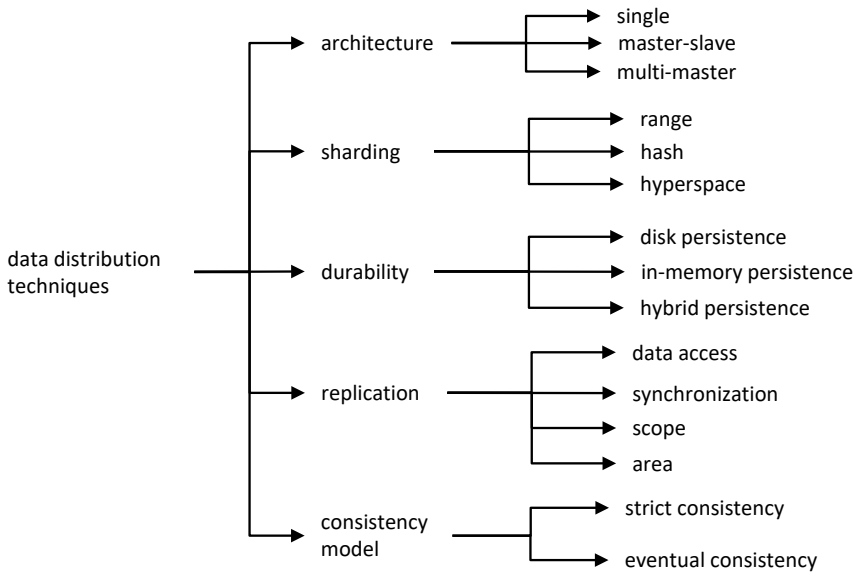


Figure 2.1: Data distribution techniques overview

favour of easing data partitioning [86]. In consequence, NewSQL DBMSs either extend RDBMSs with distribution mechanisms or they are built from scratch [137]. The NewSQL data model provides some functionalities of the relational data model, such as relations, transactions and the SQL query language. Yet, costly SQL operations such as joins might not be supported.

2.2.2 Data Distribution Techniques

The evolution towards distributed DBMS builds upon multiple data distribution techniques that can be classified into the high-level categories *architecture*, *sharding*, *durability*, *replication* and *consistency model* as depicted in Figure 2.1. For each of these categories a variety of concepts have been established which consequently lead to a heterogenous set of distributed DBMS technologies. In the following paragraphs, these concepts are introduced to provide the data distribution background that is necessary to understand the concepts of evaluation distributed DBMS on elastic infrastructures.

Architecture The hardware architecture of DBMS is classified into three types [59]: a *shared memory* DBMS runs on a single resource that consists of a collection of cores and shares a common main memory and disk system; a *shared disk* DBMS runs on disk clusters, where a collection of CPUs with private main memories share a common disk system; and a *shared nothing* DBMS does not share main memory nor disk while the DBMS instances are connected through the network. This thesis focuses on DBMSs that build upon the shared nothing architecture [5] as this architecture type is the preferred option for modern distributed DBMS [30, 42, 59].

A distributed DBMS provides a logical DBMS instance to the interacting services, where the internal DBMS instances are distributed across multiple resource entities and connected by the network. A single DBMS

instance as part of the DBMS topology is termed *DBMS node*, while the overall distributed DBMS is termed *DBMS cluster*. Consequently, a DBMS cluster comprises $n \in \{1..n\}$ DBMS nodes where each DBMS node is operated on a dedicated *resource node*. While the DBMS cluster size may change over runtime, these changes should ideally be transparent for the consuming services [51, 137].

The architecture of distributed DBMS is categorized into three general distribution models [70, 150]. A *single DBMS* comprises only one DBMS node and handles all read and write requests. The *master-slave DBMS* distribution comprises one DBMS node that represents the designated master and $n \in \{1..n\}$ DBMS nodes that act as slave nodes. Hereby, the master node handles all write and read requests as well as the synchronization of the slave nodes. Optionally, slave nodes can be configured to process read requests. In a *multi-master DBMS*, all DBMS nodes are equal, *i.e.* all DBMS nodes are processing write and read requests within the DBMS cluster.

Sharding Data sharding[2] is a core concept of distributed DBMSs to distribute data and requests across all DBMS nodes within a DBMS cluster. Therefore, the data is split into shards that are distributed across all DBMS nodes. Depending on the sharding concept, its usage requires the *manual* execution by the Database Administrator (DBA) or can be implemented in an *automated* manner by the respective DBMS. Three general sharding concepts exist: *range-sharding*, *hash-sharding* and *hyperspace-sharding*.

In the *range-sharding* approach, data items are grouped according to the continous intervals of predefined shard keys, such as the range of a unique identifier. While this approach favours scan operations, data hotspots or unbalanced shards may occur over runtime, requiring a manual revision of the shard key and consequently costly redistribution of the data across the DBMS nodes.

Hash-sharding applies a hashing function to the shard key of the data items in order to distribute them across the existing DBMS nodes. It is further classified into *simple-hashing* and *consistent-hashing* [14]. Simple-hashing uses a static hashing function such as modulo hashing that is applied to the shard key of the data items. An exemplary simple-hashing function can be based on the number of DBMS nodes in the DBMS cluster, *e.g.* $hosting\,DBMS\,node = shard\,key\,modulo\,number\,of\,DBMS\,nodes$. While this approach enables the automated sharding of data items and efficient local data lookups [25], it requires the redistribution of all data items if the DBMS cluster size changes. While the data redistribution can be executed automatically by the distributed DBMS, it is an expensive operation.

The *consistent-hashing* approach considers the scope of a hash function as multi-dimensional forms where the DBMS node identifier and the data item identifiers are randomly hashed to its positions inside the DBMS cluster [14, 25]. If a new DBMS node is added to the DBMS cluster and is hashed at position *p*, the old shard corresponding to its immediate successor is split into two new adjacent shards between *p.predecessor* and *p*, as well as *p* and *p.successor*. Consequently, consistent-hashing does not require a manual update of the the hash function and enables the automated redistribution of the data if the DBMS cluster size changes, while only a fraction of the data needs to be redistributed. Yet, the consistent-hashing does not favour data locality and joining/leaving DBMS nodes impose a high load on the neighboured DBMS nodes within the ring[165].

Hyperspace-sharding extends the consistent-hashing approach which only considers a single attribute of a data item, *e.g.* the primary key. Therefore, hyperspace-sharding includes multiple data attributes for the mapping of data items to DBMS nodes [76]. This results in a so-called *hyperspace* where the hosting DBMS node is identified by hashing each data attribute value along its corresponding dimension. While hyperspace-

[2] Sharding is also called data partitioning in the context of distributed DBMSs

sharding favours complex read operations, the hyperspace will grow exponentially with the number of data attributes. Even as first approaches to derive hyperspace-sharding function exist [97], hyperspace-sharding functions are typically application specific and require the manual definition by the DBA. Furthermore, a central coordinator service is required to assign the hyperspace parts to the respective DBMS nodes.

Durability Durability defines the DBMS persistence mechanisms regarding write and update operations. It is a crucial mechanism to address temporary DBMS node failures in case of power or network outages. For instance, if an operation is acknowledged before data is persisted, the DBMS node processing the request might fail before writing the item and data may get lost.

Therefore, durability comprises three persistence models: *disk persistence*, *in-memory persistence* and *hybrid persistence* [98, 165]. Disk persistence enforces that the data item is written to disk on all DBMS nodes before acknowledging the operation to the client. The in-memory persistence approach acknowledges write and update operations when the data item is available in memory on all DBMS nodes. Therefore it is a suitable approach for managing transient information that is needed for a limited time. Hybrid persistence first keep data items in-memory and persist them after some DBMS-specific conditions have been satisfied. Distributed DBMSs offer a variety of configuration options to specify the moment to acknowledge the operation to the client, *e.g.* after storing the data item in-memory or on disk of n DBMS nodes. Hence, DBMSs implementing the hybrid persistence allow configurations from disk persistence over custom persistence configuration (*e.g.* applying disk persistence for n DBMS nodes and in-memory persistence for m DBMS nodes) to in-memory persistence. To increase durability in case of DBMS node failures for the hybrid persistence model, additional established techniques such as write ahead logging [10] might be implemented by the respective DBMS. The hybrid-persistence model is applied by most NoSQL DBMS [150, 165].

Replication In the context of distributed DBMSs, replication defines that multiple physical copies of the same logical data item exist within a DBMS cluster. According to Wiesmann et al. [17], the primary conceptual decisions for using replication are (i) which of the physical data items may be accessed by clients and (ii) how to synchronize the remaining replicas after a modifying operation has been carried out. The first decision (i) closely relates to the architecture of distributed DBMS (*cf.* Section 2.2.2) as a master-slave architecture typically applies single-master replication [16], where a single replica represents the primary copy of the data. Only this copy can be modified by client operations while read operations might be allowed on the remaining replicas. A multi-master architecture enables write and read operations on all replicas and potential conflicts need to be resolved by the DBMS [98].

The second decision (ii) relates to synchronization of the replicas which is classified into synchronous and asynchronous replication [16, 17]. Synchronous replication ensures that the operation is executed before the client receives a response, while asynchronous replication executes the synchronization independently from the client response.

An additional replication aspect as defined by Domaschka et al. [98] is the replication *scope*. It is classified into full and partial replication. Full replication means that each DBMS node stores a full copy of the entire database. In partial replication, each data item may only be stored on a subset of all DBMS nodes. Consequently, the replication factor is smaller than the DBMS cluster size. This is a capability also closely relates to sharding (*cf.* Section 2.2.2). If sharding is used without replication, there is no tolerance against DBMS node failures. On the other hand, using replication without sharding means that all data is available on all DBMS

nodes. If sharding and replication are used in parallel, each DBMS node might contain only a subset of all data items [25].

Furthermore, modern distributed DBMS might apply different replication *areas* to enable geographically distributed DBMS clusters [80, 109]. Yet, these replications concepts are highly DBMS-specific.

Consistency Model By applying the sharding and replication concepts in distributed DBMS, the need for data consistency guarantees increases but also becomes more challenging. The view on data consistency is classified into the *data-centric* consistency on the DBMS provider side and the *client-centric* consistency on the client side [28]. The DBMS provider considers the DBMS-internal state of the (distributed) data. The client considers the DBMS as a black box and relies on a set of promised consistency guarantees that could be part of a Service Level Agreement (SLA).

With the rise of distributed DBMSs, consistency paradigms have evolved as well. Regarding the data-centric consistency, *strong consistency* concepts have been established such as strong consistency in *ACID* paradigm [4] that guarantees atomicity, consistency, isolation and durability for each request. On the contrary, *eventual consistency* guarantees [37] have been implemented as in the *BASE* paradigm: basically available, soft state, eventually consistent [31]. In general, RDBMSs and NewSQL DBMSs apply the strong consistency while NoSQL DBMS build upon the eventual consistency concepts. However, these consistency concepts only reflect two general and contrary consistency paradigms, while there are more fine-grained consistency model classifications [83, 93].

With respect to the client-centric consistency models, relational DBMS implement the conventional concepts of monotonic read consistency, read your writes consistency, monotonic write consistency or write follows read consistency [28, 37]. However, NoSQL DBMS implement heterogenous and highly customizable client-centric consistency models that do not explicitly guarantee the properties of the aforementioned consistency models. Yet, experimental results show that these consistency models can be partially fulfilled by NoSQL DBMS [48, 61].

2.2.3 Non-Functional Features

The feature set of DBMSs is classified into functional features, such as query interfaces or security mechanisms [86], and non-functional features such as performance [9]. As this thesis focuses on the non-functional features, we omit further details on functional features and refer to existing literature [51, 54, 150, 203].

The presented distribution concepts of modern distributed DBMSs enable a set of key non-functional features that are required in the context of Web, Big Data and IoT applications [86, 98, 150, 165]. This section provides a general definition of these non-functional features.

Performance Performance is typically referred to as one of the most important non-functional features of a DBMS [9]. Performance directly relates to the processing of requests and on a high level it is classified into *write* and *read* performance.

Scalability Scalability addresses the general ability to process changing workload intensities. A scalability definition for distributed DBMS is provided by Agrawal et al. [46], defining the terms *scale-up*, *scale-down*, *scale-out* and *scale-in*. Scale-up and scale-down represent the *vertically scaling* concept that is implemented

by adapting the resources of a single DBMS node. The actions scale-out (*i.e.* adding DBMS nodes to a DBMS cluster) and scale-in (*i.e.* removing DBMS nodes from a DBMS cluster) represent the concepts of the *horizontal scaling* concept. Consequently, to implement horizontal scaling, a distributed DBMS is required. As vertically scaling actions require a restart of the DBMS in order to adapt to the updated resources, these actions implicitly cause a downtime of the DBMS. Horizontal scaling actions can be executed at runtime without a downtime, if the DBMS supports *elasticity*.

Elasticity Elasticity is tightly coupled to scalability as horizontal scalability is a strict requirement to enable elasticity in the context of distributed DBMSs. While scalability targets the general ability to process arbitrary workload sizes, elasticity targets the ability to cope with sudden workload fluctuations at runtime, *i.e. "the ability to deal with load variations by adding more resources during high load or consolidating the tenants to fewer nodes when the load decreases, all in a live system without service disruption, is therefore critical for these systems."* [46].

Availability For computing systems, *availability* is defined as *the degree to which a system is operational and accessible when required for use* [7]. Besides *reliability* (the *measure of the continuity of correct service"* [21]), availability is the main pillar of many fault-tolerant implementations [98]. The availability of the DBMS can be affected by two conditions: (i) A high number of requests issued concurrently by clients, overloading the DBMS so that the requests of clients cannot be handled at all or are handled with an unacceptable $latency > \Delta t$. (ii) Resource failures occur that impact network connectivity or the compute/storage resources the DBMS is hosted on [98].

Consistency As there are numerous consistency models, their implementation in the respective DBMS are very heterogenous. Consequently, the resulting quality of the consistency guarantees depends on the DBMS-specific implementation, additional runtime configurations and external impact factors such as the workload intensity or failures [93]. Therefore, even if the consistency guarantees of a specific consistency model are expected, there is a risk of (temporal) inconsistencies such as stale data on the client side or the violation of the request execution order on data side [48, 93].

2.2.4 Boundaries: CAP and PACELC

The quality of the introduced non-functional features is dependent on the applied distribution techniques (*cf.* Section 2.2.2) but it is also limited by the following theorems, CAP [15] and PACLEC [62].

In 2000, the renowned CAP theorem has been defined [15]. It imposes the constraint that a distributed storage system can only provide two out of three properties: consistency (C), availability (A) and partition tolerance (P) [15]. Consequently, the CAP theorem enables the theoretical classification of distributed DBMSs into CA, CP or CP favouring systems. Yet, the evolvement of distributed DBMSs and elastic infrastructures hosting these DBMSs led to revisiting the CAP theorem in 2012 and refining the statement that providing two out of the three CAP properties is not a binary decision [65].

In 2010[3], the PACELC classification of distributed DBMSs has been defined [62]. The PACLEC explicitly considers the impact on latency and consistency during the normal operation and in case of failures leading to

[3] `http://dbmsmusings.blogspot.com/2010/04/problems-with-cap-and-yahoos-little.html`

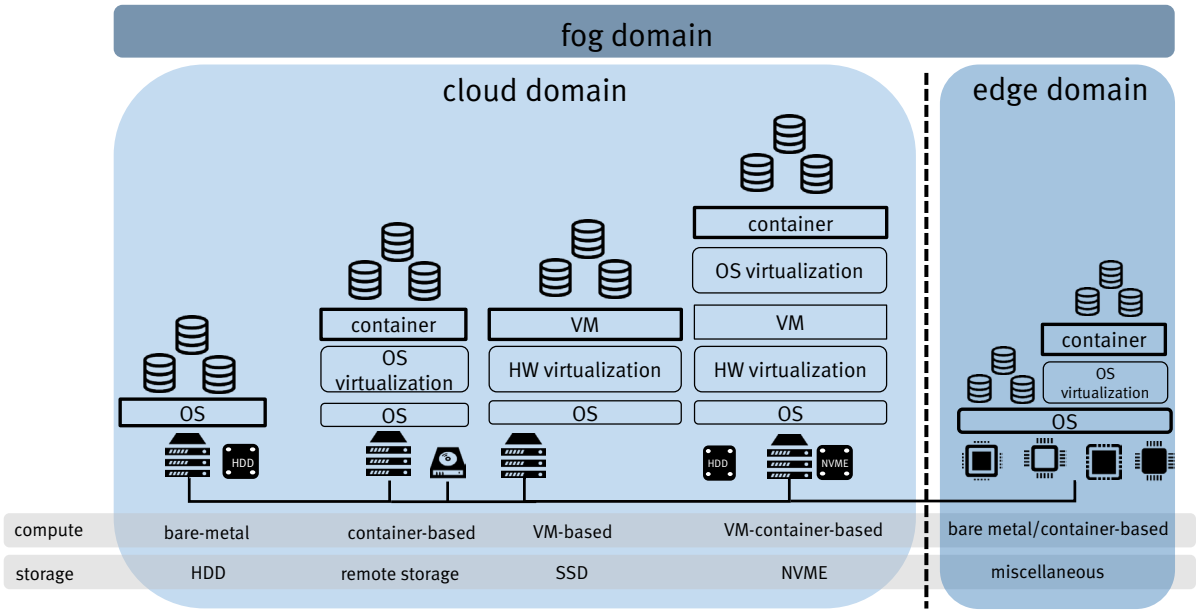


Figure 2.2: DBMS operational models on cloud, edge and fog resources

partitions. This results in the definition that in case of a partition, there is an availability-consistency trade-off. In normal operation, there is a latency-consistency trade-off. Thus, it extends the CAP classification with two possible choices in case partitions and two for normal operation. However, many distributed DBMSs cannot be assigned exclusively to one single PACELC classification and the specific PACLEC classification PC/EL is hard to assign to any distributed DBMSs [150].

As distributed DBMSs implement a variety of heterogenous distribution concepts to cater for a broad range of use cases [98, 150], the CAP theorem and PACLEC provide conceptual classifications of their consistency, availability and partition tolerance. Yet, these classifications do not provide any estimation on the quality of the presented non-functional features. Moreover, the heterogenous distribution techniques even increase the challenge of quantifying the quality of the non-functional DBMS features.

2.3 Operational Models of Distributed DBMS

In order to fully exploit the advances of distributed DBMSs such as horizontal scalability and elasticity, similar concepts are required on the resource level. As elastic resources provide these concepts (*cf.* Section 2.1), the operational models of distributed DBMSs have changed from dedicated bare-metal deployments to manifold deployments on virtualized resources. Figure 2.2 depicts a non-exhaustive set of common operational models for distributed DBMSs on elastic infrastructures.

The compute resources of the cloud domain range from dedicated bare-metal resources, container-based resources, VM-based resources to a combination of VM- and container-based resources [199]. And in the context of the overarching fog domain, edge resources provide an additional set of heterogenous resources that can be applied for distributed DBMSs [209]. Apart from compute resources, the storage backends are

of major importance for DBMSs and the variety of offered storage backends ranges from dedicated Hard Disk Drive (HDD), Solid State Drive (SSD) and Non-Volatile Memory Express (NVME) devices to remote storage. In consequence, we define the DBMS operational model as follows:

Definition D.1 (DBMS operational model)**.** *The DBMS operational model is defined as the DBMS, its runtime configuration and the applied elastic resources. In consequence, the operational model of distributed DBMSs on elastic infrastructures needs to be determined from RDBMS, NoSQL DBMS, NewSQL DBMS and elastic resource offers. The runtime configurations comprise heterogeneous DBMS- and distribution-specific options.*

With the flexibility offered by these manifold operational models, there comes a number of challenges that need to be addressed to operate distributed DBMSs on elastic infrastructures. Starting from the placement of the virtual resource onto the physical resources [117, 138] over selecting the best matching resource based on application requirements [158, add7] to determining the required number of instances and runtime configuration to satisfy the application requirements [159, 177]. While these challenges hold for stateless and stateful applications, there are numerous additional challenges that need to be considered for distributed DBMSs specifically.

Already the initial resource selection for distributed DBMS becomes a more important decision compared to stateless applications. Stateless applications can be easily migrated to new resource types but the migration of a distributed DBMS is way more costly in terms of time and temporary service degradation [168]. Moreover, the selection of the storage backend is of crucial importance for distributed DBMSs [107, 169, 205].

As elastic resources build upon shared resources, the impact of resource interferences caused by other applications running on the same physical resources[4] needs to be considered when selecting the DBMS operational model [44, 136], especially as container- and VM-based resources vary in their provided resource isolation capabilities [139].

These resource-specific challenges are extended with the heterogeneity of distributed DBMS technologies and the corresponding runtime configurations (*cf.* Section 2.2.2). As these decisions are dependent on each other [89], the selection of an optimal DBMS operational model requires the correlation of the resource characteristic, the distributed DBMS and its runtime configurations.

In consequence, already the following basic operational model decision becomes a challenging task:

"Does DBMS A provide better performance if operated with 3 nodes at resource provider X on resource VM_medium or if operated with 5 nodes at resource provider Y on resource VM_SMALL?"

While the exemplary question only focuses on the objective performance and omits several DBMS- and resource-specific characteristics, it already requires dedicated elastic resources and DBMS domain knowledge to address this question. Yet, even with the required domain knowledge, it is not possible to provide a well-grounded answer without supportive decision-making tools in the form of benchmarks. There are supportive benchmarks with the dedicated focus either on elastic resources (*cf.* Section 3.3) or distributed DBMS (*cf.* Section 3.2). But these benchmarks do not provide a unified approach by combining the resource and DBMS domain and consequently, they hinder the correlation of resource characteristics with the DBMS and its runtime configuration. Therefore, this thesis contributes in providing a unified DBMS evaluation approach across the resource and DBMS domain.

[4] Also called the Noisy Neighbour effect.

2.4 Summary

This chapter presented the technical fundamentals of the elastic infrastructure domains cloud, edge and fog. It summarized the respective virtualization concepts, resource characteristics, service models and orchestration concepts. These infrastructures build the base for operating modern distributed DBMSs. Furthermore, the technical background on distributed DBMS was introduced, covering the relational, NoSQL and NewSQL data models; distributed DBMS architectures and key distribution techniques such as sharding, durability, replication and consistency models. These techniques enable the non-functional features performance, scalability, elasticity, availability and consistency that are key requirements of data-intensive applications. Therefore, an established definition of each non-functional feature was provided, including the discussion of their boundaries within the CAP and PACELC theorem. These technical concepts from the basis for the methodological DBMS evaluation that is presented in Chapter 4. In order to combine elastic resources with distributed DBMSs into their operational model, common operational models of distributed DBMS were presented for the cloud, edge and fog domain; including relevant resource characteristics that need to be considered. In conclusion, the challenges in the selection of a concrete DBMS operational model were discussed, including elastic resource and distributed DBMS characteristics as well as their interdependencies. In Chapter 5, we use these concepts to provide comprehensive evaluation methods that enable the automated evaluation of distributed DBMS on elastic infrastructures by explicitly considering DBMS and resource characteristics.

Chapter 3

Related Work

This chapter summarizes related work on DBMS and elastic infrastructure benchmarking that is part of the contributing publications of this thesis [core1] included in chap 8, [core2] included in Chapter 9, [core3] included in Chapter 10, [core4] included in Chapter 11, [core5] included in Chapter 12, [core6] included in Chapter 13, [core7] included in Chapter 14, [core8] included in Chapter 15, [core9] included in Chapter 16, [core10] included in Chapter 17 and [core11] included in Chapter 18.

This thesis provides novel concepts for evaluating the non-functional features of distributed DBMS on elastic infrastructures. Therefore, we discuss related DBMS benchmarks and elastic infrastructure benchmarks in following sections. In particular, Section 3.1 defines an evaluation terminology, Section 3.2 presents established DBMS benchmarks with the focus on distributed DBMS and Section 3.3 presents elastic infrastructure benchmarks. In Section 3.4, advanced evaluation approaches are discussed that target the evaluation of non-functional features apart from performance. Section 3.5 summarizes the related approaches in scope of this thesis by discussing overlapping and limiting aspects.

3.1 Terminology

The evaluation of DBMSs and elastic resources is a widely discussed research topic and in consequence, a diverse set of terms for the involved systems and tools have been established [9, 94, 100, 148, 201, 189, 190]. This thesis builds upon the DBMS operational model definition (*cf.* Section 2.3) and applies the following terminology:

Definition D.2 (Workload)**.** *The workload defines the lowest common denominator in the non-functional features evaluation toolbox and it is required to measure the non-functional features of an operational model. Therefore, a workload emulates application-specific load patterns as realistically as possible.*

Definition D.3 (Benchmark)**.** *The workload is generated by a benchmark that adds additional features such as the metric reporting, metric processing and the coordinated execution of multiple workloads. A benchmark can provide different workload types.*

Definition D.4 (Evaluation framework)**.** *An* evaluation framework *represents an advanced concept for evaluating the non-functional features as it orchestrates the DBMS operational model and n benchmark instances. Therefore, an evaluation framework supports the evaluation automation to a certain degree, ranging from the allocation of elastic resources over the deployment of the DBMS operational model to adaptations during the evaluation execution.*

Definition D.5 (Evaluatuation objective)**.** *An evaluation objectives defines, which non-functional feature is to be evaluated. The standard evaluation objective for DBMSs is performance.*

Definition D.6 (Higher-level evaluatuation objective)**.** *A higher-level evaluation objectives builds upon the performance objective by incorporating additional DBMS and runtime aspects. Higher-level evaluation objectives are scalability, elasticity, availability and consistency.*

3.2 DBMS Benchmarking

DBMSs are a key component of each data-intensive application domain. Consequently, numerous DBMS benchmarks have been established over the last decades to evaluate the non-functional features of DBMSs and represent a foundation of this thesis. Hereafter, we focus on DBMS benchmarks of the Web, Big Data and IoT domain, renowned Transaction Processing Performance Council (TPC) benchmarks that have been presented before the cloud era [119]. These DBMS benchmarks focus on the evaluation objective performance under consideration of various workload types. We classify the workload types into *Online Transaction Processing (OLTP)*, *Online Analytical Processing (OLAP)* and *Hybrid Transaction-Analytical Processing (HTAP)*. There are more fine-grained and diverse classifications for DBMS workloads. For instance, the work of Friedrich et al. [99] considers workloads for NoSQL DBMS as OLTP while others define that OLTP is only applicable for RDBMS [107], considering NoSQL workloads as a separate category.

The specification of workloads is either based on *trace-based* application data or *synthetic* data, which is modelled after real-world load patterns and generated by a set of workload-specific constraints [148]. Synthetic workloads provide more customization options which decrease the possibility that evaluation targets are optimized for a specific workload type at design time. Yet, synthetic workloads also include potential randomness, as workload constraints may build upon probabilistic decisions, *e.g.* determining the distribution key of the zipfian distribution [11]. Trace-based workloads are more significant as they are created from real application data and enable the deterministic execution. But they provide only limited customization options and the evaluation targets can be optimized at design time for well-known trace-based workloads.

An additional workload concept represents the *workload generation model* that is classified into *closed*, *open* and *partially open* [24]. The closed model defines that new requests are only issued after all current ones have been processed, *i.e.* following a sequential order. Its implementation is rather simple builds upon a static thread pool to issue the requests [148]. To support fluctuating workloads, an *open* or at least *partly-open* workload generation model is required [148]. The open workload model issues the request by following probability distributions and individual requests are completely decoupled. Yet, the implementation of this approach bears some challenges such as the need for a dynamic thread pool to scale fast enough and not to disturb the scheduling process and it requires thorough monitoring [148]. The partly-open addresses the challenges of the closed workload model by issuing requests either on a probabilistic distributions or on a sequential order. Yet, its implementation requires precise scheduling and it is hard to determine how long a scheduled thread will be busy [148], Moreover, it might overload either the workload generation instance or the target DBMS [148].

Within this thesis we build upon these workload concepts and their generating benchmarks to enable the evaluation of higher-level evaluation objectives by considering the volatility of elastic infrastructures. Therefore we provide in the following sections an overview of established benchmarks classified by their workload

type. It is noteworthy that this thesis focuses on OLTP and partially HTAP workloads, while its methodology is independent of a specific workload type or benchmark.

3.2.1 OLTP

OLTP workloads represent the typical operational requests of each application domain. Consequently, OLTP workloads comprise independent, short and repetitive request patterns based on arbitrary data sizes [13]. The TPC benchmark provides standardized OLTP workloads for different application domains, such as the TPC-C [45] workload for the e-commerce context or the TPC-E [122] workload for financial context. The TPC council only provides the workload specification, whereas the implementation of the benchmark as workload generator is the responsibility of the user. The synthetic TPC workloads compute not only the raw performance metrics but also more significant performance metrics such as efficiency. Yet, these workloads target only RDBMS, distributed NoSQL and NewSQL DBMS are not explicitly considered in their specification.

The BigBench [85] benchmark builds upon the TPC-DS [214] workload specification, focusing the Big Data context based on synthetic data. By now, it is incorporated by the TPC and released as the TPCX-BB [206]. BigBench also focuses on the relational data model with extensions for Big Data tools that offer an SQL interface on top of file systems, such as Apache Hive[1]. BigBench provides DBMS performance metrics as well a composed metrics that target Big Data processing frameworks [92].

The OLTP-Bench [82] comprises 15 workload classes of different workload types for RDBMS, including trace-based and synthetic workloads. These workloads cover the Web, financial and e-commerce context. OLTP-Bench provides the common performance metrics and composed metrics by correlating system metrics with performance metrics. Apart from that, this benchmarks provides support in adapting the workload intensities at runtime and advanced metric processing.

The widely adopted Yahoo Cloud Serving Benchmark (YCSB) [40] targets the performance evaluation of a variety of NoSQL and RDBMS. To enable the support for a wide range of data models, it builds upon simple Cread, Read, Update, Delete (CRUD) and scan requests. In addition, it provides a predefined set of workloads with a variety of workload- and DBMS-specific configuration options. Its modular architecture favours extensibility and it has been extended by the YCSB++ [57], YCSB-T [96] and the GeoYCSB [197].

YCSB++ [57] adds the evaluation of data consistency with the focus on staleness. Moreover, it extends the basic requests with support for transactional and more complex request types. It focuses only on the column-oriented data model. Moreover, it adds support for the distributed execution of multiple YCSB++ instances.

The YCSB+T [96] extends the YCSB with transactional workloads that enrich the performance measurements and add consistency-centric evaluation. Therefore, it measures the transactional overhead and potential violations of the requests ordering.

GeoYCSB [197] extends YCSB with the support for geospatial data. Therefore, it integrates new components to its architecture and extends the supported workload classes with geospatial workloads for different NoSQL DBMSs.

The BG benchmark [77, 161] also builds upon YCSB, focussing on the evaluation objectives performance and consistency. Therefore, the BG benchmark enables requests that reflect the workload of a real social media platform. Moreover, it enables to support not only the measurement of performance, but also the specification and validation of performance and consistency SLAs that need to be fulfilled by the target DBMS.

[1] https://hive.apache.org/

Additional OLTP workloads for NoSQL DBMS are summarized in [99, 154], also including dedicated workloads for graph-based NoSQL DBMS. As graph-based DBMS do not necessarily provide a distributed architecture (*cf.* Section 2.2.1), we omit the discussion of these benchmarks in this section. Yet, the results of this thesis can also be applied for graph-based DBMS with the respective workloads.

3.2.2 OLAP

OLAP workloads target the decision support and focus on consolidated data rather than single requests. In this regard, the data size is significantly larger than for OLTP workloads. The requests are read heavy with mostly ad hoc and complex requests, accessing millions of records and performing joins and aggregations [13].

The TPC provides a set of OLAP workload specifications with the latest version of the TPC-H workload [186]. This workload specification simulates the industry-driven decision support based on a high volume of data and complex requests. It measures the performance with the TPC-H specific composite metric query-per-hour. While the workload specification targets the relational data model, there are implementations for NoSQL DBMS [67, 111].

An OLAP workload implementation with the focus on NoSQL DBMS is provided by SSB+ [111]. It extends the established OLAP star schema benchmark (SSB) [36] with the support for two NoSQL DBMS. Therefore, the required data is generated in DBMS-specific formats and it supports the distribution of the data to ensure the scalability of the workload itself compared to the original SSB.

3.2.3 HTAP

HTAP workloads represent the latest workload category by combining OLTP and OLAP characteristics into the HTAP workload class [71, 144]. While OLTP and OLAP workloads are typically handled by separate data management systems, HTAP workloads define OLAP requests over operational data, while defining OLTP requests on the same data. In consequence, HTAP workloads enable to address the research challenge to which degree are OLAP operations decreasing the OLTP Performance and vice versa [12].

A first approach towards HTAP benchmarks is presented by CH-benCHmark [52] that combines the execution of OLTP workload TPC-C [45] and the OLAP workload TPC-H [186]. A similar approach is pursued by HTAP-Bench [149], also building upon the TPC-C and TPC-H workloads. Moreover, a unified metric for HTAP systems is defined that considers the execution of constantly increasing OLAP requests in correlation to the impact on OLTP performance.

Additional approaches for the combination of OLTP and OLAP workloads are presented by [29, 49, 63, 50]. Yet, these approaches are focusing solely on the relational data model and consequently only on RDBMS and NewSQL DBMS, while NoSQL DBMS also claim to be a suitable alternative for HTAP processing [195].

3.3 Elastic Infrastructure Benchmarks

With the evolvement of elastic infrastructures cloud, edge and fog, the need for elastic resource benchmarks increases. Consequently, a number of elastic infrastructure benchmarks have been established over the last decade, enabling the comparative evaluation of existing resource offers. As elastic resources represent a

crucial part of the DBMS operational model (*cf.* Section 2.3), the concepts of elastic infrastructure benchmarks are a foundation of this thesis.

3.3.1 Cloud Benchmarks

Already web applications led to a new set of benchmarks that focuses on the evaluation of distributed applications such as the RuBiS benchmark [20] or SPECWeb[2]. Yet, the characteristics of cloud computing, such as shared resources or fluctuating workloads, demands for new resource-centric benchmark approaches in the late 2000s [32, 34].

As one of the first cloud resource benchmarks, Cloudstone [32] enables the evaluation of multiple cloud resources by providing a distributed web application for its benchmark. It measures performance and provides a composed cost metric. The Cloudstone benchmark stresses the need for automation to enable large-scale evaluations in the cloud. Therefore, it provides a first toolset to automate the evaluation on the AWS Elastic Compute Cloud (EC2).

A similar approach is presented by the CloudCmp benchmark [43], enabling the comparison of different public and private cloud providers based on performance and costs. Therefore, CloudCmp provides a set of web applications, comprising a storage-intensive one, a compute-intensive one and a latency-sensitive application. The scope of CloudCmp considers all kinds of IaaS resources, *i.e.* compute, storage and network.

The Rain benchmark [39] provides an extensive set of open, closed and partly open workloads [24] for cloud-hosted web applications. It benchmark targets a dedicated exemplary web application, but can also be applied for generic distributed applications. Yet, it does not consider elastic resource characteristics nor does it provide any support to automate the evaluation on elastic resources.

Benchmarks with explicit focus on performance and additional aspects, such as the provisioning times of VM types of different cloud providers are presented by [110, 131, 182, 183, 189] and the SPEC Cloud® IaaS benchmark[3]. These benchmarks comprise workloads of different cloud application types such as web, high performance computing or financial applications.

The applied applications in such benchmarks are often represented by a 3-tier web application or microservice applications such as TeaStore [192] and they are operated on a set of elastic resources.

Approaches that focus not only on VM-based resources but also on container-based resources are provided by [115, 118, 155, 139]. These benchmarks evaluate the performance overhead of VM-based applications in relation to container-based applications under consideration of compute and storage intensive workloads. Besides performance, resource isolation is an important evaluation objective for container-based applications. This evaluation objective is addressed by multiple approaches [139, 181, 124], comparing the resource isolation capabilities of VMs with containers, partially focusing on container-based DBMS.

3.3.2 Edge and Fog Benchmarks

Even as the Edge and Fog resource offers are in an early stage, first resource- and application-centric benchmarks have been established.

The FogExplorer [171] considers infrastructure and application design options for operating IoT applications on fog resources. Thus, the focus relies on the simulation of operational model parameters such as resource

[2] https://www.spec.org/web2009/
[3] https://www.spec.org/cloud_iaas2018/

capacity, locality or number of instances [172]. As the approach is purely simulation-based, it does not support the evaluation of real applications on fog resources.

The DeFog[200] benchmark addresses the comparative performance evaluation of applications that are deployed on cloud and edge resources. Therefore, it proposes a standardized methodology with a set of significant metrics. DeFog comprises a set representative fog workloads with workload-specific metrics.

3.4 Advanced Evaluation Frameworks

Besides the performance-centric DBMS and infrastructure benchmarks, there are approaches that address the evaluation of higher-level evaluation objectives as scalability, elasticity or availability.

BenchFoundry [146] provides a DBMS benchmark for trace-based workloads. Apart from that, it defines and implements a set of additional features that go beyond the capabilities of sole DBMS benchmarks. These features comprise a DSL to specify workloads, post-processing capabilities to enhance the analysis of raw performance results and the support for distributed workload execution. However, the focus of BenchFoundry is the workload execution while the DBMS operational model is not in its scope.

The DBMS evaluation framework presented by Klems et al. [69, 88] focuses on the objective DBMS scalability on elastic infrastructures by considering DBMS-specific consistency mechanisms. in line with our approach, the framework of Klems et al. [69] emphasizes automation by orchestrating the resource allocation and DBMS deployment. Yet, this framework does not focus on the characteristics of elastic resource characteristics nor does it support runtime adaptations to evaluate the objectives elasticity and availability.

The Bungee framework [116] addresses the elasticity evaluation of auto-scalers for cloud-hosted applications. In particular, it evaluates elasticity in the context of accuracy of auto-scaling frameworks. While such auto-scaling frameworks can be applied for distributed DBMS, their evaluation target is not the evaluation of the DBMS-specific elasticity capabilities. Yet, these capabilities need to be throurougly evaluated to apply auto-scaling for distributed DBMS [165].

A DBMS-centric elasticity evaluation framework is presented by [55] that enacts DBMS adaptations at runtime to measure the performance development during scale-out/-in adaptations at runtime, *i.e.* elastic scaling. While the framework automates these elastic scaling actions, it does not automate the full evaluation process nor does it consider the characteristics of elastic resources.

With respect to DBMS availability, a first approach is presented by Gao et al. [170] that analyses existing DBMS bugs regarding failure recovery mechanisms. Yet, the study focuses only on the DBMS layer and does not consider potential failures of elastic resources nor their impact on the availability of the DBMS cluster.

The Gremlin approach [133] presents a failure injection framework for evaluating the availability of microservices. Hereby, a failure model describes the potential failures on the network level and injects them into a running system, analysing the latency impact on the client side. Yet, Gremlin only considers stateless applications network failures, without focusing on stateful services and resource failures.

3.5 Summary

This chapter summarized related benchmarking approaches for DBMSs and elastic infrastructures.

Regarding DBMS benchmarking, numerous benchmarks were presented that target the evaluation of DBMSs for dedicated data models, *i.e.* relational, NoSQL or NewSQL. These benchmarks offer a variety of

OLTP, HTAP and OLAP workloads that range from synthetic to trace-based workloads. They target the evaluation objective performance by assuming a given DBMS operational model and consequently their methodology only comprises the workload domain, excluding DBMS runtime parameters and elastic resources as depicted in Figure 3.1. In consequence, these benchmarks provide the foundation for higher-level evaluation objectives such as scalability, elasticity or availability, but do not provide a methodical approach for their evaluation nor do they consider the characteristics of elastic resources.

Elastic infrastructure benchmarks target the performance of elastic resources by assuming dynamic operational models. This results in the explicit consideration of elastic resource characteristics, such as virtualization technologies, resource interferences or different storage backends. In this context, numerous benchmarks with custom workloads were presented. These benchmarks either target a dedicated elastic resource or exemplary 3-tier web applications. As elastic infrastructure benchmarks focus on evaluating the performance of dedicated resources or the overall application performance, evaluating the dedicated performance of distributed DBMS under the consideration of elastic infrastructure characteristics is not in their scope. Moreover, the orchestration of elastic resources to evaluate higher-level non-functional features such as scalability and elasticity is not considered by the presented elastic infrastructure benchmarks.

The focus of both benchmarking areas relies on the evaluation objective performance and consequently, they provide comprehensive performance evaluation methodologies. Yet, higher-level evaluation objectives are not in the scope of the presented benchmarks even if some benchmarks outline scalability and elasticity methodologies. Yet, these methodologies are not explicitly supported by the respective benchmarks nor do these methodologies address the characteristics of elastic infrastructures.

There is a small number of advanced evaluation frameworks that target the isolated evaluation of higher-level evaluation objectives by providing required orchestration capabilities on resource or application level. But these frameworks address only the evaluation of a single higher-level evaluation objective and their focus either lies solely on the resource level or on stateless applications.

In order to address the shortcomings of the presented approaches, we argue that a novel methodology for evaluating distributed DBMS on elastic infrastructures is required that addresses the shortcomings of the presented benchmarking approaches: (i) the consolidation of the orthogonal DBMS and elastic infrastructure benchmarking concepts; (ii) support for higher-level evaluation objectives; (iii) explicit consideration of the DBMS operational model.

In order to address these challenges, this thesis builds upon three pillars: the presented research areas of DBMS and elastic infrastructure benchmarking, and the orchestration of distributed applications (*cf.* Section 2.1.4) as depicted in Figure 3.1. We argue that performance evaluation methodologies of the orthogonal DBMS and elastic resource benchmarking areas need to be consolidated and extended to enable comprehensive DBMS evaluations on elastic infrastructures as depicted in Figure 3.1. The required methodology needs to be extended with DBMS distribution characteristics to enable the direct correlation with these distribution characteristics and elastic resource characteristics. Besides, resource and application orchestration concepts need to be included in this methodology to enable runtime adaptations that are a crucial requirement for the higher-level DBMS evaluation objectives scalability, elasticity and availability. The methodological contributions to address these challenges are presented in Chapter 4. Chapter 5 presents novel evaluation methods that build upon these concepts and enable multi-objective DBMS evaluations for distributed DBMS on elastic infrastructures, including supportive methods to automate the full evaluation process.

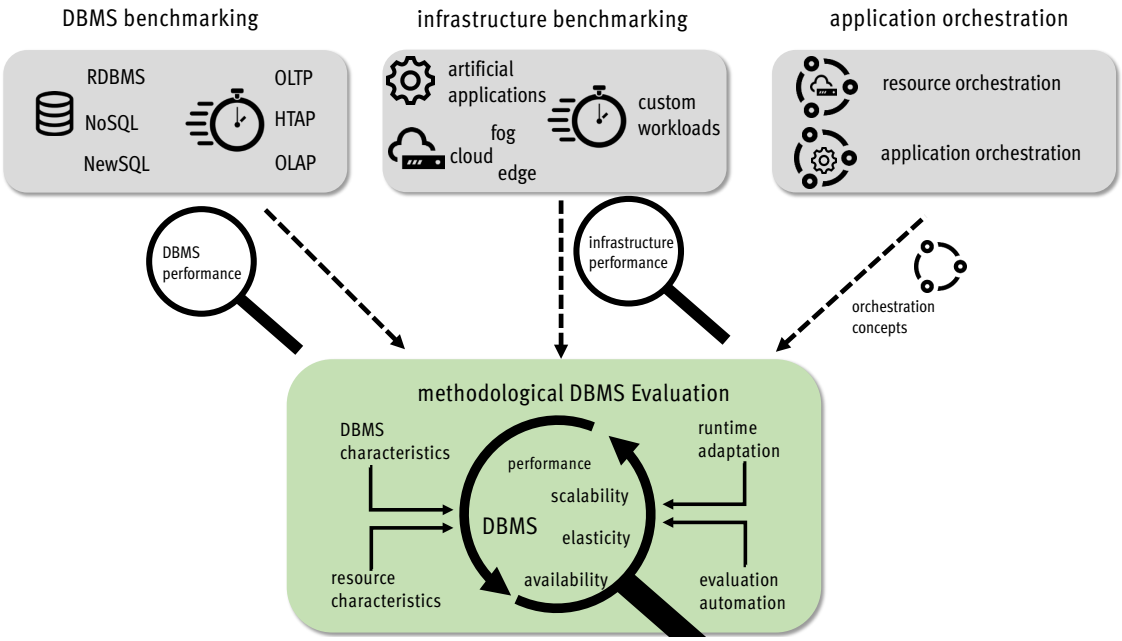


Figure 3.1: Related research concepts

Chapter 4

Methodological DBMS Evaluation

This chapter summarizes selected findings of the publications [core1] included in chap 8, [core2] included in Chapter 9, [core3] included in Chapter 10, [core4] included in Chapter 11, [core5] included in Chapter 12, [core6] included in Chapter 13, [core7] included in Chapter 14, [core8] included in Chapter 15, [core9] included in Chapter 16, [core10] included in Chapter 17 and [core11] included in Chapter 18 that contribute to the novel evaluation methodology for distributed DBMSs on elastic infrastructures. In particular, these publications address the first research objective:

RO-A: *Establishing a comprehensive DBMS evaluation methodology that enables the reproducible evaluation of high-level non-functional DBMS features (scalability, elasticity, availability) by explicitly considering elastic resource characteristics.*

Based on the identified shortcomings of related DBMS and elastic infrastructure benchmarks presented in Chapter 3, we provide a comprehensive evaluation methodology for evaluating distributed DBMSs on elastic infrastructures. In particular, the following shortcomings are addressed.

Separated Evaluation Domains DBMS and elastic infrastructure benchmarks are evolving in orthogonal directions. DBMS benchmarks primarily focus on the evaluation objective performance under the consideration of manifold workload models while assuming a static DBMS operational model. Elastic infrastructure benchmarks focus on resource characteristics by applying resource-driven workloads on entire distributed applications without considering the characteristics of distributed DBMS.

Separated Evaluation Principles Besides the manifold benchmarks of the DBMS and elastic infrastructure domains, an extensive number of evaluation guidelines have been established that define the key principles for significant DBMS [9, 94, 180, 190] and elastic infrastructure [68, 100, 148, 201] evaluations. While these principles are dedicated to the DBMS or elastic infrastructure domains and partially overlap, a cross-domain view on these principles is missing.

High-level Evaluation Objectives The advances of distributed DBMSs and elastic infrastructures enable non-functional features that go beyond performance, *i.e.* scalability, elasticity and availability. In consequence, such high-level non-functional features need to be addressed as evaluation objectives. Yet, existing benchmarks mainly focus on performance and only a small set of evaluation frameworks target high-level evaluation objectives. Those which do so, consider only a limited subset of these high-level evaluation objectives and their primarily focus lies on stateless applications.

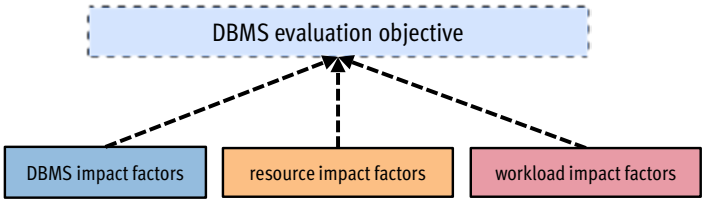


Figure 4.1: Evaluation impact factors

The findings that address these limitations and establish a comprehensive DBMS evaluation methodology are presented in the following: Section 4.1 derives the domain-specific impact factors that need to be considered for comprehensive DBMS evaluations on elastic infrastructure. Section 4.2 aligns DBMS and elastic infrastructure benchmarking principles, and Section 4.3 defines the fine-grained evaluation process for the evaluation objectives performance, scalability, elasticity and availability.

4.1 Evaluation Impact Factors

In general, large software systems offer an extensive number of configurable parameters which can have direct impact on the non-functional features [74, 121, 126] (*cf.* Section 2.2.3). In the context of DBMSs operated on elastic infrastructures, the plethora of configuration options are a result of the continuously evolving elastic infrastructure and DBMS domains. As these configuration options need to be considered in the DBMS operational model and consequently in the evaluation of the non-functional DBMS features, they are referred to as *DBMS* and *resource impact factors* in this thesis. Besides, the specification of the application-specific workload adds a third impact factor category that needs to be considered for comprehensive DBMS evaluations. In order to define a comprehensive evaluation methodology, the findings of this section address the following research question:

RQ-A.1 *What are relevant impact factors of the distributed DBMS, elastic resource and workload domain that need to be considered in a comprehensive evaluation methodology?*

The identified impact factors are classified into three main categories that need to be considered when designing DBMS evaluations for elastic infrastructures as depicted in Figure 4.1.

Each of these impact factor categories is further classified into their domain-specific impact factors. Thus, it is distinguished between *technology-specific* impact factors and *evaluation-specific* impact factors. Technology-specific impact factors are predefined by the considered technologies for designing the evaluation, *e.g.* the DBMS data model [86] or the elastic resource types [114], but also external events that cannot be specified during the evaluation design, such as resource variations [44] or resource failures [132]. *Evaluation-specific* impact factors offer a configurable but technology-specific range of configuration options that need to be considered in the evaluation design.

While we address the evaluation of non-functional DBMS features on elastic infrastructures, studies that evaluate the functional features of DBMS [51, 54, 86, 150, 165, 203] and elastic resources [58, 73, 108, 134, 176,

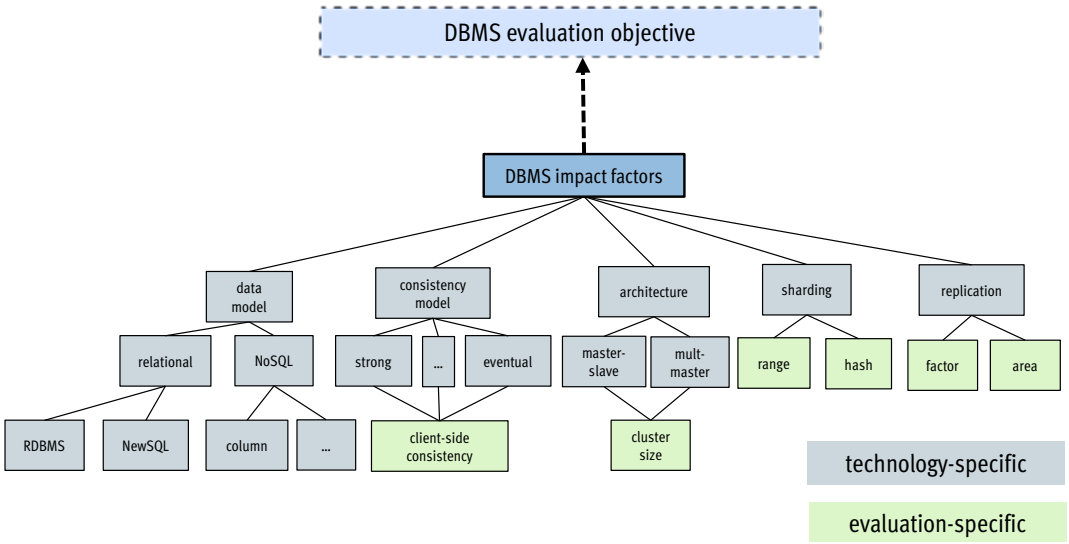


Figure 4.2: DBMS impact factors

add7] are not in the scope of this thesis. But these studies contribute with their results to our classification of technology-specific the impact factors.

The resulting classification represents a systematic overview on general domain-specific impact factors that need to be mapped to concrete domain technologies during the evaluation design. While this classification represents the current state-of-the-art of impact factors, all three domains are constantly evolving and the classification might require extensions with future resource types, DBMS concepts or workload types. Hereafter, we present the key technology- and evaluation-specific impact factors for each domain.

DBMS Impact Factors The DBMS impact factors are guided by the data distribution techniques (*cf.* Section 2.2.2), resulting in the classification *data model*, *consistency model*, *architecture*, *sharding* techniques and *replication* [core1, core11] as shown in Figure 4.2.

The data model represents a technology-specific impact factors and its selection is a central decision in the DBMS selection process [203]. Consequently, the number of DBMS technologies for designing the non-functional feature evaluation is limited by the eligible data models.

This also applies for the consistency model where distributed DBMSs implement heterogenous consistency models, resulting in a multitude of technology-specific impact factors. Depending on the DBMS and the consistency model, there are DBMS-specific *client consistency settings* that offer a range of evaluation-specific configurations.

The supported architectures are DBMS-specific but the actual *cluster size* represents an evaluation-specific impact factor. For distributed DBMS architectures, sharding and replication are DBMS-specific impact factors that offer a set of evaluation-specific configuration options. For instance, *range-* and *hash-based* sharding strategies, and for replication, a configurable *replication factor* and *replication area* for cluster- or geo-replication.

Elastic Resource Impact Factors The elastic resource impact factors are guided by the concepts of elastic infrastructures (*cf.* Section 2.1). The impact factors are classified into *provider type*, *virtualization type*, *operating system*, *resource capacity*, *locality* and *interference* [core10, core5, core11] as depicted in Figure 4.3.

The provider type represents a technology-specific impact factor that narrows down the eligible resource providers for the evaluation design. For each resource provider, the impact factors virtualization type, resource capacity and locality comprise a set of evaluation-specific impact factors that offer a provider-specific range of configuration options. The virtualization type is distinguished into *bare metal*, *hardware virtualization (HW)* and *operating system (OS)* virtualization. The impact factor resource capacity is distinguished into the evaluation-specific impact factors *compute*, *i.e.* number of cores and memory size, *storage*, *i.e.* storage type and capacity, and *network*, *i.e.* network bandwidth. It is noteworthy that identical resource capacities do not necessarily result in identical resource performance if the resources are virtualized. This is because the underlying hardware can be different even within the same resource provider.

The locality impact factor defines the locality of the resources to operate the distributed DBMS and is further classified into *cloud*, *edge* and *fog*. The cloud locality is distinguished into the evaluation-specific impact factors *server*, *rack*, *zone* and *region*. Regarding the edge and fog locality, there has not yet been a common classification for location entitles established (*cf.* Section 2.1.3).

The interference impact factor represents technology-specific impact factors that are dependent on the selected resource provider. In consequence, they are not directly configurable during the evaluation design, but need to be considered as impact factor. The interference impact factors are classified into *noisy neighbour* and *failure probability* as typical impact factors on elastic infrastructures [139, 132]. The noisy neighbour impact factor represents runtime variations in the resource performance which are caused by additional services operating on the same underlying resources. If the resource placement is configurable, *e.g.* in a private cloud with access to the resource scheduling mechanisms, the noisy neighbour impact factor becomes an evaluation-specific impact factor. The failure probability represents the probability that compute, network or storage resources might fail.

Workload Impact Factors The workload impact factors are derived from established DBMS benchmarks (*cf.* Section 3.2). The resulting workload impact factors are depicted in Figure 4.4 where the technology-specific *workload type* represents the major impact factors in the evaluation design as it decisive for realistic evaluations. As the workload type depends on the applied benchmark technology, it is classified into *OLTP*, *HTAP* and *OLAP*. A further technology-specific impact factor represents the *workload generation model* (*cf.* Section 3.2) that defines the temporal development of the workload intensity over the evaluation runtime. The workload generation model is dependent on the applied benchmark and classified into *open*, *partially open* and *closed* workload generation [24].

Evaluation-specific impact factors represent the *data distribution* and *request characteristics* [core6]. The data characteristics are distinguished into the evaluation-specific impact factors *data set size*, *i.e.* the number of records for the evaluation, and *record size*, *i.e.* the size of a single records. Both impact factors are dependent on the selected benchmark technology. The request characteristics are distinguished into *request distribution*, *i.e.* the composition of different request types within the workload; the *intensity*, *i.e.* the number of parallel requests executed by the benchmark; and the *variance*, *i.e.* constant or fluctuating workload intensities over the evaluation runtime [core6]. These impact factors are a evaluation-specific and their configuration range is dependent on the applied benchmark. In addition, the workload *generation* represents a technology-specific impact factor that is dependent on applied benchmark technology (*cf.* Section 3.2).

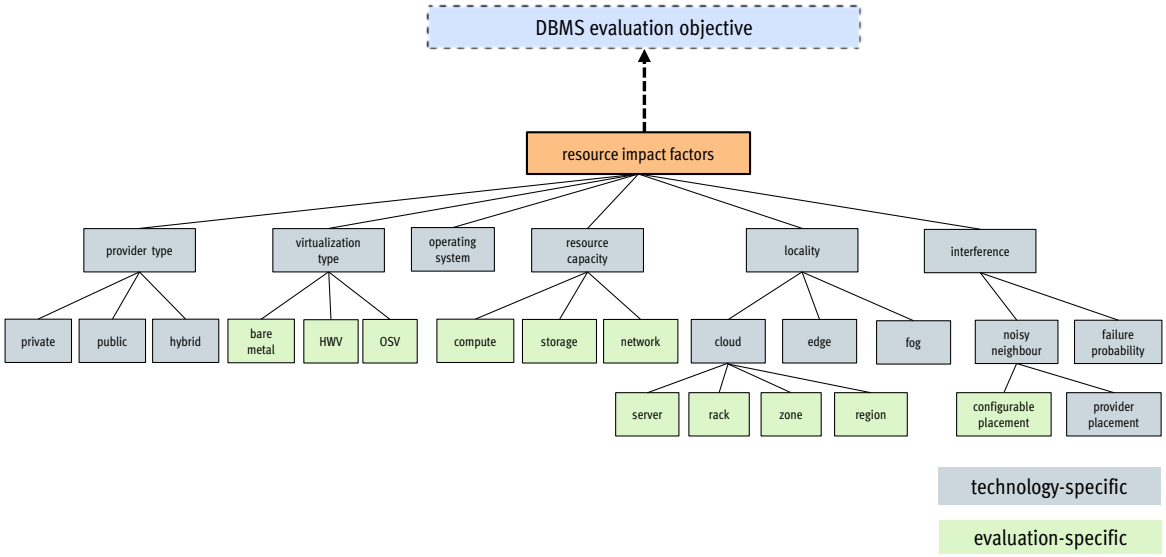


Figure 4.3: Resource impact factors

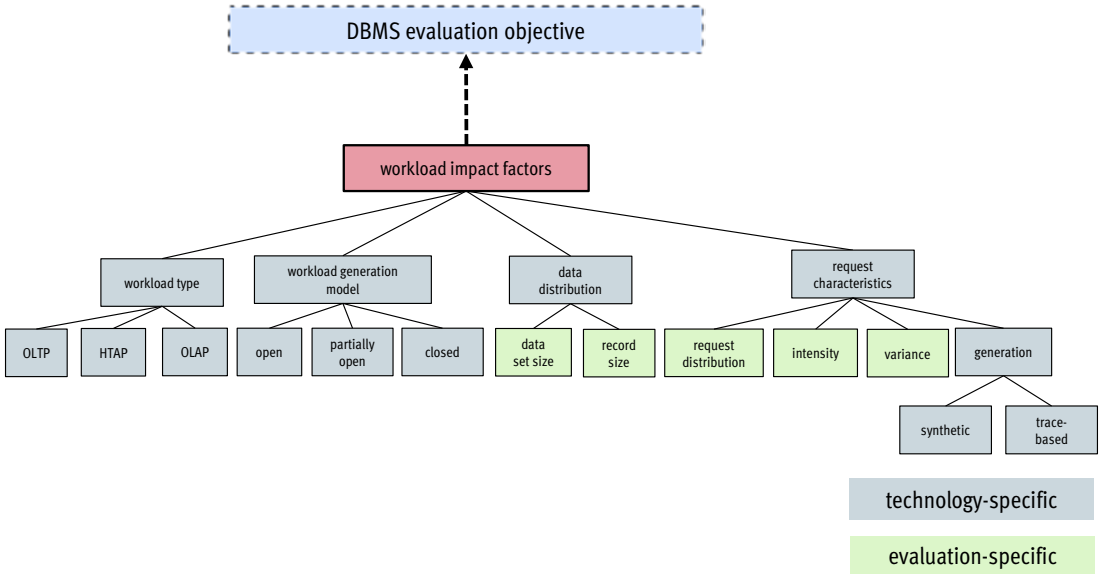


Figure 4.4: Workload impact factors

4.2 Cross-Domain Evaluation Principles

The methodological evaluation of DBMSs and elastic infrastructures is supported by numerous domain-specific evaluation guidelines that are part of DBMS benchmarks (*cf.* Section 3.2) and elastic infrastructure benchmarks (*cf.* Section 3.3). These guidelines define key Evaluation Principles (EPs) for implementing meaningful evaluation methods. While these EPs are established for their respective domain, a cross-domain view that considers DBMS and elastic infrastructure principles in a coherent context is missing. Therefore, this section addresses the following research question:

RQ-A.2 *Which evaluation principles of the DBMS elastic resource domains need to be adopted by a comprehensive DBMS evaluation methodology?*

Established DBMS evaluation principles [9, 94, 146, 180, 190] and elastic infrastructure evaluation principles [68, 100, 148, 201] are refined for evaluating distributed DBMS on elastic infrastructures [core2, core3, core9, core5]. Moreover, this cross-domain view is extended with the novel evaluation principles *automation* and *orchestration*. Both are of fundamental importance for enabling higher-level evaluation objectives [core3, core5]. Hereafter, the resulting eight cross-domain evaluation principles are presented.

EP1: Usability The usability of supportive evaluation methods is a general principle of both domains [9, 94, 68, 100] to enable the adoption of the respective method. Complex methods do not only hinder their adoption, but also limit their credibility if its control is not exposed transparently. In consequence, expressive interfaces are needed that enable human- but also machine-based interaction in a simple and intelligible manner.

EP2: Extensibility As both domains are continuously evolving and new technologies of the respective domains [9, 94, 100, 148, 180] appear frequently on the market, an extensible method ensures to keep track with these evolvements by considering new resource providers, resource offers, new DBMS technologies including their runtime configurations and future workload types. Moreover, prospective evolvements of these domains might require extensions to support additional evaluation objectives.

EP3: Significance The EP significance combines the statistical analysis of the results in a mathematical context and their expressiveness in the context of DBMS evaluations.

Significant evaluation methods apply relevant and realistic domain technologies [9, 94, 100, 148]. In consequence, they need to support realistic DBMS workload types for a wide range of DBMS technologies, evaluation-specific configuration and the usage of eligible elastic resource offers.

Furthermore, as the extent of evaluation-specific impact factors depends on the application-specific evaluation design, analytical methods are required to ensure the significance of the evaluation results. These methods need to collect not only the evaluation metrics but also evaluation-specific metadata such as the applied resources and DBMS configurations [core5], DBMS utilization metrics and system utilization metrics [94, 148, 180] to enable cross-relations between evaluation metrics and evaluation-specific metadata. Therefore, the resulting data sets need to be provided in machine-interpretable formats [190] to ease the usage of advanced post processing methods to analyse cross-relations and increase the significance of the obtained

results [201]. Apart from that, we argue that evaluation methods for higher-level evaluation objectives including runtime adaptation, *i.e.* elasticity and availability, need to provide fine-grained execution traces of the adaptation actions to ensure significant evaluation results [core4, core6].

EP4: Reproducibility The reproducibility is continuous principle for DBMS [9, 94, 146] and elastic infrastructure evaluations [68, 100, 148, 201] with the overarching goal of *reproducible evaluation results* for identical evaluation conditions. Therefore, reproducible evaluation results are achieved by the following key concepts that enable a *reproducible evaluation execution*: (i) providing the complete set of technical configurations that have been applied to obtain the results; (ii) providing supportive software-based tooling that encapsulates the technical configurations and enforces the identical evaluation execution.

The rapidly evolving resource offers and underlying technologies of the elastic infrastructure domain challenge the reproducibility of identical evaluation results as obtained evaluation results can be outdated rather quickly, even if the resource offers do not change. For instance, if provider X offers VM_SMALL with 2 cores and 4GB, and the underlying CPU of the physical server is updated from the 2015 Intel Broadwell to the 2018 Coffee Lake, this might not be visible to the customer. Moreover, distributed DBMSs can build upon non-deterministic implementations that can affect the evaluation results [160]. In consequence, *reproducible evaluation results* may not be identical for identical evaluation conditions and can diverge.

Therefore, a move from *reproducible evaluation results* towards *reproducible evaluation execution* by providing supportive software-based methods has already been emphasized in 2006 [23], arguing that the exact reproduction of evaluation results is rarely possible, and typically unnecessary. This aspect becomes even more important in the context of elastic infrastructure and their characteristics of shared resources [100]. Papadopoulos et al. [201] take up the concepts of Feitelson et al. [23] and define the provisioning of software-based methods and comprehensive data sets as a key concept for enabling reproducibility of evaluations on elastic infrastructures.

Thus, in the context of evaluating distributed DBMSs on elastic infrastructures, we define the EP reproducibility as the *reproducible evaluation execution* that requires the comprehensive experiment specification along with the methods to ensure the deterministic evaluation execution. These concepts enable the control of technology- and evaluation-specific impact factors and ensure the transparent disclosure of what is measured and how it is obtained.

EP5: Abstraction The elastic resource, DBMS and workload domain add domain-specific technologies and increase the technical complexity for designing and executing evaluations. Therefore, a reasonable level of abstraction is required to allow the evaluation execution for a multitude of domain-specific technologies within one evaluation method [core3, 190]. We argue that this abstraction is required for the evaluation design and experiment execution [core5]. Regarding the evaluation design, the concept of cloud modelling languages [add2] needs be applied to provide the DBMS operational model as abstract deployment model [core5] along with a workload model. For the experiment execution, if possible, technical details of each domain need to be abstracted by the usage of supportive COTs [core2, core9]. For instance, the abstraction of elastic resource provider interfaces can be abstracted to ease the access to multiple elastic resource providers [add7]. Regarding the workload domain, benchmarks can abstract multiple workload types by providing a uniform interface [82].

EP6: Portability The portability principle builds upon the abstraction principle and enables transferable DBMS evaluations across different domains by replacing dedicated domain-specific parameters [9, 68]. Therefore, we define that portability needs to be enabled on evaluation design and experiment execution level. The experiment specification needs to allow the exchange of domain-specific parameters, such as the resource provider, the resource offer, the DBMS technology or the workload type, and the experiment executions needs still be supported by the evaluation method [core9, core5]. Consequently, portability enables the comparative evaluation for technology-specific and evaluation-specific impact factors.

EP7: Automation Automation is a key concept for operating services on elastic infrastructures as it improves the reliability of a system by limiting human errors in repetitive and technically complex tasks [202]. In consequence, it is also a mandatory principle for evaluating evaluating cloud services [189] which is adopted for the evaluation of elastic resources by automating the elastic resource allocation tasks [106, 90].

Yet, the evaluation of distributed DBMS on elastic infrastructures comprises a multitude of additional interdependent tasks for each domain, such as allocating the deploying and configuring the DBMS cluster, deploying the benchmark and executing the workload, and processing the evaluation objective metrics. Thus, we argue that their manual execution is technically extremely complex, time consuming and error prone due to the required domain knowledge and the domain-specific technologies [core9, core5]. In consequence, a supportive automation method is required that executes these tasks transparently for the performance engineer. Hence, we argute that the implementation of the automation principle is of crucial importance to increase reproducibility (EP4) and portability (EP6) by ensuring the deterministic experiment execution for varying domain-specific parameters [core5].

EP8: Orchestration Orchestration extends the automation principle and represents a key principle to enable the evaluation of the higher-level evaluation objectives elasticity and availability, which require the time- and event-based coordination of evaluation tasks at evaluation runtime [core3, core9]. These tasks comprise actions such as scaling the DBMS cluster based on predefined triggers [core8, core6] for evaluating elasticity or injecting failures on different levels to evaluate availability [core4, core7]. Moreover, workload orchestration is emphasized by [94, 148, 190] to enable partly open and open workload models [24, 148], enabling the emulation of realistic workload patterns.

Therefore, we emphasize that orchestration is a key principle for the methodological DBMS evaluation to enable the holistic automation of higher-level evolution objectives [core5, core6, core7]. Besides, as orchestration ensures the transparent execution of each evaluation task it increases the significance (EP3) of the results and the reproducibility (EP4) of their execution by providing comprehensive execution traces [core6].

4.3 Evaluation Design

By building upon the identified impact factors and the cross-domain evaluation principles, we specify the evaluation design space for the evaluation objectives performance, scalability, elasticity and availability. In this context, the following research questions are addressed:

RQ-A.3 *What are significant metrics to evaluate the higher-level non-functional DBMS features scalability, elasticity, availability?*

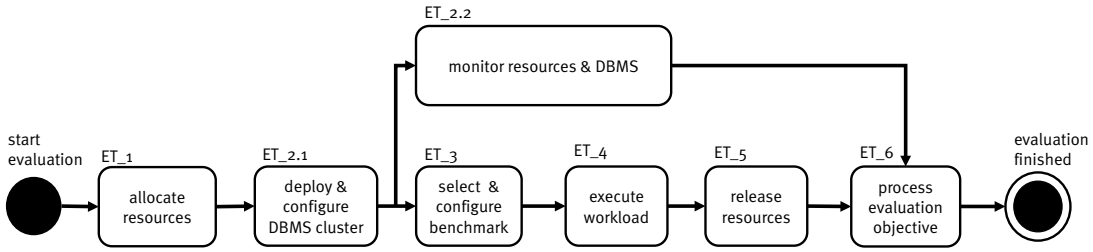


Figure 4.5: Performance evaluation process

RQ-A.4 *RQ-A.4: What are the required tasks that define the non-functional, feature-specific evaluation process and enable the measurement of higher-level metrics resulting from RQ-A.3?*

The evaluation design for each evaluation objective is defined by a set of *quantifiable* and *evaluation objective-specific metrics*. These metrics are measured by an *objective-specific evaluation process* that defines all required tasks under consideration of the domain specific impact factors defined in Section 4.1.

4.3.1 Performance Evaluation Design

The evaluation objective performance represents the primary evaluation objective for DBMS evaluations [9] and provides the foundation for the higher-level evaluation objectives scalability, elasticity and availability [core3]. In consequence, its evaluation design in terms of evaluation process and metrics has been well established over the last decades [9, 180] and we build upon these established concepts by extending them to enable reproducible and automated evaluations on elastic resources.

4.3.1.1 Metrics

The DBMS performance metrics are throughput and latency [9]. These metrics are measured from the client's perspective during the workload execution. Throughput represents the number of requests that are processed per time unit. Latency is measured per request and represents the round trip time for one request. Depending on the selected benchmark, these metrics are reported in configurable time intervals and as aggregated functions over the entire evaluation runtime.

4.3.1.2 Evaluation Process

We define the comprehensive performance evaluation process for distributed DBMS on elastic infrastructures as depicted in Figure 4.5 with the evaluation tasks Evaluation Tasks (ETs) described in Table 4.1 [core5]. Apart from the workload execution, this process explicitly considers the resource-specific and DBMS-specific tasks to provide a comprehensive evaluation design.

Table 4.1: Performance evaluation tasks

ET	Description
1	Selecting and allocating eligible resource by considering the resource impact factors. Each process execution allocates a new set of resources to ensure fair evaluations by avoiding cached data of previous iterations on the OS and DBMS level [180].
2 .1	Deploying and configuring the DBMS cluster on the allocated resources under consideration of applicable DBMS impact factors.
2.2	Deploying and configuring a monitoring service to collect resource and DBMS utilization over the evaluation runtime. This task enables the identification bottlenecks and increases the significance of the results.
3	Applying an application-specific benchmark and configuring the desired workload. As there are numerous DBMS benchmarks [core3, 154], their selection and configuration represents a central task to ensure meaningful results [94, 180].
4	Executing the specified workload via the benchmark to measure the performance metrics.
5	Releasing the allocated resources after the workload execution is finished.
6	Processing the performance objective by analysing the performance metrics in conjunction with the collected monitoring data and the applicable impact factors.

4.3.2 Scalability Evaluation Design

The scalability evaluation design builds upon the performance design and extends its scope by considering varying resource capacities and DBMS cluster sizes to evaluate the vertical and horizontal scalability capabilities (*cf.* Section 2.2.3) in the context of application-specific workloads. It is noteworthy that an evaluation-specific configuration, such as resource capacity X or DBMS cluster size Y, is applied over an entire scalability process execution and and they are not adapted at evaluation runtime as depicted in Figure 4.6.

4.3.2.1 Scalability Metrics

The scalability metric *scalability factor* builds upon the performance metrics and denotes the improvement in latency and throughput by correlating different resource capacities (*i.e.* vertical scalability) or DBMS cluster sizes (horizontal scalability) [40]. Accordingly, good horizontal scalability is indicated by a constant latency as well as throughput increasing proportionally with the cluster sizes for a specified workload if the workload is high enough to saturate the DBMS cluster [40]. Ideally, not only the cluster size, but all available resources and DBMS runtime configurations need to be considered. Yet, the scalability factor exclusively measures the performance difference between two independent cluster sizes and it does not consider the transition phase from cluster size CS_N to cluster size $CS\_N + 1$ or $CS\_N - 1$ as this is part of the elasticity design (*cf.* Section 4.3.3).

4.3.2.2 Evaluation Process

The derived scalability evaluation process is depicted in Figure 4.6. It extends the performance evaluation process by adding the iterative task ET_6 to correlate different resources and DBMS cluster sizes for evaluating the vertical and horizontal scalability capabilities [core8, core5]. The ETs are described in Table 4.2.

Table 4.2: Scalability evaluation tasks

ET	Description
1	Selecting and allocating eligible resource by considering the resource impact factors. Each process execution allocates a new set of resources to ensure fair evaluations by avoiding cached data of previous iterations on the OS and DBMS level [180].
2.1	Deploying and configuring the DBMS cluster on the allocated resources under consideration of applicable DBMS impact factors.
2.2	Deploying and configuring a monitoring service to collect resource and DBMS utilization over the evaluation runtime. This task enables the identification bottlenecks and increases the significance of the results.
3	Applying an application-specific benchmark and configuring the desired workload. In order to ensure a sufficient workload intensities to saturate even large DBMS clusters, the scalability of the workload itself is mandatory and multiple workload instances might be required.
4	Executing the specified workload via the benchmark to measure the performance metrics.
5	Releasing the allocated resources after the workload execution is finished.
6	Scale the resource capacities/DBMS cluster size and execute next scalability iteration with a new resource and DBMS cluster deployment.
7	Processing the scalability objective by correlating the performance metrics with the applied resource/DBMS cluster sizes to calculate the scalability factor. Besides, the monitoring data is analysed to identify bottlenecks and to increase significance.

4.3.3 Elasticity Evaluation Design

The elasticity evaluation design builds upon the (horizontal) scalability design as horizontal scalability and consequently a distributed DBMS architecture are the mandatory requirements to provide elasticity. In contrast to scalability, DBMS elasticity represents the ability to cope with fluctuating workloads *at runtime* by adapting the DBMS cluster size accordingly [40, 46]. Consequently, the adaptation of the DBMS cluster at evaluation runtime is a mandatory task in the elasticity evaluation design. The adaptation of the workload represents an optional task that is required to emulate fluctuating workloads. The DBMS and workload adaptations can be defined by time or monitoring data based triggers [core6].

Moreover, considering the DBMS utilization is an important aspect in the elasticity evaluation design to design significant evaluations, including realistic adaptation triggers [core8, core6]. If the DBMS utilization

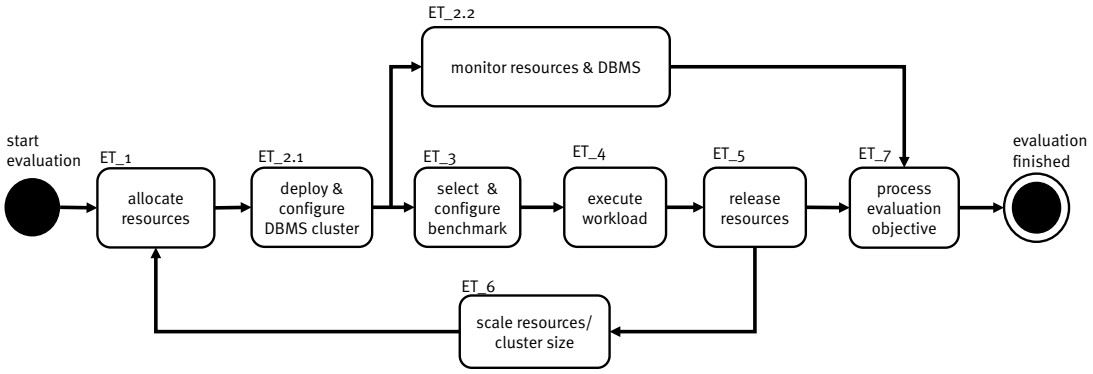


Figure 4.6: Scalability evaluation process

states are not known beforehand, the performance of the initial DBMS cluster, its monitoring data and the applied workload intensity need to be correlated to determine DBMS-specific utilization states in a *calibration phase*. This calibration phase comprises multiple iterations of the performance evaluation process with varying workload intensities [core8, core6].

As a result, each elasticity evaluation comprises at least three phases: (i) an *initial phase* with a predefined DBMS operational model and a predefined workload; (ii) an *elastic scale* phase, executing a DBMS adaptation; (iii) a *stabilization phase* after the adaptation is finished [core6]. Additional phases are *workload adaptation phases* or additional *elastic scale* phases.

Elasticity Metrics While elasticity metrics are a widely discussed aspect in cloud computing research [174], we focus specifically on elasticity metrics of DBMS. DBMS elasticity metrics are proposed by [53, 75] that calculate a dimensionless metric by considering only a subset of the elasticity evaluation design space [core6]. We build upon the descriptive *elastic speedup* metric defined by Cooper et al. [40] and extend it by defining the quantifiable elasticity metrics *provisioning time*, *data distribution impact* and *data distribution time* that need to be measured during the elastic scale-out/-in task [core6].

Evaluation Process We define the elasticity evaluation process as depicted in Figure 4.7. It comprises the required tasks that need to be executed to measure established elasticity metrics [53, 75, core6]. Executing solely the required tasks enables the elasticity evaluation based on a constant workload intensity. Including the optional tasks enables the elasticity evaluation under fluctuating workloads [core6]. The dedicated elasticity evaluation tasks are described in Table 4.3.

4.3.4 Availability Evaluation Design

The availability of distributed storage systems can be affected by a multitude of software- or hardware-related events [98, 41]. In this thesis, we evaluates the availability of distributed DBMSs under consideration of elastic resource failures [core4, core7]. Therefore, the availability evaluation design builds upon chaos engineering concepts [128] and applies the failure injection as a key concept for evaluating DBMS availability [core4, core7]. Due to different virtualization levels of elastic infrastructures, a fine-grained failure model is required to cover

Table 4.3: Elasticity evaluation tasks

ET	Opt.	Description
1		Selecting and allocating eligible resource by considering the resource impact factors. Each process execution allocates a new set of resources to ensure fair evaluations by avoiding cached data of previous iterations on the OS and DBMS level [180].
2.1		Deploying and configuring the DBMS cluster on the allocated resources under consideration of applicable DBMS impact factors.
2.2		Deploying and configuring a monitoring service to collect resource and DBMS utilization over the evaluation runtime. This task enables the identification bottlenecks and increases the significance of the results.
2.3	X	If fluctuating workloads are specified, an additional monitoring service is deployed to monitor the workload execution over the evaluation runtime
3		Applying an application-specific benchmark and configuring the desired workload. In order to enable the (optional) workload adaptation in the subsequent task ET_5 without additional orchestration services, the selected benchmark needs to support partially open or open workload model [24, 148].
4		Executing the specified workload via the benchmark to measure the performance metrics.
5		Adapting the DBMS cluster by processing the specified adaptation triggers. The trigger processing uses predefined thresholds and the combination of time, system and DBMS monitoring data.
6	X	Adapting the workload intensity by processing the workload adaptation triggers, using time or workload monitoring data.
7		Releasing the allocated resources after the workload execution and all adaptation tasks are finished.
8		Determining the dedicated elasticity phases and processing the elasticity objective by taking into account the performance metrics for each identified phase, the applied adaptations, and the collected monitoring data.

physical and virtual resource failures [core4]. In consequence, a key task of the availability evaluation is the injection of resources and DBMS failures on different levels. Apart from that, we also consider failure injection on the DBMS level as additional failure type [core7].

To provide a comprehensive availability evaluation design, failure-dependent recovery tasks represent an additional key design concept [core4, core7]. Including the recovery task ensures realistic evaluations by incorporating the performance impact of the recovery task on the DBMS cluster [core7]. As a result, the availability evaluation comprises at least four phases: (i) a *healthy phase* with a predefined DBMS cluster; (ii) a *unhealthy phase* in which a failure has been injected; (iii) a *recovery phase* in which the recovery task is executed (iv) a *recovered phase* after the recovery task is finished [core7]. In addition, an availability evaluation

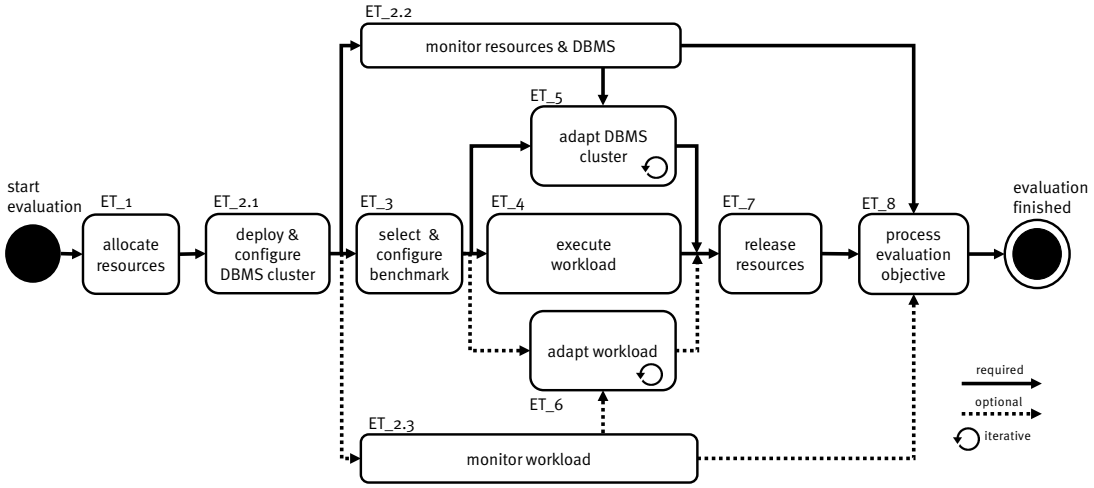


Figure 4.7: Elasticity evaluation process

can comprise multiple failure injections and consequently multiple unhealthy and recovery phases [core7].

Metrics Similar to elasticity, distributed system and cloud research proposes different availability metrics [174]. But as these do not explicitly focus on the distributed DBMS, we build upon the performance metrics in relation to the different availability evaluation phases, and define a novel set of DBMS availability metrics [core4, core7]. These metrics enable the quantification of the DBMS availability in case of DBMS and elastic resource failures. The resulting metrics are the *performance impact*, *accessibility* and *request error rate* that can be applied for each availability evaluation phase [core4, core7].

Evaluation Process In order to measure the defined availability metrics, we define comprehensive availability evaluation process [core4, core7] that is depicted in Figure 4.8. The key concept of the availability evaluation process is the failure injection during the workload execution to emulate the unhealthy phase. In addition, the optional execution of failure- and DBMS-dependent recovery tasks enables the emulation of the recovery phase to reach the recovered phase. The tasks are described in the following Table 4.4:

Table 4.4: Availability evaluation tasks

ET	Opt.	Description
1		Selecting and allocating eligible resource by considering the resource impact factors. Each process execution allocates a new set of resources to ensure fair evaluations by avoiding cached data of previous iterations on the OS and DBMS level [180].
2		Deploying and configuring the DBMS cluster on the allocated resources under consideration of applicable DBMS impact factors.
2.1		Deploying and configuring a monitoring service to collect resource and DBMS utilization over the evaluation runtime. This task enables the identification bottlenecks and increases the significance of the results.
3		Selecting an application-specific benchmark and configuring the desired workload. The benchmark needs to provide fine-grained performance metrics over the evaluation runtime to enable the processing of the phase-specific availability metrics
4		Executing the specified workload via the benchmark to measure the performance metrics.
5		Injecting a predefined failure on DBMS or elastic resource level to trigger the transition of the DBMS cluster into the unhealthy phase.
6	X	Executing a predefined recovery action that is dependent on the injected failure and the applied DBMS
7		Rereleasing the allocated resources after the workload execution and the recovery execution are finished
8		Determining the dedicated availability phases and processing the availability objective by taking into account the performance metrics for each identified phase, injected failures, executed recovery actions, and the collected monitoring data.

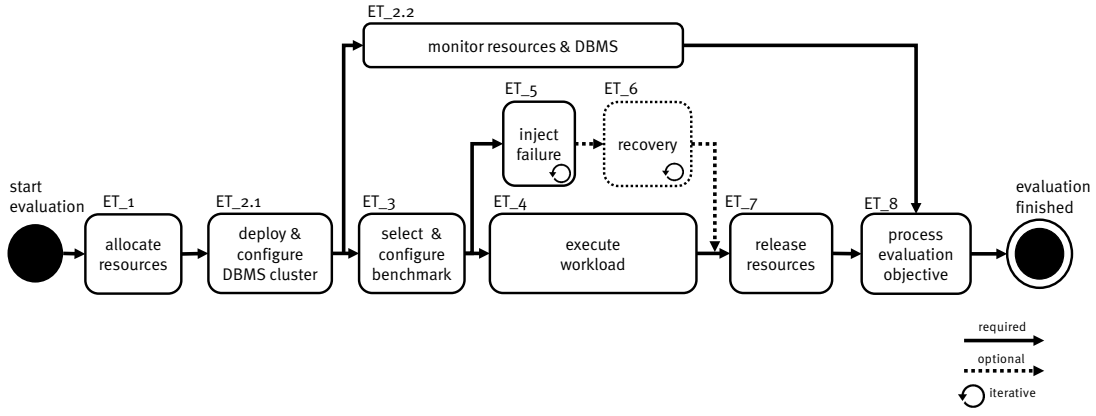


Figure 4.8: Availability evaluation process

4.4 Summary

This chapter presented the findings that contributed in addressing RO-A by defining a comprehensive evaluation methodology for distributed DBMSs on elastic infrastructures. This methodology was built upon three key concepts. First, the identification of the domain-specific impact factors that needed to be considered for designing DBMS evaluations. Furthermore, the resulting impact factors were classified into technology-specific and evaluation-specific impact factors. Secondly, the DBMS and elastic infrastructure evaluation principles were reviewed and extended to establish cross-domain evaluation principles for the evaluation of distributed DBMS on elastic infrastructures. Thirdly, the evaluation design space for performance, scalability, elasticity and availability was defined. Each evaluation objective design comprised the relevant metrics and a comprehensive evaluation process with dedicated evaluation tasks.

The resulting evaluation methodology provided comprehensive concepts for designing and executing DBMS evaluation on elastic infrastructures. Yet, it was still demanding for extensive domain-knowledge and technical expertise in the DBMS, elastic infrastructure and workload domain. Furthermore, the sheer number of DBMS operational models that are considerable evaluation targets makes the manual execution of the respective evaluation processes basically impossible. In consequence, novel technological methods are required that are able to consider the identified impact factors, follow the cross-domain evaluation principles and implement the design of the respective evaluation objective. In the Chapter 5, we present the novel DBMS evaluation framework Mowgli that adopts the presented methodology and provides the technical methods to automate the full evaluation process for each evaluation objective.

Chapter 5

Methods for the Automated Evaluation of Non-functional DBMS Features

This chapter builds upon the established evaluation methodology of Chapter 4 and summarizes selected findings of the publications [core2] included in Chapter 9, [core5] included in Chapter 12, [core6] included in Chapter 13, [core7] included in Chapter 14, and [core10] included in Chapter 17 that present the resulting technical methods for addressing the second research objective:

RO-B: *Providing the technical evaluation methods that enable the automated execution of the comprehensive evaluation methodology established in Research Objective A.*

This chapter is structured into four sections, which address the accompanying research questions RQ-B.1 - RQ-B.4. Section 5.1 presents the findings on the specification of comprehensive DBMS evaluations. Building upon these specifications, Section 5.2 presents *Mowgli*, the novel DBMS evaluation automation framework for the evaluation objectives performance and scalability. Section 5.3 summarizes the evaluation frameworks *Kaa* and *King Louie* that enable the automated evaluation of the higher-level evaluation objectives elasticity and availability. Section 5.4 presents the scope of the evaluation data sets that can be generated with these novel evaluation methods before Section 5.5 summarizes this chapter.

5.1 DBMS Evaluation Templates

The comprehensive evaluation of distributed DBMS on elastic infrastructures requires the consideration of the overarching DBMS, elastic resource and workload impact domains that comprise a multitude of domain-specific impact factors (*cf.* Section 4.1). In consequence, the following research question is addressed to enable the automated evaluation execution:

RQ-B.1: *How should comprehensive DBMS evaluations be specified to enable the automated execution while ensuring reproducibility and portability?*

In order to address this research question, we follow the concepts of DSLs such as CAMEL [add2] and encapsulate all domain-specific properties in a unified *Evaluation Scenario Template (EST)* [core5]. An EST provides a predefined structure with relevant domain-specific properties to define comprehensive evaluations. An

EST represents an abstract specification, which requires concrete domain-specific parameters to be provided. Therefore, each EST defines the evaluation-specific impact factors as configurable parameters in a declarative manner. This enables the abstraction principle (EP5) (*cf.* Section 4.2) by separating the technical execution from the declarative description of the evaluation.

As an EST includes all evaluation-specific parameters, it enables the reproducibility (EP4) (*cf.* Section 4.2) of conducted evaluations for adopters [core5]. Besides, reference ESTs can be provided in community-driven or enterprise-specific EST repositories.

Each EST is divided into multiple sub-templates, depicted as simplified JavaScript Object Notation (JSON) examples in Listing 5.1. The `deploymentTemplate` and `workloadTemplate` are mandatory sub-templates across all evaluation objectives and enable the specification of performance and scalability ESTs [core5]. The `adaptationTemplate` is required for the evaluation objectives elasticity and availability (*cf.* Section 4.3) and is further classified into `elasticitySpec` templates [core6] and `failureSpec` templates [core7]. Each sub-template represents an independent and reusable template entity to ensure the portability principle (EP 6) across different evaluation objectives. In consequence, an EST is designed by composing a set of sub-templates and applying domain-specific parameters.

The target evaluation objective determines the required sub-templates. An exemplary evaluation API that exposes objective-specific interfaces is shown in Listing 5.2. Depending on the evaluation objective, the specified EST in combination with additional metadata, such as an identifier and the number of iterations, is required to carry out an evaluation.

In the following, each sub-template is briefly introduced with respect to its domain-specific properties.

```
"EST": {
 "deploymentTemplate": {
    // required for the objectives: performance, scalability, elasticity, availability
    ...
  },
 "workloadTemplate": {
    // required for the objectives: performance, scalability, elasticity, availability
    ...
  },
 "adaptationTemplate": {
   // required for the objectives: elasticity, availability
    ...
  }
}
```

Listing 5.1: Evaluation scenario template

```
"evaluation_API": {
  "objective/performance":{
    "post": {
      "id" :"PERFORMANCE_EVALUATION_ID",
      "iterations": 10,
      "EST" :{PERFORMANCE_EST}
    }
  },
```

```
  "objective/scalability":{
    "post": {
      "id" :"SCALABILITY_EVALUATION_ID",
      "iterations": 10,
      "EST" :{SCALABILITY_EST}
    }
  },
  "objective/elasticity":{
    "post": {
      "id" :"ELASTICITY_EVALUATION_ID",
      "iterations": 10,
      "EST" :{ELASTICITY_EST}
    }
  },
  "objective/availability":{
    "post": {
      "id" :"AVAILABILITY_EVALUATION_ID",
      "iterations": 10,
      "EST" :{AVAILABILITY_EST}
    }
  }
}
```

Listing 5.2: Exemplary evaluation API

5.1.1 Deployment Template

The deployment template exposes evaluation-specific parameters to deploy a DBMS cluster on elastic resources, *i.e.* the desired resource configurations, the DBMS and its runtime parameters. An exemplary deployment template for the deployment of distributed DBMS on cloud resources is depicted in Listing 5.3. The deployment template is separated into the target DBMS technology and version and a set of DBMS components. The DBMS components are classified into the data processing components `SEED` and `DATA`, and the optional `MANAGEMENT` component. This concept enables the specification of common distributed DBMS topologies as shown in [core5]. Each component is associated with a `resource` entity that contains the evaluation-specific resource parameters (*cf.* Section 4.1). For VM-based cloud deployments, the `resource` entity follows the concept of a cloud provider agnostic model [add7] by defining the abstract properties `idCloud`, `idImage`, `idHardware` and `idLocation`. The provided parameters need to match provider-specific identifiers such as the EC2 identifiers for specific VM types or they need to be interpretable by supportive evaluation frameworks. Additional resource entities are defined for container-based deployments [core10]. Moreover, a set of evaluation-specific DBMS parameters has to be provided, defining common distributed DBMS properties `replicationFactor` and `nodeConfiguration`. In addition, a set of DBMS-specific parameters can be applied via the `customConfiguration` property.

```
"dbmsCluster": {
   "type": "CASSANDRA",
   "version" :"3.11.2",
```

```
  "databaseComponent": [
    {
      "type": "SEED",
      "instances": 3,
      "resource": {
        "idCloud": CLOUD_ID,
        "idImage": IMAGE_ID,
        "idHardware": HARDWARE_ID,
        "idLocation": LOCATION_ID
      },
      "replicationFactor": {
        "envName": "REPLICATIONFACTOR",
        "envValue": 3
      },
      "nodeConfiguration": {
        "dataMemory": {
          "envName": "DATAMEMORY",
          "envValue": 1500
        },
        "indexMemory": {
          "envName": "INDEXMEMORY",
          "envValue": 500
        }
      },
      "customConfiguration": [
          {
           "envName": "SHARDING_STRATEGY",
           "envValue": "org.apache.cassandra.dht.Murmur3Partitioner"
          }
      ],
    }
  ]
}
```

Listing 5.3: Cloud deployment template

5.1.2 Workload Template

The workload template exposes relevant properties to specify an application-specific workload. The extent of the exposed properties are benchmark-specific as our approach does not provide an abstraction on different benchmarks. In consequence, different workload types such as YCSB or TPC-C workloads result in different workload templates [core5]. In consequence, the properties of a workload template need to be defined manually for different benchmarks. The following Listing 5.4 shows a simplified workload template for a YCSB workload. The `dbmsEndpoints` entity represents a common entity across every workload template and is required to establish the connection between the deployed DBMS cluster and the workload execution during the evaluation execution. The additional entities `measurementConfig`, `workloadConfig` and

databaseConfig are YCSB-specific and need to be provided to ensure a comprehensive and reproducible workload description [core5]. In consequence, workloadTemplates enable reusable workload specifications. Additional workloadTemplate examples of TPC-C and a time-series workloads are available online[1].

```
"YCSBWorkload": {
   "dbmsEndpoints": [
      {
        "ip" :"192.168.2.1",
        "port" :9042
      }
   ],
   "measurementConfig": {
     "interval": 10
   },
   "workloadConfig": {
     "workloadClass": "com.yahoo.ycsb.workloads.CoreWorkload",
     "maxExecutionTime": 1800,
     "threadCount": 16,
     "recordCount": 4000000,
     "operations": 10000000,
     "fieldCount": 10,
     "fieldLength": 500,
     "requestdistribution": "UNIFORM",
     "readProportion": 0.5,
     "updateProportion": 0.5,
     "insertProportion": 0
   },
   "databaseConfig": {
     "databaseBinding": "CASSANDRA2",
     "endpointParameterName": "hosts",
     "tableParameterName": "cassandra.keyspace",
     "tableName": "ycsb",
     "configPorperties": [
       {
         "name": "cassandra.writeconsistencylevel",
         "value": "ONE"
       }
     ]
   }
 }
```

Listing 5.4: Workload template YCSB

[1] https://omi-gitlab.e-technik.uni-ulm.de/mowgli/getting-started/tree/master/examples

5.1.3 Adaptation Templates

The adaptation templates comprise the required properties to design ESTs for the elasticity and availability objective. Therefore, they are classified into the elasticity-specific `elasticitySpec` and the availability-specific `failureSpec` adaptation templates which are described below.

Elasticity Templates As defined in the elasticity evaluation process in Section 4.3, it requires the specification of DBMS adaptations at evaluation runtime with optional workload adaptation. Therefore, the `elasticitySpec` needs to expose the respective properties to specify comprehensive and reproducible DBMS and workload adaptations. The resulting elasticity template [core6] follows the concept of sophisticated adaptation models for cloud applications such as the Scalability Rule Language (SRL) [102, 113]. Yet, in contrast to these advanced approaches that address the adaptation over the full application lifecycle, we focus on on the relevant parameters to enable dedicated elasticity evaluations [core6]. An exemplary `elasticitySpec` template is depicted in Listing 5.5 that can include n elastic adaptation entities. Each elastic adaptation can target the DBMS or the workload, depending on the `adaptationTarget`. The elasticity adaptation defines a set of `scalingType` and `scalingTrigger` entities [core6]. While the example in Listing 5.5 contains a time-based `scalingTrigger`, the `elasticitySpec` also supports composed `scalingTrigger` based on system and DBMS metrics [core6].

```
"elasticitySpec": [
   {
    "adaptationTarget" :"DBMS",
    "nodeType" :"DATA",
    "scalingType": "OUT",
    "scalingTrigger": {
       "scalingType" :"TIME",
       "scalingDate" :"600s"
    },
    {
     // a second elasticity adaptation
    }
]
```

Listing 5.5: Elasticity Template

Availability Templates The key concept of the availability evaluation process defined in Section 4.3 is the failure injection in conjunction with optional recovery actions. In consequence, the availability template needs to enable the speciation of these concepts. The resulting template defines the `failureSpec` depicted in Listing 5.6, which enables to specify n failure injections associated with optional recovery actions [core7]. It allows the timing of the failure injection in terms of the evaluation runtime via `failureTiming`. The speciation of different failure levels is enabled via the `failureType`, such as VM, availability zone or region, and their `severity`. Besides, optional recovery actions are specified via `recoveryType` with an associated *recoveryTiming* entity. The resulting availability ESTs enable the specification of comprehensive and reproducible DBMS availability evaluations in the context of elastic resource failures [core7].

```
"failureSpec":[
    {
     "failureTiming": "600s"
     "failureType": "VM",
     "severity": "permanent"
     "failureRecovery": true,
     "recoveryTiming": "900s",
     "recoveryType": "ADD-NODE"
    },
    {
     // a second failure to be injected
    }
]
```

Listing 5.6: Resource failure template

5.1.4 Implementation

The ESTs are implemented as JSON documents as JSON is widely adopted in cloud DSLs [84]. We apply the Swagger framework[2] for implementing the ESTs Swagger builds upon the OpenAPI specification [213] and supports code generation for multiple programming languages. Yet, the concept of ESTs is not limited to any technology and other common DSL technologies such as the Eclipse Modelling Framework (EMF)[3] could also be used.

Exemplary ESTs that have been defined within this thesis are available in a public repository[1] and in the resulting data sets [data2, data1, data4].

5.2 Mowgli Framework

By building upon the previously introduced concept of ESTs that ensure the comprehensive speciation of the relevant domain-specific parameters, the following research question needs to be addressed to enable the automated evaluation execution:

RQ-B.2: *What technical concepts are required to enable the evaluation automation across the elastic resource, DBMS and workload domain, ensuring reproducibility and portability?*

In order to address this research question, we provide the novel evaluation framework Mowgli. Mowgli fully automates the evaluation execution of distributed DBMS on elastic infrastructures based on the provided ESTs while ensuring reproducibility and portability [core5]. The Mowgli framework supports the evaluation objectives performance and scalability by implementing the reference evaluation processes defined in Section 4.3. Hereafter, we summarize the key automation concepts of Mowgli and provide an overview of its technical architecture.

[2] https://swagger.io/
[3] https://www.eclipse.org/modeling/emf/

5.2.1 Automation Concepts

In order to automate the full evaluation process, the Mowgli framework applies the following key automation concepts:

Technology Catalogues Mowgli defines the concept of domain-specific technology catalogues that enable the mapping of the provided EST parameters to concrete technical implementations. Therefore, Mowgli contains three domain-specific catalogues: the *resource catalogue* enables the mapping to provider-specific resource offers, the *DBMS catalogue* contains the DBMS-specific deployment and configuration scripts and the *workload catalogue* contains the technical benchmark implementations to execute the specified workload [core5]. Each catalogue represents a software component with a dedicated API. This ensures the independent extensibility of each catalogue with additional technologies.

Process Automation ESTs only provide the required parameters for the evaluation execution and do not expose any configurations that affect the evaluation process flow. This ensure the deterministic evaluation execution by preventing the modification of the objective-specific evaluation process. In return, the Mowgli framework implements internally the process for each objective-specific evaluation and exposes only relevant configurations such as the number of evaluation iterations. Following this concept, Mowgli enables reproducible evaluations by guarantees the deterministic evaluation process for each iteration. Therefore, each evaluation task is implemented in reusable software artefacts within the Mowgli framework. Their execution is automated by applying a lightweight workflow engine to orchestrate the execution of the dedicated evaluation tasks.

Deployment Automation Logical tasks with an evaluation process can be composed by a varying number of technical tasks. Specifically, the recurring allocation of resources (ET1) and the deployment of the DBMS cluster (ET2) comprise numerous internal provider- and DBMS-specific tasks. To abstract these technology-specific steps and ease their automation, Mowgli applies a COT [core2] to automate these steps. Therefore, a COT is selected that enables the multi-cloud resource management, provides advanced application lifecycle handling [112] and supports the automated discovery of cloud resources [add7] to enable dynamic resource catalogues.

Objective Processing In order to enable significant evaluations, Mowgli provides an extensive set of additional metadata besides the raw benchmark metrics. The additional metadata comprises system- and DBMS-specific monitoring data which is automatically prepared in visual and machine-readable formats. Moreover, the applied resources, DBMS and workload parameters, are provided in machine-readable formats to enable the postprocessing by external data analytics tools. Mowgli's internal data analytics capabilities enable the automated processing of the performance and scalability metrics [core5], also shown in the accompanying data sets [data2]

5.2.2 Framework Architecture

The resulting framework architecture of Mowgli is depicted in Figure 5.1 and comprises six loosely coupled software components that interact via REST-based APIs.

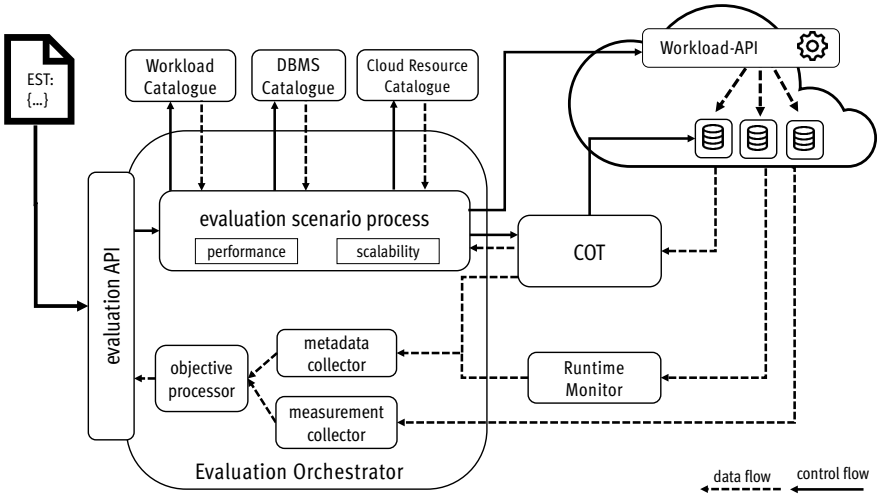


Figure 5.1: Mowgli architecture

The *Evaluation Orchestrator* represents the central component within the framework that provides the *Evaluation API* for submitting the specified performance and scalability ESTs. Therefore, it provides a programmatic Representational State Transfer (REST) interface and basic web interface. The submitted EST is processed to an executable *Evaluation Scenario Process (ESP)* by mapping the specified EST parameters to the technical implementations via the *Workload Catalogue*, *DBMS Catalogue* and the *Cloud Resource Catalogue*. The *Workload Catalogue* and *DBMS Catalogue* are independent components that provide the required workload and DBMS lifecycle actions in script-based formats. The *Cloud Resource Catalogue* is provided by the multi-cloud COT [add9, add7] which enables the automated resource discovery for multiple cloud providers. The *Evaluation Orchestrator* executes the ESP by applying the internal workflow engine to automate the execution of the process-specific tasks. Thus, the *Evaluation Orchestrator* invokes the COT to allocate the required resources as the COT abstracts the provider-specific APIs into a generic resource API [add7]. In addition, the COT is also invoked to deploy the specified DBMS by executing the DBMS-specific lifecycle actions. The *Workload-API* represents an independent component that is invoked by the *Evaluation Orchestrator* to execute the specified workload. It provides a basic abstraction layer over the supported benchmarks by providing a unified API that extends the benchmark-specific parameters with the Mowgli-specific parameters such as the dynamic DBMS endpoints or network types to use. The *Workload-API* bundles the required benchmark implementations. Mowgli supports the automated deployment of the *Workload-API* via the COT or the usage of self-deployed *Workload-API* instances. In consequence, according to the specified number of workload instances in the EST, Mowgli orchestrates the required number of *Workload-API* instances to create the specified workload. The *Runtime Monitor* stores the monitoring data, execution traces and additional metadata for each evaluation in as time-series data. The *Evaluation Orchestrator* follows established cloud monitoring and orchestration concepts [add10] and enables the dynamic monitoring orchestration during each evaluation execution. The resulting benchmark metrics, system and DBMS monitoring data and additional resource metadata are joined by the *Measurement Collector* and the *Meta-data Collector*. Subsequently, the *Objective Processor* computes the evaluation objective according to the EST, *i.e.* performance and scalability. The objective specific results

are provided in machine interpretable formats for advanced post processing and in visual formats for the explorative analysis [data2].

5.2.3 Implementation

The Mowgli framework components are implemented as Java-based REST services by building upon the Swagger toolkit[2]. In addition, python-based software components are applied for the *objective processor*.

Within Mowgli, the COT is implemented by Cloudiator [add9, add7] that enables multi-cloud resource management, provides advanced application lifecycle handling [112] and supports the automated discovery of cloud resources [add7]. Mowgli is not limited to Cloudiator and additional COTs [157, 207] can be integrated to increase the orchestration capabilities or extend the scope of resource offers. The *Runtime Monitor* is implemented by building upon the time-series DBMS InfluxDB[4].

The Mowgli framework is available open source under the Apache 2 licence[5]. A first version of Mowgli has been released in 2019 [data3] and in its current version, Mowgli supports eight NoSQL and NewSQL DBMS[6], three workload types based on the YCSB [40], a DBMS-specific TPC-C[7] and a time-series benchmark[8]. Each component of the Mowgli framework is provided as a Docker container and the framework is deployable via an accompanying Docker Compose file[9].

5.3 Mowgli for Higher-Level Evaluation Objectives

While Mowgli provides a novel method for the automated performance and scalability evaluation of distributed DBMS on elastic infrastructures, the following research question needs to be addressed for higher-level evaluation objectives:

RQ-B.3: *Which adaptation concepts are required by supportive evaluation methods to enable the evaluation of the higher-level non-functional features elasticity and availability?*

We address this research question by building upon the concepts of the Mowgli framework and extend them with novel evaluation methods for higher-level evaluation objectives, namely *Kaa* [core6] for the elasticity objective and *King Louie* [core7] for the availability objective. The resulting evaluation methods are presented below, including their integration into the Mowgli framework as depicted in Figure 5.2.

5.3.1 Elasticity: Kaa Framework

The Kaa framework [core6] builds Mowgli and enables the automation of the elasticity evaluation process defined in Section 4.3 under the consideration of DBMS and optional workload adaptations at evaluation runtime. The Kaa framework is integrated into Mowgli and its advances are highlighted in blue in Figure 5.2.

[4] https://www.influxdata.com/
[5] https://omi-gitlab.e-technik.uni-ulm.de/mowgli
[6] https://omi-gitlab.e-technik.uni-ulm.de/mowgli/getting-started/tree/master/features
[7] https://github.com/cockroachdb/loadgen
[8] https://github.com/timescale/tsbs
[9] https://omi-gitlab.e-technik.uni-ulm.de/mowgli/docker

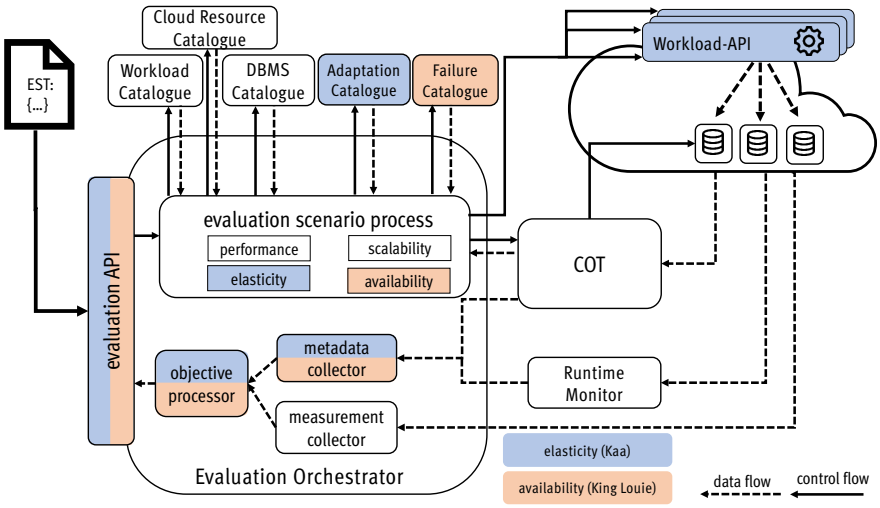


Figure 5.2: Kaa and King Louie framework architecture

The *Evaluation-API* is extended to enable the submission of elasticity ESTs. The *Evaluation Orchestrator* is extended with the corresponding *elasticity ESP*. The *Adaptation Catalogue* implements the required catalogue that enables the mapping of the DBMS and workload adaptations in the EST to the respective technical invocations for the execution of the elasticity ESP. The *Workload-API* is extended to report the workload progress which is required to enable progress-based adaptations [core6]. Within the elasticity process, the data of the runtime monitor and the workload progress reported by the Workload-API are consciously evaluated against the defined DBMS and workload adaptation triggers in the EST. The *Meta-data Collector* automatically computes the duration of each DBMS adaptation and processes the phase-specific times such as resource provisioning time or data distribution time. They are required to process the elasticity metrics [core6] via the *Objective Processor*. These advances enable the following key automation concepts for the elasticity evaluation of distributed DBMS on elastic resources:

DBMS Adaptation Kaa supports the automation of time-based and metric-based DBMS adaptations via the *Evaluation Orchestrator* which processes the specified `elasticitySpec` of the EST. For time-based adaptations, predefined timestamps of the EST are used as a trigger. Time-based adaptations enable isolated elasticity evaluations for dedicated workload configurations, while emphasizing reproducibility. The automation of the metric-based adaptations is executed by continuously evaluating the data of the *Runtime Monitor* against the metric-based thresholds of the EST. Metric-based adaptations enable more realistic evaluation scenarios, but their specification requires a thorough understanding of the DBMS utilization in order to apply reasonable adaptation thresholds [core6]. The execution of the DBMS adaptations, *i.e.* scale-out or scale-in, are carried out via the adaptation interface of the applied COT. For a scale-out, the COT allocates new resources and executes the respective DBMS lifecycle actions that are predefined in the DBMS catalogue. For a scale-in, the COT executes the required lifecycle actions to remove the DBMS node from the cluster and releases the associated resources.

Workload Adaptation The automation of workload adaptation follows the concept of the DBMS adaptations where the *Evaluation Orchestrator* processes the optional workload adaptations in the `elasticitySpec`. Kaa supports time-based, progress-based and metric-based workload adaptations [core6]. The *Evaluation Orchestrator* invokes the adaptations via the *Workload-API*. A distinction is made between benchmarks that implement an open or partially open workload model and those benchmarks which implement a closed workload model [24, 148]. While the former benchmarks allow the adjustment of the workload at runtime directly via the *Workload-API*, the latter prevent the direct adjustment of the workload at runtime. The orchestration capabilities of Kaa address this limitation, as additional *Workload-API* instances can be used to schedule fluctuating workloads. For instance, this enables the emulation of a partially open workload model for the YCSB [40] which implements a closed workload model.

Implementation Kaa builds upon the technological stack of Mowgli (*cf.* Section 5.2.3). The adaptation triggers are implemented by mapping the specified `scalingTrigger` of the elasticity templates to InfluxDB-specific queries. For this, Kaa uses the InfluxQL query language of InfluxDB[10]. The DBMS adaptations are implemented by building upon the scaling interfaces of Cloudiator [add10, add6]. The Kaa framework is integrated into the Mowgli framework and publicly available[5]

5.3.2 Availability: King Louie Framework

The DBMS availability evaluation is enabled by the King Louie framework which builds upon Mowgli and Kaa. It automates the availability evaluation process defined in Section 4.3 and enables the availability evaluation under consideration of elastic resource failures on different levels [core4]. Therefore, King Louie extends Mowgli as depicted in Figure 5.2 with the advances highlighted in red. The *Evaluation-API* is extended to support availability ESTs and accordingly the *Evaluation Orchestrator* is extended with *Availability ESP*, implementing the reference availability evaluation process. The *Failure Catalogue* contains the technical invocations that need to be executed for the specified failure injections and (optional) recovery actions of the provided EST. The extension of the *Meta-data Collector* enables the automated determination of the DBMS cluster states within an availability evaluation execution: *healthy*, *unhealthy*, *recovering* and *recovered* [core7]. These phases are provided to the extended *Objective Processor* to process the availability metrics. Building upon these extensions, King Louie automates the following technical concepts that are essential to evaluate DBMS availability:

Failure Injection The failure injection capabilities of King Louie enable the emulation of realistic failures of elastic resources across different levels [core4]. Therefore, King Louie exploits the resource management capabilities of the applied COT or it can apply dedicated failure injection frameworks [133]. The *Evaluation Orchestrator* processes the `failureSpec` of the EST and maps the specified failures to the required resource management invocations via the COT interface. This enables the injection of different failure types such as the (temporary) stopping or deletion of a single resource instance, but also the injection of failures on availability zone or region level which will affect multiple resource instances. In addition, custom failure types can be added as scripts to the *Failure Catalogue* and their execution is enabled by the *Evaluation Orchestrator*.

[10] `https://docs.influxdata.com/influxdb/v1.8/query_language/`

Failure Recovery The execution of failure recovery actions represents an optional step in the implementation of the availability evaluation process that enables the availability metric processing of the *recovering* and *recovered* phase. The failure recovery steps depend on the specified failures and execute the required invocations to restore the preceding healthy cluster state. In consequence, for temporary failures, these invocations target existing resources and DBMS instances while for permanent failures, the steps comprise the allocation of new resources and the deployment of new DBMS nodes like in the elastic scale-out process during the elasticity evaluation. Depending on the DBMS, the execution of additional recovery lifecycle actions might be required which need to be specified in the *DBMS Catalogue*.

Implementation King Louie builds upon the technological stack of Mowgli (*cf.* Section 5.2.3) and Kaa (*cf.* Section 5.3.1). The applied failure injection framework is an own Java-based implementation with additional technical details presented in [core4, core7]. It exploits the resource management capabilities of Cloudiator to inject failures on the resource level and applies direct SSH connections to inject failures on the OS and DBMS levels. The recovery action implementation builds upon the elastic scaling capabilities provided by Kaa with the extension for executing additional DBMS-specific lifecycle actions. The King Louie framework is integrated into the Mowgli framework and publicly available[5].

5.4 Evaluation Data Collection

Mowgli [core5], Kaa [core6] and King Louie [core7] provide the automation methods for the evaluation objectives performance, scalability, elasticity and availability. In addition to the automation of the respective evaluation processes, the following research question is addressed to ensure significant results and facilitate the knowledge discovery:

RQ-B.4: *How can the automation methods ensure significant results and reduce the efforts in the knowledge discovery of the non-functional DBMS features?*

Mowgli, Kaa and King Louie address this research question by providing the results as comprehensive evaluation data sets which contain four types of data: (i) benchmark-specific metrics, (ii) system and DBMS monitoring data, (iii) execution logs and (iv) evaluation metadata. These data sets enable exploratory and confirmatory data analysis approaches [148]. In the common process of making use of evaluation results as defined by Bermbach et al. [148], Mowgli, Kaa and King Louie automate the evaluation *execution*, *data collection and storage* as well as *pre-processing* of the evaluation results. The advanced result analysis based on sophisticated statistical [167, 187, 196] and machine learning [153, 208, 210] methods is out of scope of this thesis. Yet, the generated data sets of Mowgli provide valuable input for further offline processing steps based such methods, Moreover, Mowgli's modular architecture is prepared to integrate such methods to extend Mowgli's online analysis capabilities.

All data sets are provided in a file-based manner to facilitate their usage for data analytics methods. To follow the reproducibility principle [201], a performance and scalability data set [data2] created by Mowgli [core5], an elasticity data set [data1] created by Kaa [core6] and an availability data set [data4] created by King Louie [core7] are published as open access data sets as examples. The characteristics of the four data types of each evaluation data set are summarized in the following:

Benchmark-specific Metrics Each data set contains the benchmark-specific metrics, such as throughput and latency measured by the YCSB or transaction rate measured by the TPC-C benchmark. The format of the metrics is dependent on the benchmark and can be an aggregated value or a time series over the evaluation runtime. In addition, the metrics are aggregated based on configurable statistical methods. The resulting data is provided in machine-readable formats as input for advanced data analytics methods and in visual formats to enable quick high-level insights for the performance engineer.

Monitoring Data The monitoring data is collected from the *Runtime Monitor* and pre-processed by the *Meta-data Collector* (*cf.* Figure 5.2) as time series over the evaluation runtime. The data is provided in machine-readable and visual formats. The monitoring data comprises resource utilization data such as CPU usage, memory usage, I/O usage and network traffic of the resources hosting the DBMS cluster and Workload-API instances. In addition, DBMS-specific monitoring data is provided. The monitoring data increases the significance of the results as it enables the identification of resource bottlenecks (on the virtualization level) for the DBMS cluster and eases the verification that the Workload-API instances are not a bottleneck.

Execution Logs The execution logs are generated by the *Meta-data Collector* and stored in machine-readable formats that map the respective evaluation process steps (*cf.* Section 4.3) to timestamps of the respective evaluation runtime. In consequence, the execution logs increase the significance by ensuring full transparency of the automation process. Moreover, the execution logs enable the processing of additional metrics such as the provider-specific resource provisioning time or the DBMS deployment time. For the higher-level evaluation objectives elasticity and availability, the execution logs contain the timestamps for all executed adaptation steps, as these are required for processing the higher-level evaluation objective metrics (*cf.* Section 4.3).

Evaluation Metadata In addition to the resulting evaluation data, each data set contains metadata to ensure the reproducibility of the evaluations. Therefore, the applied EST is included and additional resource-specific metadata. Yet, the underlying physical infrastructure or even the capacity of the resource types can change over time [201]. Thus, the extent of the resource-specific metadata is dependent on the resource provider. For instance, for public cloud providers such as AWS, Mowgli only provides metadata for the virtual resources and it is unable to provide any metadata for the physical resources. For private clouds Mowgli offers the capabilities to extend the metadata with physical resource details if the cloud infrastructure provides appropriate interfaces.

Data Set Based on these data sets, Mowgli enables the processing of the evaluation objectives performance, scalability, elasticity and availability and ensures the reproducibility of the evaluation as all relevant technical details are included. The significance of the results is supported by underpinning the provided benchmark metrics with additional monitoring data and resource-specific metadata. The knowledge discovery is supported by pre-processing the benchmark-specific metrics and the monitoring data, and providing them in visual formats for initial analysis while all evaluation data is provided in machine-readable formats for advanced data analytics methods.

5.5 Summary

This chapter presented the findings that contributed to achieve RO-B by building upon the established methodological DBMS evaluation concepts of Chapter 4. The findings provide novel methods for DBMS evaluations on elastic infrastructures: first, a comprehensive specification for DBMS evaluations was defined as DBMS Evaluation Scenario Templates (ESTs). ESTs expose the evaluation-specific impact factors as configurable parameters. Each template comprises objective-specific sub-templates for the DBMS deployment, workload execution and runtime adaptations. The specified ESTs serve as input to the novel DBMS evaluation framework Mowgli that fully automates the performance and scalability evaluation processes. Therefore, it builds upon a modular architecture and applies the following key automation concepts: technology catalogues as domain-specific abstraction, evaluation process automation, deployment automation and objective processing. Mowgli has been enriched with the Kaa framework that enables the DBMS elasticity objective by the key concepts of DBMS and workload adaptations at evaluation runtime. In order to enable the availability evaluation of DBMS with the focus on elastic resource failures, Mowgli has been further enriched with the King Louie framework that applies the key concepts of automated failure injection and failure recovery. For each evaluation objective, Mowgli provides a comprehensive data set that contains the objective-specific metrics and monitoring data, execution logs and metadata in pre-processed machine readable and visual formats.

These novel DBMS evaluation methods have been applied in several realistic case studies and in the following Chapter 6, the results are discussed and validated against the cross-domain evaluation principles.

Chapter 6

Validation

This chapter summarizes the validation of Mowgli including Kaa and King Louie with respect their capabilities for evaluating distributed DBMSs on elastic infrastructures. The feature set of Mowgli, Kaa and King Louie is validated against their applicability for realistic use cases, the support for evaluating higher-level evaluation objectives and the compliance with the established evaluation principles (*cf.* Section 4.2). Therefore, we apply a case study driven and a feature driven validation. First, in Section 6.1 the applicability for evaluating distributed DBMSs on elastic infrastructures is validated by four industry-driven case studies. These address the DBMS performance impact of elastic resources [core10], a comparative performance and scalability study across different DBMS technologies, cloud providers and cloud resources [core5], a DBMS elasticity study under consideration of different workload intensities [core6] and a DBMS availability evaluation in case of resource failures [core7]. Secondly, Mowgli's feature set is evaluated against the established cross-domain evaluation principles [core5, core6, core7] in Section 6.2. Section 6.3 discusses the lessons learned on the evaluation of distributed DBMSs on elastic infrastructures and Section 6.4 summarizes this chapter.

6.1 Case Studies

Mowgli's applicability for driving the decision process to determine the DBMS operation model is validated by multiple industry-driven case studies. These case studies are based on real problem statements regarding the DBMS operational model on elastic infrastructures that have been identified within the Cloud2Go research project in collaboration with Daimler TSS and within the European research project Melodic[1]. An overview on the extent of each Case Study (CS) is provided in Table 6.1. Each case study targets different evaluation objectives and in consequence, verifies Mowgli's applicability for the considered evaluation objectives performance, scalability, elasticity and availability. In addition, Table 6.1 provides in row miscellaneous a summary of additional smaller scale case studies that have been carried out in the respective projects. These case studies are not discussed in detail as their results are reflected by the large scale case studies CS1 — CS4. In the following, CS1 — CS4 are summarized based on the identified *evaluation scenarios*, the target *evaluation objective*, and the resulting *lessons learned* of the respective case study.

6.1.1 CS1 - Performance Impact of Elastic Resources

The case study is centred around the general research challenge of identifying the optimal elastic resource type for the target application by considering the trade-off between virtualization overhead and operational

[1] `https://melodic.cloud/`

Table 6.1: Case study metrics

ID	Evaluation Objective	Scenarios	Repetitions	Data Sets
CS1	DBMS performance impact of elastic resources [data6]	10	10	100
CS2	DBMS performance & scalability on elastic resources [data2]	102	5	610
CS3	DBMS elasticity on elastic infrastructure [data1]	64	5	320
CS4	DBMS availability under elastic resource failures [data4]	16	10	160
Misc.	project-specific DBMS performance, scalability and availability evaluations [data5]	118	1-5	236
Total	Performance impact of elastic resources, DBMS performance, DBMS scalability, DBMS elasticity, DBMS availability	310	N/A	1426

benefits [139]. This general challenge is substantiated for DBMSs by analysing the performance impact of different virtualization technologies and storage configurations [core10]. This case study has been carried out using an early implementation of Mowgli [core10] and its methodology and technical components have been integrated into Mowgli in the meantime.

Evaluation Scenarios Within this case study, the elastic resource types container, VM and the combination VM with container represent the evaluation-specific resource impact factors. Besides, for container-based resources, we distinguished between the host file system and the container-internal file system as storage location. All resources are specified with identical resource capacities and they are deployed in a private cloud based on OpenStack. The applied DBMS technology is MongoDB[2] in a single node deployment and the applied benchmark is the YCSB for creating a read-heavy and write-heavy workload. As a result, the case study comprises ten evaluation scenarios and each scenario is specified with ten executions [data6], which are automated by the framework[3].

Evaluation Objective The evaluation objective is performance under consideration of the applied resource types. Therefore, the performance metrics throughput and latency are related with the applied resource type and workload. The results provide a comparative analysis of the performance overhead in relation to the resource type.

Lessons Learned The results show that the usage of VM-based resources comes with a significant performance overhead compared to container resources. For container-based resources, the usage of the container-internal filesystem causes significant performance overhead for write-heavy workloads. This confirms the findings of existing studies which analyse the performance overhead of different virtualization technologies

[2] https://www.mongodb.com/de
[3] https://github.com/omi-uulm/Containerized-DBMS-Evaluation

for stateless applications [118, 139]. A further insight shows that the usage of container resources on top of VMs causes a tolerable performance overhead of 6% compared to the operational advantages enabled by containers.

6.1.2 CS2 - Performance and Scalability

The second case study [core5] validates Mowgli by addressing the overarching research challenge of selecting the optimal elastic resource provider, resource type and DBMS runtime configurations [89, 158] in the context of cloud-hosted DBMSs [core5]. Therefore, the case study evaluates the performance and scalability in the context of different cloud providers, storage resources and DBMS runtime configurations.

Evaluation Scenarios The considered resource providers within this case study are a private OpenStack cloud and the public AWS EC2 cloud. The applied resource types are VMs of identical resource capacities. The evaluation-specific resource impact factor is the storage type, *i.e.* SSD and remote storage backends. The deployed DBMS technologies are Apache Cassandra[4] and Couchbase[5]. The evaluation-specific DBMS impact factors are the cluster size that ranges from 3 - 5 - 7 - 9 nodes and the DBMS-specific client side write consistency which ranges from low to high. The applied benchmark is the YCSB for creating a write-heavy workload. As a result, this case study comprises 102 evaluation scenarios and each evaluation scenario is executed five times by Mowgli [data2].

Evaluation Objective There are three evaluation objectives of this case study. First, the DBMS-specific performance impact of varying write consistencies; secondly, the performance impact of identical resource capacities offered by different cloud providers and the impact of different storage backends; thirdly, a comparative performance and scalability analysis across the applied DBMSs.

Lessons Learned The performance results show that the impact of the applied write consistency strongly depends on the DBMS technology. While for Apache Cassandra stronger consistency settings cause a tolerable performance decrease, for Couchbase the performance decreases by 90% when comparing the lowest and highest write consistency settings. The direct performance comparison between Apache Cassandra and Couchbase shows that Couchbase achieves a higher performance for the weak write consistency while Apache Cassandra achieves better performance for strong write consistency settings. With respect to scalability objective, the results show that both DBMSs provide scalability with increasing cluster sizes but limiting factors can be the underlying resources such as the remote storage backend or the saturation of the workload. In consequence, large cluster sizes may cause negative scalability due to increasing synchronization operations. The dependable identification of such limiting factors requires comprehensive evaluation scenarios considering the relevant evaluation-specific impact factors and their repeated execution which is enabled by Mowgli [core5].

[4] `http://cassandra.apache.org/`
[5] `https://www.couchbase.com/`

6.1.3 CS3 - Elasticity

The third case study addresses the general research challenge of horizontally scaling resources and application at runtime [173, 166] in the dedicated context of distributed DBMSs on cloud resources [core6]. Therefore, the validation of Kaa [core6] by building upon Mowgli focuses on the objective elasticity under the consideration of different workload intensities and workload types.

Evaluation Scenarios The evaluation scenario builds upon VM-based resources provided by a private OpenStack cloud infrastructure with identical resource capacities. The evaluated DBMS technologies are Apache Cassandra and Couchbase, deployed as three node clusters. The YCSB is selected as the benchmark for creating write-heavy and read-heavy workloads with the evaluation-specific workload intensities low, optimal and overload. In order to determine the DBMS-specific workload intensities, an additional calibration phase is executed. The evaluation-specific adaptation action is an elastic scale-out, adding one additional DBMS node including the required resources based on a time-based trigger. The resulting case study comprises 32 evaluation scenarios for the calibration phase and 32 elasticity evaluation scenarios where each scenario is executed five times [data1].

Evaluation Objective The evaluation objectives target the DBMS-specific elasticity under consideration of varying workload intensities and the comparative elasticity analysis across the two DBMSs. Therefore, particular focus relies on the elasticity metrics provisioning time, data distribution impact and data distribution time.

Lessons Learned The results show that Apache Cassandra and Couchbase provide the expected elasticity capabilities for all workload intensities of the read-heavy workloads, *i.e.* increasing the performance after the elastic scale-out is finished. Moreover, Apache Cassandra does not show a significant data distribution impact while Couchbase shows a significant data distribution impact. Regarding the write-heavy workload, both DBMSs show unexpected elasticity behaviours as for all workload intensities the data distribution impact is severe, which results in a constant performance decrease during the elastic scale-out. These results show that DBMS adaptations need to consider a multitude of impact factors to specify efficient DBMS adaptation rules [core6].

6.1.4 CS4 - Availability

The fourth case study addresses the general impact of elastic resource failures [132, 133] in the dedicated scope of cloud-hosted DBMSs. Therefore, King Louie [core7] by building upon Mowgli is validated by a case study with the focus on the DBMS availability in case of cloud resource failures for different cluster sizes.

Evaluation Scenarios The evaluation scenario consists of VM-based resources with static resource capacities deployed at a private OpenStack-based cloud. The applied DBMSs are Apache Cassandra and Couchbase with the evaluation-specific impact factor cluster sizes of three and seven nodes. The YCSB is applied to create a write-heavy and read-heavy workload of low and high intensity. The failure specification is defined as a single failure injection on VM level in combination with a subsequent recovery action to restore the initial

cluster size of the respective evaluation scenario. In consequence, the case study comprises 16 evaluation scenarios with ten executions per evaluation scenario [data4].

Evaluation Objective The evaluation objective is availability under consideration of resource failures with dedicated focus on the performance impact during the healthy, unhealthy, recovering and recovered phase. Therefore, availability is analysed in a DBMS-specific and in a comparative context.

Lessons Learned The results show novel and unexpected insights into the availability capabilities of Apache Cassandra and Couchbase. The ten executions of each evaluation scenario result in different but reoccurring performance patterns for the respective scenarios. None of the DBMSs is able to overcome a VM resource failure without a downtime at client-side. The duration of the downtime is highly dependent on the DBMS, the cluster size and the applied workload. The downtimes for Apache Cassandra range from 10 to 240 seconds. Couchbase is unable to overcome resource failures in the 3-node cluster and for the 7-node cluster the downtimes range from 30 to 175 seconds. These insights build the foundation for further case studies to consolidate the availability capabilities of distributed DBMSs and to derive efficient failure mitigation strategies.

6.1.5 Case Study Discussion

Mowgli's applicability for evaluating the non-functional features of distributed DBMSs on elastic infrastructures is verified by the performed case studies, which comprise evaluation objectives that reach beyond basic performance evaluations and target higher-level evaluation objectives as summarized in Table 6.1. The 310 evaluation scenarios addresses the evaluation challenges of current and upcoming DBMS operational models [127, 211] and Mowgli enables their specification and automated execution. The specified evaluation scenarios comprise a multitude of *technology-specific* and *evaluation-specific* impact factors.

Across all case studies, the applied technology-specific impact factors comprise three DBMSs that implement different architectures and consistency models, two cloud providers with different virtualization and resource types.

The evaluation-specific impact factors comprise six DBMS cluster sizes, ten DBMS client-consistency configurations, three replication factors, three compute capacities, three storage capacities, three workload distributions and eight workload intensities. In addition to these case studies, further project-specific case studies [data5] have been specified and performed in cooperation with Daimler TSS and within the Melodic project. These studies comprise an additional private cloud provider, six additional DBMSs and additional workloads based on the TPC-C benchmark. It is noteworthy that this list of technology- and evaluation-specific impact factors reflects only a small excerpt of the supported impact factors of Mowgli.

Performing the required evaluation executions to create the 1426 evaluation data sets further verify Mowgli's automation capabilities and the resulting data sets enable the in-depth analysis of the objective-specific results. Moreover, Mowgli facilitates the quantification of the impact of elastic resource [core10, core5] and enables novel insights into the higher-level non-functional features of distributed DBMS [core6, core5].

6.2 Support for Evaluation Principles

This section summarizes the support for the established cross-domain evaluation principles (*cf.* Section 4.2) of Mowgli [core5], Kaa [core6] and King Louie [core7]. Therefore, we present their compliance for each principle and discuss limitations. In the following discussion, we refer to the Mowgli as the integrated framework comprising Mowgli, Kaa and King Louie.

EP1: Usability Mowgli eases its usage via a comprehensive REST interface, which is based on the OpenAPI specification. OpenAPI facilitates the specification of meaningful REST APIs by providing tool support for designing, building, documenting and consuming the respective APIs. Thus, Mowgli provides a detailed documentation of its REST interface and its usage, including the basic web interface. Yet, a certain level of DBMS evaluation knowledge is essential to specify the required ESTs and Mowgli does not provide a sophisticated web interface for editing ESTs or visualizing the evaluation results. Moreover, advanced approaches such as chatbots [212] for guiding unexperienced users through the evaluation specification and execution are not part of Mowgli.

EP2: Extensibility The extensibility of Mowgli has been confirmed by the frameworks Kaa [core6] and King Louie [core7] which extend the framework and its feature set. Moreover, Mowgli's micro-service architecture and the concept of the technology catalogues offer convenient extension points for integrating new technologies or extending existing framework features. However, the efforts in extending Mowgli are dependent on the affected components. For instance, adding a new DBMS will only require the implementation of five lifecycle scripts, an update of the EST specification and the execution of the respective code generation tools. On the contrary, the extension or replacement of the COT Cloudiator to enable more extensive container orchestration capabilities [207], will require more effort as multiple software components of Mowgli will be affected.

EP3: Significance Mowgli's significance regarding the supported evaluation scenarios and evaluation objectives is verified by its acceptance at established cloud and performance engineering conferences [core5, core6, core7]. Besides, Mowgli's significance is also confirmed in an industrial context through the cooperation with Daimler TSS where it is applied for addressing industry-driven challenges in selecting and operating distributed DBMSs. Moreover, Mowgli has been included in the internal services of Daimler TSS[6] and within the European research project Melodic[7] [204]. While Mowgli is able to cover realistic operational models of distributed DBMS, the significance of the applied workloads are dependent on the selected benchmarks. Currently, Mowgli supports synthetic benchmarks and the integration of trace-based benchmarks such as BenchFoundry [146] would improve the significance of the supported evaluation scenarios.

The significance of the obtained results are verified by comparing them against established DBMS evaluation results if applicable. Moreover, the significance of the results is underpinned by comprehensive evaluation data sets [data2, data1, data4] that provide additional evaluation-specific metadata.

[6] https://www.uni-ulm.de/in/fakultaet/in-detailseiten/news-detail/article/neues-analyse-tool-fuer-datenbankmanagementsysteme-mowgli-weist-den-weg-im-datenbanken-dschungel/

[7] https://melodic.cloud/

EP4: Reproducibility While recent findings show that the reproducibility principle finds little adoption in evaluations of the DBMS domain [190] and elastic infrastructure domain [201], Mowgli supports this principle as follows. The validation of reproducibility principle builds upon the received ACM Reusable Badge[8], which was awarded to the Mowgli framework [core5] within the artefact evaluation track of the International Conference on Performance Engineering[9]. As Kaa and King Louie build upon Mowgli and apply the same reproducibility methods, the following aspects also apply to them.

The reproducibility is verified for both aspects of the reproducibility principle, *i.e.* a complete technical description of the evaluation and the deterministic evaluation execution. First, Mowgli provides the full technical description of the applied evaluation setup, comprising the applied DBMS, resource and workload parameters as ESTs that enable the reproduction of the performed evaluation scenarios. Secondly, the Mowgli framework provides the technical methods to ensure the deterministic execution of the ESTs. Moreover, the results of the artefact evaluation track also confirm that comparable evaluation results have been obtained by the artefact evaluators. Yet, it is noteworthy that Mowgli only ensures the reproducible specification of the evaluations scenarios and their deterministic execution. The obtained results represent a snapshot that is dependent on a number of volatile and time dependent impact factors such as performance variations on the resource level [44, 135], compaction and replication processes on the DBMS level [150, 179] or statistical randomness of synthetic benchmarks on the workload level [146]. While Mowgli can not control these impact factors, it provides the required methods to reproduce the results in multiple versions to quantify the resulting variance of such volatile impact factors.

EP5: Abstraction Mowgli's abstraction capabilities are validated on evaluation design and evaluation execution level. Regarding the evaluation design, the ESTs have confirmed their abstraction capabilities by enabling the specification of realistic case studies while providing an appropriate level of abstraction for the DBMS, resource and workload domain. Moreover, the ESTs have proven their abstraction capabilities by supporting the reproducibility of the evaluations.

The abstraction capabilities for the evaluation execution are verified based on applied domain-specific technologies within the case studies. These comprise multiple cloud providers, resource types, DBMS technologies and benchmarks. While Mowgli enables a high-level abstraction for the resource provisioning and DBMS deployment by applying the provider-agnostic COT Cloudiator [add9, add7] with the DBMS and resource catalogues [core5]. It only provides a basic abstraction for the workload domain. In particular, Mowgli relies on the abstraction capabilities of the integrated benchmarks such as the YCSB and abstracts the communication parameters across all benchmarks.

EP6: Portability The performed case studies confirm that Mowgli enables the evaluation portability across multiple domains, *i.e.* CS2 [core5] performs a comparative evaluation across multiple DBMS and resource domain parameters, CS3 [core6] and CS4 [core7] apply a comparative study by applying the objective-specific adaptation actions across different DBMSs. The portability of each evaluation scenario is achieved by adapting the respective domain-specific parameters or replacing the objective-specific sub-templates. For instance, changing the target DBMS only requires the adjustment of a few EST parameters. Yet, the portability is dependant on the abstraction capabilities provided by Mowgli's components. For instance, if the applied benchmark

[8] https://www.acm.org/publications/policies/artifact-review-badging
[9] https://icpe2019.spec.org/tracks-and-submissions/artifact-evaluation-track.html

does not abstract multiple DBMSs, the portability of the respective EST is limited by the supported DBMSs of the benchmark.

EP7: Automation Mowgli's automation capabilities are verified throughout the case studies where Mowgli is able to automate all tasks of the 310 evaluation scenarios. In particular, Mowgli automates the allocation of elastic resources, the deployment of the DBMS cluster, deployment of the monitoring services, workload execution and processing of the objective-specific metrics. The automation capabilities for the resource allocation and the DBMS deployment are dependent on Cloudiator [add9, add7]. As the landscape of COTs has evolved during this thesis [157, 207], novel COTs such as Kubernetes [129] or DC/OS[10] provide additional automation capabilities for container-based resources that go beyond the means of Cloudiator. Mowgli already provides the means to incorporate these COTs on a conceptual level and only technical effort is required to implement these extensions. Besides, fog and edge automation frameworks have started to evolve during this thesis [156]. In the context of these advances, Mowgli is able to provide comprehensive automation capabilities for cloud resources but requires extensions regarding container-based resources as well as for the fog and edge resource domain. Yet, Mowgli's extensibility and the multi-cloud concepts can be applied to ease the incorporation of fog and edge concepts. Besides, the automated evaluation objective processing offers possibilities for enhancements such as advanced data processing frameworks [143] or visualization methods [148].

EP8: Orchestration In extension to Mowgli's automation capabilities, its orchestration capabilities are verified by the performed case studies CS3 and CS4 that require the orchestration of evaluation tasks at runtime such as adapting the DBMS cluster (CS3/CS4) or injecting resource failures (CS4). Moreover, these orchestration events are integrated into the metadata collection to enable the automated processing of higher-level evaluation metrics. While the applied runtime adaptations prove their applicability to enable the evaluation of the higher-level objectives elasticity and availability, these runtime adaptation offer extension points for advanced orchestration concepts. For instance, the elasticity objective can be enhanced by extending its adaptation capabilities with sophisticated DBMS auto-scaling approaches of the DBMS [81, 91] and cloud context [104, 188]. Regarding failure injection, continuative methods such as jepsen[11] or the ChaosToolkit[12] can advance Mowgli orchestration capabilities in the scope of availability objective.

6.3 Reflections on Evaluating distributed DBMS on Elastic Infrastructures

Throughout the development, implementation, and evaluation of the concepts and the resulting software framework presented within this thesis, novel insight and experiences in the process of evaluating the non-functional features of distributed DBMSs operated on elastic infrastructures have been established. The insights are described in the respective case studies CS1 — CS4 and they are based on addressing the challenges of elastic resource, distributed DBMS and DBMS workload domain, resulting in a overarching methodology and the integrated evaluation frameworks Mowgli, Kaa and King Louie. These frameworks provide extensive automation and orchestration capabilities that enable novel insights in the non-functional features of

[10] https://dcos.io/
[11] https://github.com/jepsen-io/jepsen
[12] https://chaostoolkit.org/

distributed DBMS by considering a multitude of resource, DBMS and workload parameters, and processing the objective-specific metrics. The experiences we gained throughout these performed case studies, verify that Mowgli is a valuable and powerful tool to support the performance engineer in *designing* and *executing* comprehensive evaluations in realistic environments. Yet, the experiences also show some boundaries of Mowgli, Kaa and King Louie with respective to the *evaluation objective support*, *elastic resource support* and *evaluation data processing* methods. These boundaries are discussed in the following paragraphs.

Evaluation Objective Support While Mowgli has proven its evaluation capabilities for the evaluation objectives performance, scalability, elasticity and availability as integrated framework, it does not address additional higher-level evaluation objectives such as DBMS consistency [78, 93] or the operational costs for DBMS on elastic infrastructures [35, 66], which are also of importance to determine the DBMS operational model. As Mowgli provides a modular and extensible architecture, its extension towards additional evaluation objectives it methodologically and technically feasible.

Elastic Resource Support With respect to the supported technologies, Mowgli has proven to support heterogenous DBMS, different elastic providers and multiple elastic resource types. Yet, in its current state, Mowgli does not support the latest advances in elastic resource infrastructures such as Kubernetes [129] nor does it support the heterogenous infrastructures of the still evolving edge and fog providers [194]. As these resource offers are not widely adopted for operating distributed DBMSs yet, Mowgli's resource orchestration capabilities need to be extended to support elastic resources to the full extent for future operational models of distributed DBMSs. These extensions can be built upon the existing multi-cloud concepts of Mowgli.

Data Processing: Variance Analysis Regarding the *result variance analysis*, the evaluation methodology needs to adequately deal with the volatility within the elastic resource domain (resource interferences, time-dependent performance variations), DBMS domain (DBMS-specific compaction and synchronization processes) and workload domain (statistical randomness in synthetic workloads). Consequently, evaluations need to be repeated multiple times and statistical methods need to be applied to increase the significance of the results. Mowgli's automation capabilities allow any number of evaluation repetitions. Based on the resulting data sets, it provides the basic statistical functions such as *mean*, *standard deviation* or *percentiles* for each evaluation-specific metric. By providing these basic statistical functions, Mowgli already supersedes common approaches in cloud service benchmarking [201]. However, Mowgli does currently not provide advanced statistical methods such as confidence intervals or Analysis of Variance (ANOVA) [8]. Extending Mowgli's data processing capabilities with such statistical methods is already technically possible. These methods can be applied to support the performance engineer in determining the required number of repetitions, identifying a robust statistical function for aggregating multiple data sets and increasing the statistical significance of the evaluation results.

Data Processing: Data Correlation Regarding the *evaluation data correlation*, the extensive number of domain-specific parameters within an EST makes the direct correlation between dedicated domain-specific input parameters and the resulting objective-specific metrics a challenging task. Mowgli allows the specification of a multitude of domain-specific parameters in the evaluation design and execution process and it provides the resulting data sets in human and machine readable formats to support the performance engineer.

Yet, the identification of relevant EST parameters and their correlation with the evaluation results are subject to the performance engineer's domain knowledge. Thus, the performance engineer would benefit from the automated processing of advanced data correlation functions such as the Pearson correlation coefficient or machine learning approaches such as Rafiki [153]. In order to apply such approaches Mowgli's modular architecture provides the suitable interfaces to extend it with such approaches and automate their execution within the evaluation process.

While Mowgli partially addresses advanced data analytic concepts such as *variance analysis* and *evaluation data correlation*, Mowgli's holistic evaluation automation concepts enable such concepts in the first place for distributed DBMS on elastic infrastructures as these concepts require an extensive number of evaluation data. Mowgli's reproducibility and automation capabilities enable the generation of such extensive evaluation data for distributed DBMSs on elastic infrastructures. In order to support these concepts to in an integrated way in Mowgli, Mowgli's data processing capabilities can to be extended and it already provides the required interfaces to integrate the outlined data analytic methods.

6.4 Summary

This chapter summarized the validation results of Mowgli, Kaa and King Louie. The validation follows a twofold approach: first, Mowgli applicability for evaluating distributed DBMS on elastic infrastructure was validated based on four industry-driven case studies that target the evaluation objectives performance, scalability, elasticity and availability. The case studies comprised 310 different evaluation scenarios, resulting in 1426 data sets that confirmed Mowgli's significance and its automation capabilities. Moreover, Mowgli provided novel insights into the performance impact of elastic infrastructure, the performance impact of DBMS-specific consistency configurations, DBMS elasticity under different workload patterns and the DBMS availability in case of elastic resource failures. Secondly, a qualitative validation was carried out by comparing Mowgli's capabilities against the cross-domain principles established in Chapter 4.2. The results confirmed that Mowgli follows the cross-domain evaluation principles with a dedicated focus on reproducibility, portability and automation. Yet, the validation also showed that extensions are required to provide these principles for the recent elastic resource domains of the edge and fog domain. Finally, the overall experiences in evaluating distributed DBMSs on elastic infrastructures with Mowgli were reflected and limiting aspects regarding the statistical data processing capabilities of Mowgli were discussed.

Chapter 7

Conclusion and Future Work

Started with web applications and intensified by Big Data as well as Internet of Things (IoT) applications, the need to manage massive and continuously growing amounts of data has become a key challenge for data management systems. In consequence, the Database Management System (DBMS) landscape has significantly evolved over the last decade and the established relational DBMSs are extended with distributed NoSQL and NewSQL DBMSs. These DBMS technologies promise to address the data management needs of modern data-intensive applications by ensuring high performance and providing the higher-level non-functional features (horizontal) scalability, elasticity and high-availability.

In parallel to the evolvement of DBMS technologies, their operational model also advances towards the usage of elastic resources such as the cloud or even edge and fog resources. These elastic resources enable virtually unlimited scalability on the resource level in an elastic manner. As NoSQL and NewSQL DBMS emphasize a distributed and shared-nothing architecture, they make the elastic resources the preferable operational model to enable scalability and elasticity also on the resource level. In consequence, a DBMS operational model for data-intensive applications builds upon elastic resources, a distributed DBMS and its application-specific runtime configuration.

While these advances of the DBMS and elastic resource domains offer promising capabilities to address the data management needs of data-intensive applications, the heterogeneity and extent in DBMS and elastic resource offers lead to an inherent complex decision process in identifying an operational model that ensures high performance, horizontal scalability, elasticity and high availability. Especially, as both domains are interdependent, they need to be considered in its entirety to avoid suboptimal decisions.

In consequence, this decision process needs to be supported by the evaluation of eligible DBMS operational models based on realistic workloads. There are numerous, established benchmarks to emulate realistic workloads for the DBMS and for the elastic resource domains. But these approaches come with the following limitations that hinder the comprehensive evaluation of distributed DBMSs on elastic infrastructures: (i) existing evaluation approaches only focus on their respective domain and do not consider the entire DBMS operational model; (ii) DBMS evaluation methodologies are missing that go beyond performance and address higher-level non-functional DBMS features such as scalability, elasticity and availability; (iii) holistic evaluation automation methods are required to cope with the multitude of considerable DBMS operational models.

This thesis addresses these limitations by defining a methodological approach for evaluating distributed DBMS on elastic infrastructures for the non-functional DBMS features performance, scalability, elasticity and availability. Further, we provide an advanced set of evaluation methods that build upon the established methodology, resulting in the novel DBMS evaluation framework *Mowgli* that fully automates the evaluation process.

7.1 Contribution

The Mowgli framework is the result of the two overarching research objectives addressed in this thesis: establishing methodological concepts for evaluation distributed DBMS on elastic infrastructures (RO-A) as presented in Chapter 4 and implementing novel methods for the automated evaluation of distributed DBMS on elastic infrastructures (RO-B) as presented in Chapter 5.

Methodological Concepts The first methodological contribution carries out an analysis of the relevant domain-specific *impact factors* that need to be considered for specifying and executing significant DBMS evaluations. The impact factors are classified into technology- and evaluation-specific impact factors of the DBMS, elastic resources and workload domain. The resulting unified view on domain-specific impact factors represents the foundation for establishing the methodological evaluation concepts.

Secondly, a set of eight cross-domain *evaluation principles* is established that are essential for advanced evaluation methods. These principles unify and extend established evaluation principles of the DBMS and elastic resource domains and pave the path for higher-level DBMS evaluation objectives. The resulting evaluation principles emphasize automation, reproducibility and portability as key principles to address the characteristics of the continuously evolving elastic resource and DBMS domains.

The third methodological contribution defines the comprehensive *evaluation design* for the evaluation objectives performance, scalability, elasticity and availability. For each evaluation objective, the design space is defined by the objective-specific metrics and the comprehensive evaluation process with its dedicated evaluation tasks. It is noteworthy that the evaluation design comprises a novel set of DBMS elasticity and availability metrics.

Evaluation Methods Based on the developed methodological DBMS evaluation concepts, this thesis provides the novel DBMS evaluation framework Mowgli that fully automates the DBMS evaluation on elastic infrastructures for the evaluation objectives performance and scalability. To enable the evaluation automation, Mowgli follows the concept of the comprehensive evaluation design and provides objective-specific Evaluation Scenario Templates (ESTs). An EST defines the required DBMS, elastic resource and workload properties and exposes the relevant domain-specific impact factors as configurable parameters. The ESTs represent the input for the Mowgli framework and enable the automated evaluation execution, including the resource allocation, DBMS deployment, workload execution and evaluation objective processing.

In order to allow the evaluation of higher-level evaluation objectives, Mowgli is enriched with the *Kaa* framework that enables and automates the *DBMS elasticity evaluation*. Therefore, Kaa provides support for DBMS adaptations at evaluation runtime which comprise the elastic scale-out and scale-in of the DBMS cluster. Moreover, Kaa supports the workload adaptation at evaluation runtime to emulate fluctuating workload patterns. Based on the higher-level elasticity metrics, Kaa is able to process them automatically by correlating basic performance metrics and additional meta-data.

The *King Louie* framework represents the second advancement of Mowgli and enables the automated *DBMS*

availability evaluation in the context of elastic resource failures. Therefore, King Louie provides a comprehensive resource failure injection framework and allows DBMS-specific recovery actions. Moreover, the developed availability metrics are automatically processed by the King Louie framework.

Validation The integrated frameworks Mowgli, Kaa and King Louie are validated based on a two-fold approach as presented in Chapter 6. First, Mowgli's applicability for realistic evaluation scenarios is validated based on a series of industry-driven case studies that target the evaluation objectives performance, scalability, elasticity and availability. The resulting 310 evaluation scenarios verify Mowgli's significance for evaluating cloud-hosted DBMS. Yet, to fully support the latest elastic resource types of the edge and fog domain, the design concepts and automation methods of Mowgli require extensions to cover these resource domains to their full extent. Secondly, Mowgli is validated against the established cross-domain evaluation principles. The results verify that Mowgli follows the established principles while emphasizing reproducibility, portability and automation for all evaluation objectives.

7.2 Future Research

Selecting the optimal DBMS operational model for data-intensive applications is a complex and manifold process that requires a methodological approach as presented in this thesis. The Mowgli framework provides a first method to support this decision process but also reveals some limitations that are discussed throughout the document. Therefore, we briefly summarize these limitations and point out future research that is envisioned to address these limitations. We classify these future research directions into *evaluation design*, *evaluation automation* and *evaluation analytics*.

Evaluation Design While the concepts of ESTs provides the model-based abstraction for designing evaluation scenarios, we envision that ESTs can be enhanced with sophisticated cloud modelling concepts to enable a higher level of abstraction and in consequence an even further reduction of the required domain knowledge. Moreover, extending ESTs with established cloud modelling concepts will enable the specification comprehensive performance models to define fine-grained service-level objectives for the target DBMS operational model.

Evaluation Automation In order to support the full scope of elastic resources, Mowgli's automation and orchestration capabilities need be extended to support the recent advances of edge and fog resources, enabling the DBMS evaluations of wide area DBMS deployments. Following this approach would also enable the evaluation of latency-specific failure scenarios in geo-replicated DBMS deployments.

Apart from that, Mowgli does not consider costs as evaluation objective. Yet, its modular architecture allows the seamless extension of the framework to measure the operational costs of the applied elastic resources and consider them as additional evaluation objective. With such an extension, Mowgli could be applied addressing complex capacity planning problems as well as for the direct comparison of self-hosted DBMS and established Database as a Service (DBaaS) providers.

Evaluation Analytics While Mowgli already provides basic data processing capabilities to automatically process the evaluation objective metrics, we see great potential in applying more sophisticated data analytic methods to process the resulting data sets. This would allow to identify parametric dependencies between the DBMS, resource and workload domain. At the same time, automatically deriving DBMS adaptation strategies for diverse workload patterns could be enabled. Moreover, first data analytic methods building upon Mowgli are already under development that enable the prediction of evaluation results by applying machine learning. Based on these future research directions, Mowgli could pave the path for a generic performance model of distributed DBMS on elastic infrastructures.

Acronyms

ANOVA Analysis of Variance

API Application Programming Interface

AWS Amazon Web Services

COT Cloud Orchestration Tool

CRUD Cread, Read, Update, Delete

DBA Database Administrator

DBaaS Database as a Service

DBMS Database Management System

DSL Domain Specific Language

EC2 Elastic Compute Cloud

EMF Eclipse Modelling Framework

ESP Evaluation Scenario Process

EST Evaluation Scenario Template

EP Evaluation Principle

ET Evaluation Task

HDD Hard Disk Drive

HTAP Hybrid Transaction-Analytical Processing

HWV Hardware Virtualization

IaaS Infrastructure as a Service

IoT Internet of Things

IT Information Technology

JSON JavaScript Object Notation

NIST National Institute of Standards and Technology

NVME Non-Volatile Memory Express

OLAP Online Analytical Processing

OLTP Online Transaction Processing

OS Operating System

OSV Operating System Virtualization

PaaS Plattform as a Service

RDBMS Relational Database Management System

REST Representational State Transfer

RO Research Objective

RQ Research Question

SaaS Software as a Service

SLA Service Level Agreement

SRL Scalability Rule Language

SSD Solid State Drive

TPC Transaction Processing Performance Council

VM Virtual Machine

YCSB Yahoo Cloud Serving Benchmark

Bibliography

[1] Donald D. Chamberlin and Raymond F. Boyce. “SEQUEL: A Structured English Query Language”. In: *Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*. SIGFIDET ’74. Ann Arbor, Michigan: Association for Computing Machinery, 1974, pp. 249–264. ISBN: 9781450374156. DOI: 10.1145/800296.811515.

[2] Gerald J. Popek and Robert P. Goldberg. “Formal Requirements for Virtualizable Third Generation Architectures”. In: *Commun. ACM* 17.7 (July 1974), pp. 412–421. ISSN: 0001-0782. DOI: 10.1145/361011.361073.

[3] E. F. Codd. “A Relational Model of Data for Large Shared Data Banks”. In: *Commun. ACM* 26.1 (Jan. 1983), pp. 64–69. ISSN: 0001-0782. DOI: 10.1145/357980.358007.

[4] Theo Haerder and Andreas Reuter. “Principles of Transaction-Oriented Database Recovery”. In: *ACM Comput. Surv.* 15.4 (Dec. 1983), pp. 287–317. ISSN: 0360-0300. DOI: 10.1145/289.291.

[5] Michael Stonebraker. “The case for shared nothing”. In: *IEEE Database Eng. Bull.* 9.1 (1986), pp. 4–9. URL: https://dsf.berkeley.edu/papers/hpts85-nothing.pdf.

[6] Jim Gray. “A View of Database System Performance Measures”. In: *SIGMETRICS Perform. Eval. Rev.* 15.1 (May 1987), pp. 3–4. ISSN: 0163-5999. DOI: 10.1145/29904.29905.

[7] Anne Geraci, Freny Katki, Louise McMonegal, Bennett Meyer, John Lane, Paul Wilson, Jane Radatz, Mary Yee, Hugh Porteous and Fredrick Springsteel. *IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries*. IEEE Press, 1991. ISBN: 1559370793.

[8] Raj Jain. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling*. Wiley professional computing. Wiley, 1991, pp. I–XXVII, 1–685. ISBN: 978-0-471-50336-1.

[9] Jim Gray. *Benchmark Handbook: For Database and Transaction Processing Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992. ISBN: 1558601597.

[10] C. Mohan, Don Haderle, Bruce Lindsay, Hamid Pirahesh and Peter Schwarz. “ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging”. In: *ACM Trans. Database Syst.* 17.1 (Mar. 1992), pp. 94–162. ISSN: 0362-5915. DOI: 10.1145/128765.128770.

[11] Jim Gray, Prakash Sundaresan, Susanne Englert, Ken Baclawski and Peter J. Weinberger. “Quickly Generating Billion-Record Synthetic Databases”. In: *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’94. Minneapolis, Minnesota, USA: Association for Computing Machinery, 1994, pp. 243–252. ISBN: 0897916395. DOI: 10.1145/191839.191886.

[12] Clark D French. “‘One size fits all’ database architectures do not work for DSS”. In: *ACM SIGMOD Record*. Vol. 24. 2. ACM. 1995, pp. 449–450. DOI: 10.1145/223784.223871.

[13] Surajit Chaudhuri and Umeshwar Dayal. “An Overview of Data Warehousing and OLAP Technology”. In: *SIGMOD Rec.* 26.1 (Mar. 1997), pp. 65–74. ISSN: 0163-5808. DOI: 10.1145/248603.248616.

[14] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine and Daniel Lewin. “Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web”. In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*. STOC ’97. El Paso, Texas, USA: Association for Computing Machinery, 1997, pp. 654–663. ISBN: 0897918886. DOI: 10.1145/258533.258660.

[15] “Towards Robust Distributed Systems (Abstract)”. In: *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*. PODC ’00. Portland, Oregon, USA: Association for Computing Machinery, 2000, p. 7. ISBN: 1581131836. DOI: 10.1145/343477.343502.

[16] Matthias Wiesmann, Fernando Pedone, André Schiper, Bettina Kemme and Gustavo Alonso. “Database replication techniques: A three parameter classification”. In: *Proceedings 19th IEEE Symposium on Reliable Distributed Systems SRDS-2000*. IEEE. 2000, pp. 206–215. DOI: 10.1109/RELDI.2000.885408.

[17] Matthias Wiesmann, Fernando Pedone, André Schiper, Bettina Kemme and Gustavo Alonso. “Understanding replication in databases and distributed systems”. In: *Proceedings 20th IEEE International Conference on Distributed Computing Systems*. IEEE. 2000, pp. 464–474. DOI: 10.1109/ICDCS.2000.840959.

[18] Hakan Hacigumus, Bala Iyer and Sharad Mehrotra. “Providing database as a service”. In: *Proceedings 18th International Conference on Data Engineering*. IEEE. 2002, pp. 29–38. DOI: 10.1109/ICDE.2002.994695.

[19] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt and Andrew Warfield. “Xen and the Art of Virtualization”. In: *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. SOSP ’03. Bolton Landing, NY, USA: Association for Computing Machinery, 2003, pp. 164–177. ISBN: 1581137575. DOI: 10.1145/945445.945462.

[20] Emmanuel Cecchet, Anupam Chanda, Sameh Elnikety, Julie Marguerite and Willy Zwaenepoel. “Performance comparison of middleware architectures for generating dynamic web content”. In: *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer. 2003, pp. 242–261. DOI: 10.1007/3-540-44892-6_13.

[21] Algirdas Avizienis, J-C Laprie, Brian Randell and Carl Landwehr. “Basic concepts and taxonomy of dependable and secure computing”. In: *IEEE transactions on dependable and secure computing* 1.1 (2004), pp. 11–33. DOI: 10.1109/TDSC.2004.2.

[22] M. Stonebraker and U. Cetintemel. “’One size fits all’: an idea whose time has come and gone”. In: *21st International Conference on Data Engineering (ICDE’05)*. 2005, pp. 2–11. DOI: 10.1109/ICDE.2005.1.

[23] Dror G Feitelson. *Experimental computer science: The need for a cultural change*. Tech. rep. School of Computer Science and Engineering, The Hebrew University of Jerusalem, 2006. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.76.1883&rep=rep1&type=pdf.

[24] Bianca Schroeder, Adam Wierman and Mor Harchol-Balter. "Open versus Closed: A Cautionary Tale". In: *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3*. NSDI'06. San Jose, CA: USENIX Association, 2006, p. 18. URL: https://static.usenix.org/events/nsdi06/tech/full_papers/schroeder/schroeder.pdf.

[25] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels. "Dynamo: Amazon's Highly Available Key-Value Store". In: *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*. SOSP '07. Stevenson, Washington, USA: Association for Computing Machinery, 2007, pp. 205–220. ISBN: 9781595935915. DOI: 10.1145/1294261.1294281.

[26] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin and Anthony Liguori. "KVM: the Linux Virtual Machine Monitor". In: *In Proceedings of the 2007 Ottawa Linux Symposium (OLS'-07*. 2007. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.423.3470&rep=rep1&type=pdf.

[27] Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem and Pat Helland. "The End of an Architectural Era: (It's Time for a Complete Rewrite)". In: *Proceedings of the 33rd International Conference on Very Large Data Bases*. VLDB '07. Vienna, Austria: VLDB Endowment, 2007, pp. 1150–1160. ISBN: 9781595936493. DOI: 10.1145/3226595.3226637.

[28] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[29] Anja Bog, Jens Kruger and Jan Schaffner. "A composite benchmark for online transaction processing and operational reporting". In: *2008 IEEE Symposium on Advanced Management of Information for Globalized Enterprises (AMIGE)*. IEEE. 2008, pp. 1–5. DOI: 10.1109/AMIGE.2008.ECP.30.

[30] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes and Robert E Gruber. "Bigtable: A distributed storage system for structured data". In: *ACM Transactions on Computer Systems (TOCS)* 26.2 (2008), p. 4. DOI: 10.1145/1365815.1365816.

[31] Dan Pritchett. "Base: An acid alternative". In: *Queue* 6.3 (2008), pp. 48–55. DOI: 10.1145/1394127.1394128.

[32] Will Sobel, Shanti Subramanyam, Akara Sucharitakul, Jimmy Nguyen, Hubert Wong, Arthur Klepchukov, Sheetal Patil, Armando Fox and David Patterson. "Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0". In: *Proc. of CCA*. Vol. 8. 2008, p. 228. URL: https://pdfs.semanticscholar.org/34dd/c3da70f5b17ae0a73266ad1e4f9ae155811f.pdf.

[33] Ming Zhong, Kai Shen and Joel Seiferas. "Replication Degree Customization for High Availability". In: *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*. Eurosys '08. Glasgow, Scotland UK: Association for Computing Machinery, 2008, pp. 55–68. ISBN: 9781605580135. DOI: 10.1145/1352592.1352599.

[34] Carsten Binnig, Donald Kossmann, Tim Kraska and Simon Loesing. "How is the Weather Tomorrow? Towards a Benchmark for the Cloud". In: *Proceedings of the Second International Workshop on Testing Database Systems*. DBTest '09. Providence, Rhode Island: Association for Computing Machinery, 2009. ISBN: 9781605587066. DOI: 10.1145/1594156.1594168.

[35] Daniela Florescu and Donald Kossmann. “Rethinking Cost and Performance of Database Systems”. In: *SIGMOD Rec.* 38.1 (June 2009), pp. 43–48. ISSN: 0163-5808. DOI: 10.1145/1558334.1558339.

[36] Patrick O’Neil, Elizabeth O’Neil, Xuedong Chen and Stephen Revilak. “The star schema benchmark and augmented fact table indexing”. In: *Technology Conference on Performance Evaluation and Benchmarking*. Springer. 2009, pp. 237–252. DOI: 10.1007/978-3-642-10424-4_17.

[37] Werner Vogels. “Eventually Consistent”. In: *Commun. ACM* 52.1 (Jan. 2009), pp. 40–44. ISSN: 0001-0782. DOI: 10.1145/1435417.1435432.

[38] Michael Armbrust et al. “A View of Cloud Computing”. In: *Commun. ACM* 53.4 (Apr. 2010), pp. 50–58. ISSN: 0001-0782. DOI: 10.1145/1721654.1721672.

[39] Aaron Beitch, Brandon Liu, Timothy Yung, Rean Griffith, Armando Fox and David A. Patterson. *Rain: A Workload Generation Toolkit for Cloud Computing Applications*. Tech. rep. UCB/EECS-2010-14. EECS Department, University of California, Berkeley, Feb. 2010. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-14.html.

[40] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan and Russell Sears. “Benchmarking Cloud Serving Systems with YCSB”. In: *Proceedings of the 1st ACM Symposium on Cloud Computing*. SoCC ’10. Indianapolis, Indiana, USA: Association for Computing Machinery, 2010, pp. 143–154. ISBN: 9781450300360. DOI: 10.1145/1807128.1807152.

[41] Daniel Ford, François Labelle, Florentina I. Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes and Sean Quinlan. “Availability in Globally Distributed Storage Systems”. In: OSDI’10 (2010), pp. 61–74.

[42] Avinash Lakshman and Prashant Malik. “Cassandra: a decentralized structured storage system”. In: *ACM SIGOPS Operating Systems Review* 44.2 (2010), pp. 35–40. DOI: 10.1145/1773912.1773922.

[43] Ang Li, Xiaowei Yang, Srikanth Kandula and Ming Zhang. “CloudCmp: Comparing Public Cloud Providers”. In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. IMC ’10. Melbourne, Australia: Association for Computing Machinery, 2010, pp. 1–14. ISBN: 9781450304832. DOI: 10.1145/1879141.1879143.

[44] Jörg Schad, Jens Dittrich and Jorge-Arnulfo Quiané-Ruiz. “Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance”. In: *Proc. VLDB Endow.* 3.1–2 (Sept. 2010), pp. 460–471. ISSN: 2150-8097. DOI: 10.14778/1920841.1920902.

[45] *TPC Benchmark ™C*. Version 5.11. Transaction Processing Performance Council (TPC). 2010. URL: http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf.

[46] Divyakant Agrawal, Amr El Abbadi, Sudipto Das and Aaron J Elmore. “Database scalability, elasticity, and autonomy in the cloud”. In: *International Conference on Database Systems for Advanced Applications*. Springer. Springer Berlin Heidelberg, 2011, pp. 2–15. DOI: 10.1007/978-3-642-20149-3_2.

[47] Matthew Aslett. *How will the database incumbents respond to NoSQL and NewSQL*. Tech. rep. 451 Research, 2011. URL: https://www.cs.cmu.edu/~pavlo/courses/fall2013/static/papers/aslett-newsql.pdf.

[48] David Bermbach and Stefan Tai. “Eventual Consistency: How Soon is Eventual? An Evaluation of Amazon S3’s Consistency Behavior”. In: *Proceedings of the 6th Workshop on Middleware for Service Oriented Computing*. MW4SOC ’11. Lisbon, Portugal: Association for Computing Machinery, 2011. ISBN: 9781450310673. DOI: 10.1145/2093185.2093186.

[49] Anja Bog, Hasso Plattner and Alexander Zeier. “A mixed transaction processing and operational reporting benchmark”. In: *Information Systems Frontiers* 13.3 (2011), pp. 321–335. DOI: 10.1007/s10796-010-9283-8.

[50] Anja Bog, Kai Sachs and Alexander Zeier. “Benchmarking Database Design for Mixed OLTP and OLAP Workloads”. In: *Proceedings of the 2nd ACM/SPEC International Conference on Performance Engineering*. ICPE ’11. Karlsruhe, Germany: Association for Computing Machinery, 2011, pp. 417–418. ISBN: 9781450305198. DOI: 10.1145/1958746.1958806.

[51] Rick Cattell. “Scalable SQL and NoSQL data stores”. In: *Acm Sigmod Record* 39.4 (2011), pp. 12–27. DOI: 10.1145/1978915.1978919.

[52] Richard Cole et al. “The Mixed Workload CH-benCHmark”. In: *Proceedings of the Fourth International Workshop on Testing Database Systems*. DBTest ’11. Athens, Greece: ACM, 2011, 8:1–8:6. ISBN: 978-1-4503-0655-3. DOI: 10.1145/1988842.1988850.

[53] Thibault Dory, Boris Mejıas, PV Roy and Nam-Luc Tran. “Measuring elasticity for cloud databases”. In: *Proceedings of the The Second International Conference on Cloud Computing, GRIDs, and Virtualization*. Citeseer. 2011, pp. 37–48. URL: https://www.info.ucl.ac.be/~pvr/CC2011elasticityCRfinal.pdf.

[54] Robin Hecht and Stefan Jablonski. “NoSQL Evaluation: A Use Case Oriented Survey”. In: *Proceedings of the 2011 International Conference on Cloud and Service Computing*. CSC ’11. USA: IEEE Computer Society, 2011, pp. 336–341. ISBN: 9781457716355. DOI: 10.1109/CSC.2011.6138544.

[55] Ioannis Konstantinou, Evangelos Angelou, Christina Boumpouka, Dimitrios Tsoumakos and Nectarios Koziris. “On the Elasticity of NoSQL Databases over Cloud Management Platforms”. In: *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*. CIKM ’11. Glasgow, Scotland, UK: Association for Computing Machinery, 2011, pp. 2385–2388. ISBN: 9781450307178. DOI: 10.1145/2063576.2063973.

[56] Peter Mell, Tim Grance et al. *The NIST definition of cloud computing*. Tech. rep. 2011. DOI: 10.6028/NIST.SP.800-145.

[57] Swapnil Patil, Milo Polte, Kai Ren, Wittawat Tantisiriroj, Lin Xiao, Julio López, Garth Gibson, Adam Fuchs and Billie Rinaldi. “YCSB++: Benchmarking and Performance Debugging Advanced Features in Scalable Table Stores”. In: *Proceedings of the 2nd ACM Symposium on Cloud Computing*. SOCC ’11. Cascais, Portugal: Association for Computing Machinery, 2011. ISBN: 9781450309769. DOI: 10.1145/2038916.2038925.

[58] Zia ur Rehman, Farookh K Hussain and Omar K Hussain. “Towards multi-criteria cloud service selection”. In: *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE. 2011, pp. 44–48. DOI: 10.1109/IMIS.2011.99.

[59] Michael Stonebraker and Rick Cattell. “10 Rules for Scalable Performance in “simple Operation” Datastores”. In: *Commun. ACM* 54.6 (June 2011), pp. 72–80. ISSN: 0001-0782. DOI: 10.1145/1953122.1953144.

[60] Steve Strauch, Oliver Kopp, Frank Leymann and Tobias Unger. “A taxonomy for cloud data hosting solutions”. In: *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*. IEEE. 2011, pp. 577–584. DOI: 10.1109/DASC.2011.106.

[61] Hiroshi Wada, Alan Fekete, Liang Zhao, Kevin Lee and Anna Liu. “Data Consistency Properties and the Trade-offs in Commercial Cloud Storage: the Consumers’ Perspective.” In: *CIDR*. Vol. 11. 2011, pp. 134–143. URL: http://www.echronos.systems/publications/nicta_full_text/4341.pdf.

[62] Daniel Abadi. “Consistency tradeoffs in modern distributed database system design: CAP is only part of the story”. In: *Computer* 45.2 (2012), pp. 37–42. DOI: 10.1109/MC.2012.33.

[63] Anja Bog, Kai Sachs and Hasso Plattner. “Interactive Performance Monitoring of a Composite OLTP and OLAP Workload”. In: *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’12. Scottsdale, Arizona, USA: Association for Computing Machinery, 2012, pp. 645–648. ISBN: 9781450312479. DOI: 10.1145/2213836.2213921.

[64] Flavio Bonomi, Rodolfo Milito, Jiang Zhu and Sateesh Addepalli. “Fog Computing and Its Role in the Internet of Things”. In: *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. MCC ’12. Helsinki, Finland: Association for Computing Machinery, 2012, pp. 13–16. ISBN: 9781450315197. DOI: 10.1145/2342509.2342513.

[65] Eric Brewer. “CAP twelve years later: How the” rules” have changed”. In: *Computer* 45.2 (2012), pp. 23–29. DOI: 10.1109/MC.2012.37.

[66] Avrilia Floratou, Jignesh M. Patel, Willis Lang and Alan Halverson. “When Free Is Not Really Free: What Does It Cost to Run a Database Workload in the Cloud?” In: *Topics in Performance Evaluation, Measurement and Characterization*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 163–179. ISBN: 978-3-642-32627-1. DOI: 10.1007/978-3-642-32627-1_12.

[67] Avrilia Floratou, Nikhil Teletia, David J. DeWitt, Jignesh M. Patel and Donghui Zhang. “Can the Elephants Handle the NoSQL Onslaught?” In: *Proc. VLDB Endow.* 5.12 (Aug. 2012), pp. 1712–1723. ISSN: 2150-8097. DOI: 10.14778/2367502.2367511.

[68] Enno Folkerts, Alexander Alexandrov, Kai Sachs, Alexandru Iosup, Volker Markl and Cafer Tosun. “Benchmarking in the cloud: What it should, can, and cannot be”. In: *Technology Conference on Performance Evaluation and Benchmarking*. Springer. 2012, pp. 173–188. DOI: 10.1007/978-3-642-36727-4_12.

[69] Markus Klems, David Bermbach and Rene Weinert. “A runtime quality measurement framework for cloud database service systems”. In: *2012 Eighth International Conference on the Quality of Information and Communications Technology*. IEEE. 2012, pp. 38–46. DOI: 10.1109/QUATIC.2012.17.

[70] Pramod J. Sadalage and Martin Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. 1st. Addison-Wesley Professional, 2012. ISBN: 0321826620.

[71] Michael Stonebraker. “New Opportunities for New SQL”. In: *Commun. ACM* 55.11 (Nov. 2012), pp. 10–11. ISSN: 0001-0782. DOI: 10.1145/2366316.2366319.

[72] Michael Stonebraker. *Newsql: An alternative to nosql and old sql for new oltp apps*. Blog@CACM. 2012. URL: https://cacm.acm.org/blogs/blog-cacm/109710-new-sql-an-alternative-to-nosql-and-old-sql-for-new-oltp-apps/comments?page=1 (visited on 03/04/2020).

[73] Smitha Sundareswaran, Anna Squicciarini and Dan Lin. “A brokerage-based approach for cloud service selection”. In: *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE. 2012, pp. 558–565. DOI: 10.1109/CLOUD.2012.119.

[74] Dennis Westermann, Jens Happe, Rouven Krebs and Roozbeh Farahbod. “Automated Inference of Goal-Oriented Performance Prediction Functions”. In: *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. ASE 2012. Essen, Germany: Association for Computing Machinery, 2012, pp. 190–199. ISBN: 9781450312042. DOI: 10.1145/2351676.2351703.

[75] Rodrigo F Almeida, Flávio RC Sousa, Sérgio Lifschitz and Javam C Machado. “On defining metrics for elasticity of cloud databases.” In: *SBBD (Short Papers)*. 2013, pp. 12–1. URL: https://sbbd2013.cin.ufpe.br/Proceedings/artigos/pdfs/sbbd_shp_12.pdf.

[76] Sérgio Almeida, João Leitão and Luıs Rodrigues. “ChainReaction: A Causal+ Consistent Datastore Based on Chain Replication”. In: *Proceedings of the 8th ACM European Conference on Computer Systems*. EuroSys ’13. Prague, Czech Republic: Association for Computing Machinery, 2013, pp. 85–98. ISBN: 9781450319942. DOI: 10.1145/2465351.2465361.

[77] Sumita Barahmand and Shahram Ghandeharizadeh. “BG: A Benchmark to Evaluate Interactive Social Networking Actions.” In: *6th Biennial Conference on Innovative Data Systems (CIDR)*. 2013. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.306.9222&rep=rep1&type=pdf.

[78] David Bermbach and Jörn Kuhlenkamp. “Consistency in distributed storage systems”. In: *Networked Systems*. Springer, 2013, pp. 175–189. DOI: 10.1007/978-3-642-40148-0_13.

[79] Tobias Binz, Uwe Breitenbücher, Florian Haupt, Oliver Kopp, Frank Leymann, Alexander Nowak and Sebastian Wagner. “OpenTOSCA–a runtime for TOSCA-based cloud applications”. In: *International Conference on Service-Oriented Computing*. Springer. 2013, pp. 692–695. DOI: 10.1007/978-3-642-45005-1_62.

[80] James C. Corbett et al. “Spanner: Google’s Globally Distributed Database”. In: *ACM Trans. Comput. Syst.* 31.3 (Aug. 2013). ISSN: 0734-2071. DOI: 10.1145/2491245.

[81] Francisco Cruz, Francisco Maia, Miguel Matos, Rui Oliveira, João Paulo, José Pereira and Ricardo Vilaça. “MeT: Workload Aware Elasticity for NoSQL”. In: *Proceedings of the 8th ACM European Conference on Computer Systems*. EuroSys ’13. Prague, Czech Republic: Association for Computing Machinery, 2013, pp. 183–196. ISBN: 9781450319942. DOI: 10.1145/2465351.2465370.

[82] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino and Philippe Cudre-Mauroux. “OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases”. In: *Proc. VLDB Endow.* 7.4 (Dec. 2013), pp. 277–288. ISSN: 2150-8097. DOI: 10.14778/2732240.2732246.

[83] Jörg Domaschka. “A comprehensive approach to transparent and flexible replication of Java services and applications”. PhD thesis. Universität Ulm. Fakultät für Ingenieurwissenschaften und Informatik, 2013. DOI: 10.18725/OPARU-2485.

[84] Nicolas Ferry, Franck Chauvel, Alessandro Rossini, Brice Morin and Arnor Solberg. “Managing Multi-Cloud Systems with CloudMF”. In: *Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies*. NordiCloud ’13. Oslo, Norway: Association for Computing Machinery, 2013, pp. 38–45. ISBN: 9781450323079. DOI: 10.1145/2513534.2513542.

[85] Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte and Hans-Arno Jacobsen. “BigBench: Towards an Industry Standard Benchmark for Big Data Analytics”. In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’13. New York, New York, USA: Association for Computing Machinery, 2013, pp. 1197–1208. ISBN: 9781450320375. DOI: 10.1145/2463676.2463712.

[86] Katarina Grolinger, Wilson A. Higashino, Abhinav Tiwari and Miriam AM Capretz. “Data management in cloud environments: NoSQL and NewSQL data stores”. In: *Journal of Cloud Computing: Advances, Systems and Applications* 2.1 (2013), p. 22. ISSN: 2192-113X. DOI: 10.1186/2192-113X-2-22.

[87] S. Kächele, C. Spann, F. J. Hauck and J. Domaschka. “Beyond IaaS and PaaS: An Extended Cloud Taxonomy for Computation, Storage and Networking”. In: *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*. 2013, pp. 75–82. DOI: 10.1109/UCC.2013.28.

[88] Markus Klems and Hoàng Anh Lê. “Position Paper: Cloud System Deployment and Performance Evaluation Tools for Distributed Databases”. In: *Proceedings of the 2013 International Workshop on Hot Topics in Cloud Services*. HotTopiCS ’13. Prague, Czech Republic: Association for Computing Machinery, 2013, pp. 63–70. ISBN: 9781450320511. DOI: 10.1145/2462307.2462322.

[89] Harold Lim, Yuzhang Han and Shivnath Babu. “How to Fit when No One Size Fits”. In: *6th Biennial Conference on Innovative Data Systems Research (CIDR ’13)*. Vol. 4. 2013. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.360.298&rep=rep1&type=pdf.

[90] M. Silva, M. R. Hines, D. Gallo, Q. Liu, K. D. Ryu and D. d. Silva. “CloudBench: Experiment Automation for Cloud Environments”. In: *2013 IEEE International Conference on Cloud Engineering (IC2E)*. 2013, pp. 302–311. DOI: 10.1109/IC2E.2013.33.

[91] D. Tsoumakos, I. Konstantinou, C. Boumpouka, S. Sioutas and N. Koziris. “Automated, Elastic Resource Provisioning for NoSQL Clusters Using TIRAMOLA”. In: *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. May 2013, pp. 34–41. DOI: 10.1109/CCGrid.2013.45.

[92] Chaitanya Baru, Milind Bhandarkar, Carlo Curino, Manuel Danisch, Michael Frank, Bhaskar Gowda, Hans-Arno Jacobsen, Huang Jie, Dileep Kumar, Raghunath Nambiar et al. “Discussion of BigBench: a proposed industry standard performance benchmark for big data”. In: *Technology Conference on Performance Evaluation and Benchmarking*. Springer. 2014, pp. 44–63. DOI: 10.1007/978-3-319-15350-6_4.

[93] David Bermbach. “Benchmarking Eventually Consistent Distributed Storage Systems”. PhD thesis. 2014. 183 pp. ISBN: 978-3-7315-0186-2. DOI: 10.5445/KSP/1000039389.

[94] David Bermbach, Jörn Kuhlenkamp, Akon Dey, Sherif Sakr and Raghunath Nambiar. "Towards an extensible middleware for database benchmarking". In: *Technology Conference on Performance Evaluation and Benchmarking*. Springer. 2014, pp. 82–96. DOI: 10.1007/978-3-319-15350-6_6.

[95] Tobias Binz, Uwe Breitenbücher, Oliver Kopp and Frank Leymann. "TOSCA: portable automated deployment and management of cloud applications". In: *Advanced Web Services*. Springer, 2014, pp. 527–549. DOI: 10.1007/978-1-4614-7535-4_22.

[96] Akon Dey, Alan Fekete, Raghunath Nambiar and Uwe Röhm. "YCSB+T: Benchmarking web-scale transactional databases". In: *2014 IEEE 30th International Conference on Data Engineering Workshops*. IEEE. 2014, pp. 223–230. DOI: 10.1109/ICDEW.2014.6818330.

[97] Nuno Diegues, Muhammet Orazov, João Paiva, Luıs Rodrigues and Paolo Romano. "Optimizing Hyperspace Hashing via Analytical Modelling and Adaptation". In: *SIGAPP Appl. Comput. Rev.* 14.2 (June 2014), pp. 23–35. ISSN: 1559-6915. DOI: 10.1145/2656864.2656866.

[98] Jörg Domaschka, Christopher B. Hauser and Benjamin Erb. "Reliability and Availability Properties of Distributed Database Systems". In: *Proceedings of the 2014 IEEE 18th International Enterprise Distributed Object Computing Conference*. EDOC '14. USA: IEEE Computer Society, 2014, pp. 226–233. ISBN: 9781479954704. DOI: 10.1109/EDOC.2014.38.

[99] Steffen Friedrich, Wolfram Wingerath, Felix Gessert and Norbert Ritter. "Nosql OLTP benchmarking: A survey". In: *Informatik 2014*. Bonn: Gesellschaft für Informatik e.V., 2014, pp. 693–704. URL: https://pdfs.semanticscholar.org/3d48/1573dec74303b00ac86a5e276eab67a1f0ab.pdf.

[100] Alexandru Iosup, Radu Prodan and Dick Epema. "Iaas cloud benchmarking: approaches, challenges, and experience". In: *Cloud Computing for Data-Intensive Applications*. Springer, 2014, pp. 83–104. DOI: 10.1007/978-1-4939-1905-5_4.

[101] Rouven Krebs, Christof Momm and Samuel Kounev. "Metrics and techniques for quantifying performance isolation in cloud environments". In: *Science of Computer Programming* 90 (2014). Special Issue on Component-Based Software Engineering and Software Architecture, pp. 116–134. ISSN: 0167-6423. DOI: https://doi.org/10.1016/j.scico.2013.08.003.

[102] K. Kritikos, J. Domaschka and A. Rossini. "SRL: A Scalability Rule Language for Multi-cloud Environments". In: *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*. Dec. 2014, pp. 1–9. DOI: 10.1109/CloudCom.2014.170.

[103] Jörn Kuhlenkamp, Markus Klems and Oliver Röss. "Benchmarking Scalability and Elasticity of Distributed Database Systems". In: *Proc. VLDB Endow.* 7.12 (Aug. 2014), pp. 1219–1230. ISSN: 2150-8097. DOI: 10.14778/2732977.2732995.

[104] Tania Lorido-Botran, Jose Miguel-Alonso and Jose A Lozano. "A review of auto-scaling techniques for elastic applications in cloud environments". In: *Journal of grid computing* 12.4 (2014), pp. 559–592. DOI: 10.1007/s10723-014-9314-7.

[105] Sherif Sakr. "Cloud-hosted databases: technologies, challenges and opportunities". In: *Cluster Computing* 17.2 (2014), pp. 487–502. ISSN: 1573-7543. DOI: 10.1007/s10586-013-0290-7.

[106] J. Scheuner, P. Leitner, J. Cito and H. Gall. "Cloud Work Bench – Infrastructure-as-Code Based Cloud Benchmarking". In: *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*. 2014, pp. 246–253. DOI: 10.1109/CloudCom.2014.98.

[107] Michael Stonebraker, Andrew Pavlo, Rebecca Taft and Michael L Brodie. "Enterprise database applications and the cloud: A difficult road ahead". In: *2014 IEEE International Conference on Cloud Engineering*. IEEE. 2014, pp. 1–6. DOI: 10.1109/IC2E.2014.97.

[108] Le Sun, Hai Dong, Farookh Khadeer Hussain, Omar Khadeer Hussain and Elizabeth Chang. "Cloud service selection: State-of-the-art and future research directions". In: *Journal of Network and Computer Applications* 45 (2014), pp. 134–150. DOI: 10.1016/j.jnca.2014.07.019.

[109] Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao and Daniel J. Abadi. "Fast Distributed Transactions and Strongly Consistent Replication for OLTP Database Systems". In: *ACM Trans. Database Syst.* 39.2 (May 2014). ISSN: 0362-5915. DOI: 10.1145/2556685.

[110] Blesson Varghese, Ozgur Akgun, Ian Miguel, Long Thai and Adam Barker. "Cloud benchmarking for performance". In: *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*. IEEE. 2014, pp. 535–540. DOI: 10.1109/CloudCom.2014.28.

[111] Max Chevalier, Mohammed El Malki, Arlind Kopliku, Olivier Teste and Ronan Tournier. "Benchmark for OLAP on NoSQL Technologies". In: *9th IEEE International Conference on Research Challenges in Information Science (IEEE RCIS 2015)*. Athens, Greece, May 2015, pp. 480–485. URL: https://hal.archives-ouvertes.fr/hal-01375413.

[112] Jörg Domaschka, Frank Griesinger, Daniel Baur and Alessandro Rossini. "Beyond Mere Application Structure Thoughts on the Future of Cloud Orchestration Tools". In: *Procedia Computer Science* 68 (2015). 1st International Conference on Cloud Forward: From Distributed to Complete Computing, pp. 151–162. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.09.231.

[113] Jörg Domaschka, Kyriakos Kritikos and Alessandro Rossini. "Towards a Generic Language for Scalability Rules". In: *Advances in Service-Oriented and Cloud Computing*. Ed. by Guadalupe Ortiz and Cuong Tran. Cham: Springer International Publishing, 2015, pp. 206–220. ISBN: 978-3-319-14886-1.

[114] Eugene Eberbach and Alex Reuter. "Toward El Dorado for Cloud Computing: Lightweight VMs, Containers, Meta-Containers and Oracles". In: *Proceedings of the 2015 European Conference on Software Architecture Workshops*. ECSAW '15. Dubrovnik, Cavtat, Croatia: Association for Computing Machinery, 2015. ISBN: 9781450333931. DOI: 10.1145/2797433.2797446.

[115] Wes Felter, Alexandre Ferreira, Ram Rajamony and Juan Rubio. "An updated performance comparison of virtual machines and linux containers". In: *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE. 2015, pp. 171–172. DOI: 10.1109/ISPASS.2015.7095802.

[116] Nikolas Roman Herbst, Samuel Kounev, Andreas Weber and Henning Groenda. "BUNGEE: an elasticity benchmark for self-adaptive IaaS cloud environments". In: *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE. 2015, pp. 46–56. DOI: 10.1109/SEAMS.2015.23.

[117] Zoltán Ádám Mann. "Allocation of Virtual Machines in Cloud Data Centers—A Survey of Problem Models and Optimization Algorithms". In: *ACM Comput. Surv.* 48.1 (Aug. 2015). ISSN: 0360-0300. DOI: 10.1145/2797211. URL: https://doi.org/10.1145/2797211.

[118] Roberto Morabito, Jimmy Kjällman and Miika Komu. "Hypervisors vs. lightweight virtualization: a performance comparison". In: *2015 IEEE International Conference on Cloud Engineering*. IEEE. 2015, pp. 386–393. DOI: 10.1109/IC2E.2015.74.

[119] Raghunath Nambiar and Meikel Poess. "Reinventing the TPC: from traditional to big data to internet of things". In: *Technology Conference on Performance Evaluation and Benchmarking*. Springer. 2015, pp. 1–7. DOI: 10.1007/978-3-319-31409-9_1.

[120] Linh Manh Pham, Alain Tchana, Didier Donsez, Noel De Palma, Vincent Zurczak and Pierre-Yves Gibello. "Roboconf: a hybrid cloud orchestrator to deploy complex applications". In: *2015 IEEE 8th International Conference on Cloud Computing*. IEEE. 2015, pp. 365–372. DOI: 10.1109/CLOUD.2015.56.

[121] Norbert Siegmund, Alexander Grebhahn, Sven Apel and Christian Kästner. "Performance-Influence Models for Highly Configurable Systems". In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2015. Bergamo, Italy: Association for Computing Machinery, 2015, pp. 284–294. ISBN: 9781450336758. DOI: 10.1145/2786805.2786845.

[122] *TPC Benchmark ™E*. Version 1.14.0. Transaction Processing Performance Council (TPC). 2015. URL: http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-e_v1.14.0.pdf.

[123] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune and John Wilkes. "Large-Scale Cluster Management at Google with Borg". In: *Proceedings of the Tenth European Conference on Computer Systems*. EuroSys '15. Bordeaux, France: Association for Computing Machinery, 2015. ISBN: 9781450332385. DOI: 10.1145/2741948.2741964.

[124] Miguel G Xavier, Israel C De Oliveira, Fabio D Rossi, Robson D Dos Passos, Kassiano J Matteussi and César AF De Rose. "A performance isolation analysis of disk-intensive workloads on container-based clouds". In: *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE. 2015, pp. 253–260. DOI: 10.1109/PDP.2015.67.

[125] Shanhe Yi, Zijiang Hao, Zhengrui Qin and Qun Li. "Fog computing: Platform and applications". In: *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*. IEEE. 2015, pp. 73–78. DOI: 10.1109/HotWeb.2015.22.

[126] Y. Zhang, J. Guo, E. Blais and K. Czarnecki. "Performance Prediction of Configurable Software Systems by Fourier Learning (T)". In: *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2015, pp. 365–373. DOI: 10.1109/ASE.2015.15.

[127] Daniel Abadi et al. "The Beckman Report on Database Research". In: *Commun. ACM* 59.2 (Jan. 2016), pp. 92–99. ISSN: 0001-0782. DOI: 10.1145/2845915. URL: https://doi.org/10.1145/2845915.

[128] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds and C. Rosenthal. "Chaos Engineering". In: *IEEE Software* 33.03 (May 2016), pp. 35–41. ISSN: 1937-4194. DOI: 10.1109/MS.2016.60.

[129] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer and John Wilkes. "Borg, Omega, and Kubernetes". In: *Queue* 14.1 (Jan. 2016), pp. 70–93. ISSN: 1542-7730. DOI: 10.1145/2898442.2898444.

[130] Jose Carrasco, Javier Cubo, Francisco Durán and Ernesto Pimentel. "Bidimensional cross-cloud management with TOSCA and Brooklyn". In: *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE. 2016, pp. 951–955. DOI: 10.1109/CLOUD.2016.0143.

[131] Ryan Chard, Kyle Chard, Bryan Ng, Kris Bubendorfer, Alex Rodriguez, Ravi Madduri and Ian Foster. "An automated tool profiling service for the cloud". In: *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE. 2016, pp. 223–232. DOI: 10.1109/CCGrid.2016.57.

[132] Haryadi S. Gunawi, Mingzhe Hao, Riza O. Suminto, Agung Laksono, Anang D. Satria, Jeffry Adityatama and Kurnia J. Eliazar. "Why Does the Cloud Stop Computing? Lessons from Hundreds of Service Outages". In: *Proceedings of the Seventh ACM Symposium on Cloud Computing*. SoCC '16. Santa Clara, CA, USA: Association for Computing Machinery, 2016, pp. 1–16. ISBN: 9781450345255. DOI: 10.1145/2987550.2987583. URL: https://doi.org/10.1145/2987550.2987583.

[133] Victor Heorhiadi, Shriram Rajagopalan, Hani Jamjoom, Michael K Reiter and Vyas Sekar. "Gremlin: Systematic resilience testing of microservices". In: *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2016, pp. 57–66. DOI: 10.1109/ICDCS.2016.11.

[134] Kyriakos Kritikos, Kostas Magoutis and Dimitris Plexousakis. "Towards knowledge-based assisted IaaS selection". In: *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE. 2016, pp. 431–439. DOI: 10.1109/CloudCom.2016.0073.

[135] Philipp Leitner and Jürgen Cito. "Patterns in the Chaos—A Study of Performance Variation and Predictability in Public IaaS Clouds". In: *ACM Trans. Internet Technol.* 16.3 (Apr. 2016). ISSN: 1533-5399. DOI: 10.1145/2885497. URL: https://doi.org/10.1145/2885497.

[136] Asraa Abdulrazak Ali Mardan and Kenji Kono. "Containers or hypervisors: Which is better for database consolidation?" In: *2016 IEEE international conference on cloud computing technology and science (CloudCom)*. IEEE. 2016, pp. 564–571. DOI: 10.1109/CloudCom.2016.0098.

[137] Andrew Pavlo and Matthew Aslett. "What's really new with NewSQL?" In: *ACM Sigmod Record* 45.2 (2016), pp. 45–55. DOI: 10.1145/3003665.3003674.

[138] Ilia Pietri and Rizos Sakellariou. "Mapping Virtual Machines onto Physical Machines in Cloud Computing: A Survey". In: *ACM Comput. Surv.* 49.3 (Oct. 2016). ISSN: 0360-0300. DOI: 10.1145/2983575.

[139] Prateek Sharma, Lucas Chaufournier, Prashant Shenoy and Y. C. Tay. "Containers and Virtual Machines at Scale: A Comparative Study". In: *Proceedings of the 17th International Middleware Conference*. Middleware '16. Trento, Italy: Association for Computing Machinery, 2016. ISBN: 9781450343008. DOI: 10.1145/2988336.2988337.

[140] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li and Lanyu Xu. "Edge computing: Vision and challenges". In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. DOI: 10.1109/JIOT.2016.2579198.

[141] Weisong Shi and Schahram Dustdar. "The promise of edge computing". In: *Computer* 49.5 (2016), pp. 78–81. DOI: 10.1109/MC.2016.145.

[142] Bruno Xavier, Tiago Ferreto and Luis Jersak. “Time provisioning evaluation of KVM, Docker and unikernels in a cloud platform”. In: *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE. 2016, pp. 277–280. DOI: 10.1109/CCGrid.2016.86.

[143] Matei Zaharia et al. “Apache Spark: A Unified Engine for Big Data Processing”. In: *Commun. ACM* 59.11 (Oct. 2016), pp. 56–65. ISSN: 0001-0782. DOI: 10.1145/2934664.

[144] Raja Appuswamy, Manos Karpathiotakis, Danica Porobic and Anastasia Ailamaki. “The Case For Heterogeneous HTAP”. In: *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. 2017. URL: http://cidrdb.org/cidr2017/papers/p21-appuswamy-cidr17.pdf.

[145] Andreas Bader, Oliver Kopp and Michael Falkenthal. “Survey and Comparison of Open Source Time Series Databases”. In: *Datenbanksysteme für Business, Technologie und Web (BTW 2017) - Workshopband*. Bonn: Gesellschaft für Informatik e.V., 2017, pp. 249–268. URL: https://dl.gi.de/handle/20.500.12116/922.

[146] David Bermbach, Jörn Kuhlenkamp, Akon Dey, Arunmoezhi Ramachandran, Alan Fekete and Stefan Tai. “BenchFoundry: a benchmarking framework for cloud storage services”. In: *International Conference on Service-Oriented Computing*. Springer. 2017, pp. 314–330. DOI: 10.1007/978-3-319-69035-3_22.

[147] David Bermbach, Frank Pallas, David Garcıa Pérez, Pierluigi Plebani, Maya Anderson, Ronen Kat and Stefan Tai. “A research perspective on Fog computing”. In: *International Conference on Service-Oriented Computing*. Springer. 2017, pp. 198–210. DOI: 10.1007/978-3-319-91764-1_16.

[148] David Bermbach, Erik Wittern and Stefan Tai. *Cloud service benchmarking*. Springer, 2017. DOI: 10.1007/978-3-319-55483-9.

[149] Fábio Coelho, João Paulo, Ricardo Vilaça, José Pereira and Rui Oliveira. “HTAPBench: Hybrid Transactional and Analytical Processing Benchmark”. In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. ICPE ’17. L’Aquila, Italy: Association for Computing Machinery, 2017, pp. 293–304. ISBN: 9781450344043. DOI: 10.1145/3030207.3030228.

[150] Felix Gessert, Wolfram Wingerath, Steffen Friedrich and Norbert Ritter. “NoSQL Database Systems: A Survey and Decision Guidance”. In: *Comput. Sci.* 32.3–4 (July 2017), pp. 353–365. ISSN: 1865-2034. DOI: 10.1007/s00450-016-0334-3.

[151] Søren Kejser Jensen, Torben Bach Pedersen and Christian Thomsen. “Time Series Management Systems: A Survey”. In: *IEEE Transactions on Knowledge and Data Engineering* 29.11 (2017), pp. 2581–2600. DOI: 10.1109/TKDE.2017.2740932.

[152] Z. Li, M. Kihl, Q. Lu and J. A. Andersson. “Performance Overhead Comparison between Hypervisor and Container Based Virtualization”. In: *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*. 2017, pp. 955–962. DOI: 10.1109/AINA.2017.79.

[153] Ashraf Mahgoub, Paul Wood, Sachandhan Ganesh, Subrata Mitra, Wolfgang Gerlach, Travis Harrison, Folker Meyer, Ananth Grama, Saurabh Bagchi and Somali Chaterji. "Rafiki: A Middleware for Parameter Tuning of NoSQL Datastores for Dynamic Metagenomics Workloads". In: *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*. Middleware '17. Las Vegas, Nevada: Association for Computing Machinery, 2017, pp. 28–40. ISBN: 9781450347204. DOI: 10.1145/3135974.3135991.

[154] Vincent Reniers, Dimitri Van Landuyt, Ansar Rafique and Wouter Joosen. "On the State of NoSQL Benchmarks". In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. ICPE '17 Companion. L'Aquila, Italy: Association for Computing Machinery, 2017, pp. 107–112. ISBN: 9781450348997. DOI: 10.1145/3053600.3053622.

[155] Vasily Tarasov, Lukas Rupprecht, Dimitris Skourtis, Amit Warke, Dean Hildebrand, Mohamed Mohamed, Nagapramod Mandagere, Wenji Li, Raju Rangaswami and Ming Zhao. "In search of the ideal storage configuration for Docker containers". In: *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE. 2017, pp. 199–206. DOI: 10.1109/FAS-W.2017.148.

[156] K. Velasquez, D. P. Abreu, D. Gonçalves, L. Bittencourt, M. Curado, E. Monteiro and E. Madeira. "Service Orchestration in Fog Environments". In: *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*. Aug. 2017, pp. 329–336. DOI: 10.1109/FiCloud.2017.49.

[157] Denis Weerasiri, Moshe Chai Barukh, Boualem Benatallah, Quan Z. Sheng and Rajiv Ranjan. "A Taxonomy and Survey of Cloud Resource Orchestration Techniques". In: *ACM Comput. Surv.* 50.2 (May 2017). ISSN: 0360-0300. DOI: 10.1145/3054177.

[158] Neeraja J. Yadwadkar, Bharath Hariharan, Joseph E. Gonzalez, Burton Smith and Randy H. Katz. "Selecting the Best VM across Multiple Public Clouds: A Data-Driven Performance Modeling Approach". In: *Proceedings of the 2017 Symposium on Cloud Computing*. SoCC '17. Santa Clara, California: Association for Computing Machinery, 2017, pp. 452–465. ISBN: 9781450350280. DOI: 10.1145/3127479.3131614.

[159] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song and Yingchun Yang. "BestConfig: Tapping the Performance Potential of Systems via Automatic Configuration Tuning". In: *Proceedings of the 2017 Symposium on Cloud Computing*. SoCC '17. Santa Clara, California: Association for Computing Machinery, 2017, pp. 338–350. ISBN: 9781450350280. DOI: 10.1145/3127479.3128605.

[160] Daniel J. Abadi and Jose M. Faleiro. "An Overview of Deterministic Database Systems". In: *Commun. ACM* 61.9 (Aug. 2018), pp. 78–88. ISSN: 0001-0782. DOI: 10.1145/3181853.

[161] Yazeed Alabdulkarim, Sumita Barahmand and Shahram Ghandeharizadeh. "BG: a scalable benchmark for interactive social networking actions". In: *Future Generation Computer Systems* 85 (2018), pp. 29–38. DOI: 10.1016/j.future.2018.02.031.

[162] Matt Aslett and Greg Zwakman. *Total Data market projected to reach $146bn by 2022*. 451 Research. 2018. URL: https://go.451research.com/2018-04-Total-Data-market-projected-146bn-by-2022.html (visited on 30/11/2019).

[163] Alexander Bergmayr, Uwe Breitenbücher, Nicolas Ferry, Alessandro Rossini, Arnor Solberg, Manuel Wimmer, Gerti Kappel and Frank Leymann. “A Systematic Review of Cloud Modeling Languages”. In: *ACM Comput. Surv.* 51.1 (Feb. 2018). ISSN: 0360-0300. DOI: 10.1145/3150227.

[164] Rajkumar Buyya et al. “A Manifesto for Future Generation Cloud Computing: Research Directions for the Next Decade”. In: *ACM Comput. Surv.* 51.5 (Nov. 2018). DOI: 10.1145/3241737. URL: https://doi.org/10.1145/3241737.

[165] Ali Davoudian, Liu Chen and Mengchi Liu. “A Survey on NoSQL Stores”. In: *ACM Comput. Surv.* 51.2 (Apr. 2018). ISSN: 0360-0300. DOI: 10.1145/3158661.

[166] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah and P. Merle. “Elasticity in Cloud Computing: State of the Art and Research Challenges”. In: *IEEE Transactions on Services Computing* 11.2 (Mar. 2018), pp. 430–447. ISSN: 2372-0204. DOI: 10.1109/TSC.2017.2711009.

[167] S. Eismann, J. Walter, J. von Kistowski and S. Kounev. “Modeling of Parametric Dependencies for Performance Prediction of Component-Based Software Systems at Run-Time”. In: *2018 IEEE International Conference on Software Architecture (ICSA)*. Apr. 2018, pp. 135–13509. DOI: 10.1109/ICSA.2018.00023.

[168] Martyn Ellison, Radu Calinescu and Richard F. Paige. “Evaluating cloud database migration options using workload models”. In: *Journal of Cloud Computing* 7.1 (2018), p. 6. ISSN: 2192-113X. DOI: 10.1186/s13677-018-0108-5.

[169] Michael Galloway, Gabriel Loewen, Jeffrey Robinson and Susan Vrbsky. “Performance of Virtual Machines Using Diskfull and Diskless Compute Nodes”. In: *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE. 2018, pp. 740–745. DOI: 10.1109/CLOUD.2018.00101.

[170] Yu Gao, Wensheng Dou, Feng Qin, Chushu Gao, Dong Wang, Jun Wei, Ruirui Huang, Li Zhou and Yongming Wu. “An Empirical Study on Crash Recovery Bugs in Large-Scale Distributed Systems”. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2018. Lake Buena Vista, FL, USA: Association for Computing Machinery, 2018, pp. 539–550. ISBN: 9781450355735. DOI: 10.1145/3236024.3236030.

[171] Jonathan Hasenburg, Sebastian Werner and David Bermbach. “FogExplorer”. In: *Proceedings of the 19th International Middleware Conference (Posters)*. Middleware ’18. Rennes, France: Association for Computing Machinery, 2018, pp. 1–2. ISBN: 9781450361095. DOI: 10.1145/3284014.3284015.

[172] Jonathan Hasenburg, Sebastian Werner and David Bermbach. “Supporting the Evaluation of Fog-Based IoT Applications During the Design Phase”. In: *Proceedings of the 5th Workshop on Middleware and Applications for the Internet of Things*. M4IoT’18. Rennes, France: Association for Computing Machinery, 2018, pp. 1–6. ISBN: 9781450361187. DOI: 10.1145/3286719.3286720.

[173] Nikolas Roman Herbst. “Methods and Benchmarks for Auto-Scaling Mechanisms in Elastic Cloud Environments”. PhD thesis. Universität Würzburg, 2018. URL: https://opus.bibliothek.uni-wuerzburg.de/opus4-wuerzburg/frontdoor/deliver/index/docId/16431/file/Herbst_Nikolas_Dissertation.pdf.

[174] Nikolas Herbst et al. “Quantifying Cloud Performance and Dependability: Taxonomy, Metric Design, and Emerging Challenges”. In: *ACM Trans. Model. Perform. Eval. Comput. Syst.* 3.4 (Aug. 2018). ISSN: 2376-3639. DOI: 10.1145/3236332.

[175] Michaela Iorga, Larry Feldman, Robert Barton, Michael J Martin, Nedim S Goren and Charif Mahmoudi. *Fog computing conceptual model*. Tech. rep. National Institute of Standards and Technology (NIST), 2018. DOI: 10.6028/NIST.SP.500-325.

[176] Kyriakos Kritikos and Geir Horn. “IaaS Service Selection Revisited”. In: *European Conference on Service-Oriented and Cloud Computing*. Springer. 2018, pp. 170–184. DOI: 10.1007/978-3-319-99819-0_13.

[177] Veronika Lesch, André Bauer, Nikolas Herbst and Samuel Kounev. “FOX: Cost-Awareness for Autonomic Resource Management in Public Clouds”. In: *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*. ICPE ’18. Berlin, Germany: Association for Computing Machinery, 2018, pp. 4–15. ISBN: 9781450350952. DOI: 10.1145/3184407.3184415.

[178] Charif Mahmoudi, Fabrice Mourlin and Abdella Battou. “Formal definition of edge computing: An emphasis on mobile cloud and IoT composition”. In: *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE. 2018, pp. 34–42. DOI: 10.1109/FMEC.2018.8364042.

[179] A. Papaioannou and K. Magoutis. “Replica-Group Leadership Change as a Performance Enhancing Mechanism in NoSQL Data Stores”. In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 2018, pp. 1448–1453. DOI: 10.1109/ICDCS.2018.00147.

[180] Mark Raasveldt, Pedro Holanda, Tim Gubner and Hannes Mühleisen. “Fair Benchmarking Considered Difficult: Common Pitfalls In Database Performance Testing”. In: *Proceedings of the Workshop on Testing Database Systems*. DBTest’18. Houston, TX, USA: Association for Computing Machinery, 2018. ISBN: 9781450358262. DOI: 10.1145/3209950.3209955.

[181] Kim-Thomas Rehmann and Enno Folkerts. “Performance of Containerized Database Management Systems”. In: *Proceedings of the Workshop on Testing Database Systems*. DBTest’18. Houston, TX, USA: Association for Computing Machinery, 2018. ISBN: 9781450358262. DOI: 10.1145/3209950.3209953.

[182] Joel Scheuner and Philipp Leitner. “A Cloud Benchmark Suite Combining Micro and Applications Benchmarks”. In: *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. ICPE ’18. Berlin, Germany: Association for Computing Machinery, 2018, pp. 161–166. ISBN: 9781450356299. DOI: 10.1145/3185768.3186286.

[183] Joel Scheuner and Philipp Leitner. “Estimating cloud application performance based on micro-benchmark profiling”. In: *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE. 2018, pp. 90–97. DOI: 10.1109/CLOUD.2018.00019.

[184] Merlijn Sebrechts, Gregory Van Seghbroeck, Tim Wauters, Bruno Volckaert and Filip De Turck. “Orchestrator conversation: Distributed management of cloud applications”. In: *International Journal of Network Management* 28.6 (2018), e2036. DOI: 10.1002/nem.2036.

[185] Selome Kostentinos Tesfatsion, Cristian Klein and Johan Tordsson. “Virtualization Techniques Compared: Performance, Resource, and Power Usage Overheads in Clouds”. In: *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*. ICPE ’18. Berlin, Germany: Association for Computing Machinery, 2018, pp. 145–156. ISBN: 9781450350952. DOI: 10.1145/3184407.3184414.

[186] *TPC Benchmark ™H*. Version 2.18.0. Transaction Processing Performance Council (TPC). 2018. URL: http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.18.0.pdf.

[187] A. Bauer, S. Eismann, J. Grohmann, N. Herbst and S. Kounev. “Systematic Search for Optimal Resource Configurations of Distributed Applications”. In: *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. June 2019, pp. 120–125. DOI: 10.1109/FAS-W.2019.00040.

[188] A. Bauer, N. Herbst, S. Spinner, A. Ali-Eldin and S. Kounev. “Chameleon: A Hybrid, Proactive Auto-Scaling Mechanism on a Level-Playing Field”. In: *IEEE Transactions on Parallel and Distributed Systems* 30.4 (Apr. 2019), pp. 800–813. ISSN: 2161-9883. DOI: 10.1109/TPDS.2018.2870389.

[189] Antonia Bertolino, Guglielmo De Angelis, Micael Gallego, Boni Garcıa, Francisco Gortázar, Francesca Lonetti and Eda Marchetti. “A Systematic Review on Cloud Testing”. In: *ACM Comput. Surv.* 52.5 (Sept. 2019). ISSN: 0360-0300. DOI: 10.1145/3331447.

[190] Lexi Brent and Alan Fekete. “A Versatile Framework for Painless Benchmarking of Database Management Systems”. In: *Databases Theory and Applications*. Cham: Springer International Publishing, 2019, pp. 45–56. ISBN: 978-3-030-12079-5. DOI: 10.1007/978-3-030-12079-5_4.

[191] Emiliano Casalicchio. “Container Orchestration: A Survey”. In: *Systems Modeling: Methodologies and Tools*. Springer, 2019, pp. 221–235. DOI: 10.1007/978-3-319-92378-9_14.

[192] S. Eismann, J. Kistowski, J. Grohmann, A. Bauer, N. Schmitt and S. Kounev. “TeaStore - A Micro-Service Reference Application”. In: *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. June 2019, pp. 263–264. DOI: 10.1109/FAS-W.2019.00073.

[193] Donald Feinberg, Merv Adrian and Adam Ronthal. “The Future of the DBMS Market Is Cloud”. In: *Gartner, Inc.* June (2019), pp. 1–14. URL: https://pages.awscloud.com/Gartner-The-Future-of-the-DBMS-Market-Is-Cloud.html.

[194] Cheol-Ho Hong and Blesson Varghese. “Resource Management in Fog/Edge Computing: A Survey on Architectures, Infrastructure, and Algorithms”. In: *ACM Comput. Surv.* 52.5 (Sept. 2019). ISSN: 0360-0300. DOI: 10.1145/3326066.

[195] Murtadha Al Hubail, Ali Alsuliman, Michael Blow, Michael Carey, Dmitry Lychagin, Ian Maxon and Till Westmann. “Couchbase Analytics: NoETL for Scalable NoSQL Data Analysis”. In: *Proc. VLDB Endow.* 12.12 (Aug. 2019), pp. 2275–2286. ISSN: 2150-8097. DOI: 10.14778/3352063.3352143.

[196] Jinho Jung, Hong Hu, Joy Arulraj, Taesoo Kim and Woonhak Kang. “APOLLO: Automatic Detection and Diagnosis of Performance Regressions in Database Systems”. In: *Proc. VLDB Endow.* 13.1 (Sept. 2019), pp. 57–70. ISSN: 2150-8097. DOI: 10.14778/3357377.3357382.

[197] S. Kim and Y. S. Kanwar. “GeoYCSB: A Benchmark Framework for the Performance and Scalability Evaluation of NoSQL Databases for Geospatial Workloads”. In: *2019 IEEE International Conference on Big Data (Big Data)*. 2019, pp. 3666–3675. DOI: 10.1109/BigData47090.2019.9005570.

[198] Jiaheng Lu and Irena Holubová. "Multi-Model Databases: A New Journey to Handle the Variety of Data". In: *ACM Comput. Surv.* 52.3 (June 2019). ISSN: 0360-0300. DOI: 10.1145/3323214.

[199] Ilias Mavridis and Helen Karatza. "Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing". In: *Future Generation Computer Systems* 94 (2019), pp. 674–696. DOI: 10.1016/j.future.2018.12.035.

[200] Jonathan McChesney, Nan Wang, Ashish Tanwer, Eyal de Lara and Blesson Varghese. "DeFog: Fog Computing Benchmarks". In: SEC '19 (2019), pp. 47–58. DOI: 10.1145/3318216.3363299.

[201] A. V. Papadopoulos, L. Versluis, A. Bauer, N. Herbst, J. Von Kistowski, A. Ali-eldin, C. Abad, J. N. Amaral, P. T ma and A. Iosup. "Methodological Principles for Reproducible Performance Evaluation in Cloud Computing". In: *IEEE Transactions on Software Engineering* (2019), pp. 1–1. ISSN: 0098-5589. DOI: 10.1109/TSE.2019.2927908.

[202] Pini Reznik, Michelle Gienow and Jamie Dobson. *Cloud Native Patterns - Practical Patterns for Innovation*. Sebastopol, California: O'Reilly Media, Incorporated, 2019. ISBN: 978-1-492-04890-9.

[203] Noa Roy-Hubara, Peretz Shoval and Arnon Sturm. "A Method for Database Model Selection". In: *Enterprise, Business-Process and Information Systems Modeling*. Cham: Springer International Publishing, 2019, pp. 261–275. ISBN: 978-3-030-20618-5. DOI: 10.1007/978-3-030-20618-5_18.

[204] Daniel Seybold, Volker Foth, Feroz Zahid, Pawel Skrzypek, Marcin Prusinski and Jörg Domaschka. *D4.6 Data Processing Layer*. 2019. URL: http://melodic.cloud/deliverables/D4.6%20Data%20Processing%20Layer.pdf.

[205] Junjay Tan, Thanaa Ghanem, Matthew Perron, Xiangyao Yu, Michael Stonebraker, David DeWitt, Marco Serafini, Ashraf Aboulnaga and Tim Kraska. "Choosing a Cloud DBMS: Architectures and Tradeoffs". In: *Proc. VLDB Endow.* 12.12 (Aug. 2019), pp. 2170–2182. ISSN: 2150-8097. DOI: 10.14778/3352063.3352133.

[206] *TPC Express Big Bench (TPCx-BB)*. Version 1.3.1. Transaction Processing Performance Council (TPC). 2019. URL: http://www.tpc.org/tpc_documents_current_versions/pdf/tpcx-bb_v1.3.1.pdf.

[207] Eddy Truyen, Dimitri Van Landuyt, Davy Preuveneers, Bert Lagaisse and Wouter Joosen. "A Comprehensive Feature Comparison Study of Open-Source Container Orchestration Frameworks". In: *Applied Sciences* 9.5 (2019). ISSN: 2076-3417. DOI: 10.3390/app9050931.

[208] W. Xiong, K. Yang and H. Dai. "Improving NoSQL's Performance Metrics via Machine Learning". In: *2019 Seventh International Conference on Advanced Cloud and Big Data (CBD)*. Sept. 2019, pp. 90–95. DOI: 10.1109/CBD.2019.00026.

[209] Yang Yang, Qiang Cao and Hong Jiang. "EdgeDB: An Efficient Time-Series Database for Edge Computing". In: *IEEE Access* 7 (2019), pp. 142295–142307. DOI: 10.1109/ACCESS.2019.2943876.

[210] Y. Zhu and J. Liu. "ClassyTune: A Performance Auto-Tuner for Systems in the Cloud". In: *IEEE Transactions on Cloud Computing* (2019). ISSN: 2372-0018. DOI: 10.1109/TCC.2019.2936567.

[211] Daniel Abadi et al. "The Seattle Report on Database Research". In: *SIGMOD Rec.* 48.4 (Feb. 2020), pp. 44–53. ISSN: 0163-5808. DOI: 10.1145/3385658.3385668.

[212] Dušan Okanovi , Samuel Beck, Lasse Merz, Christoph Zorn, Leonel Merino, André van Hoorn and Fabian Beck. “Can a Chatbot Support Software Engineers with Load Testing? Approach and Experiences”. In: *Proceedings of the 2020 ACM/SPEC International Conference on Performance Engineering*. ICPE ’20. Edmonton, Canada: Association for Computing Machinery, 2020. DOI: 10.1145/3358960.3375792.

[213] *OpenAPI Specification*. Version 3.0.3. OpenAPI Initiative. 2020. URL: http://spec.openapis.org/oas/v3.0.3.

[214] *TPC Benchmark ™DS*. Version 2.13.0. Transaction Processing Performance Council (TPC). 2020. URL: http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.13.0.pdf.

Part II

Publications

Chapter 8

[core1] A survey on data storage and placement methodologies for Cloud-Big Data ecosystem

This article is published as follows:

Somnath Mazumdar, Daniel Seybold, Kyriakos Kritikos, and Yiannis Verginadis. "A survey on data storage and placement methodologies for Cloud-Big Data ecosystem" in *Journal of Big Data*, 6.1, Feb. 2019, p. 15, issn 2196-1115, DOI: https://doi.org/10.1186/s40537-019-0178-3.

Mazumdar *et al. J Big Data* (2019) 6:15
https://doi.org/10.1186/s40537-019-0178-3

Journal of Big Data

SURVEY PAPER Open Access

A survey on data storage and placement methodologies for Cloud-Big Data ecosystem

Somnath Mazumdar[1], Daniel Seybold[2], Kyriakos Kritikos[3*] and Yiannis Verginadis[4]

*Correspondence:
kritikos@ics.forth.gr
[3] ICS-FORTH, Heraklion, Crete, Greece
Full list of author information is available at the end of the article

Abstract

Currently, the data to be explored and exploited by computing systems increases at an exponential rate. The massive amount of data or so-called "Big Data" put pressure on existing technologies for providing scalable, fast and efficient support. Recent applications and the current user support from multi-domain computing, assisted in migrating from data-centric to knowledge-centric computing. However, it remains a challenge to optimally store and place or migrate such huge data sets across data centers (DCs). In particular, due to the frequent change of application and DC behaviour (i.e., resources or latencies), data access or usage patterns need to be analyzed as well. Primarily, the main objective is to find a better data storage location that improves the overall data placement cost as well as the application performance (such as throughput). In this survey paper, we are providing a state of the art overview of Cloud-centric Big Data placement together with the data storage methodologies. It is an attempt to highlight the actual correlation between these two in terms of better supporting Big Data management. Our focus is on management aspects which are seen under the prism of non-functional properties. In the end, the readers can appreciate the deep analysis of respective technologies related to the management of Big Data and be guided towards their selection in the context of satisfying their non-functional application requirements. Furthermore, challenges are supplied highlighting the current gaps in Big Data management marking down the way it needs to evolve in the near future.

Keywords: Big Data, Cloud, Data models, Data storage, Placement

Introduction

Over the time, the type of applications has evolved from batch, compute or memory intensive applications to streaming or even interactive applications. As a result, applications are getting more complex and become long-running. Such applications might require frequent-access to multiple distributed data sources. During application deployment and provisioning, the user can face various issues such as (i) where to effectively place both the data and the computation; (ii) how to achieve required objectives while reducing the overall application running cost. Data could be generated from various sources, including a multitude of devices over IoT environments that can generate a huge amount of data, while the applications are running. An application can further produce a large amount of data. In general, data of such size is usually referred to as *Big Data*. In general, Big Data is characterised by five properties [1, 2]. These are *volume*, *velocity* (means rapid update and propagation of data), *variety* (means different kinds of

data parts), *veracity* (related to the trustworthiness, authenticity and protection (degree) of the data) and *value* (the main added-value and the importance of the data to the business). A large set of different data types generated from various sources can hold enormous information (in the form of relationships [3], system access logs, and also as the quality of services (QoSs)). Such knowledge can be critical for improving both products and services. Thus, to retrieve the underlying knowledge from such big sized data sets an efficient data processing ecosystem and knowledge filtering methodologies are needed.

In general, Cloud-based technology offers different solutions over different levels of abstractions to build and dynamically provision user applications. The Cloud offers suitable frameworks for the clustering of Big Data as well as efficiently distributed databases for their storage and placement. However, the native Cloud facilities have a lack of guidance on how to combine and integrate services in terms of holistic frameworks which could enable users to properly manage both their applications and the data. While there exist some promising efforts that fit well under the term *Big Data-as-a-service (BDaaS)*, most of them still lack adequate support for: data-privacy [4–6], query optimisation [7], robust data analytics [8] and data-related service level objective management for increased (Big Data) application quality [9]. Currently, the application placement and management over multi or cross-Clouds is being researched. However, the additional dimension of Big Data management does raise significantly the complexity of finding adequate and realistic solutions.

The primary goal of this survey is to present the current state-of-affairs in Cloud computing with respect to the Big Data management (mainly storage and placement) from the application's administration point-of-view. To this end, we have thoroughly reviewed the proposed solutions based on the placement and storage of Big Data through the use of a carefully designed set of criteria. Such criteria were devised under the prism of non-functional properties. This was performed in an attempt to unveil those solutions which can be deemed suitable for the better management of different kinds of applications (while taking into consideration non-functional aspects). In the end, the prospective readers (such as Big Data application owners, DevOps) can be guided towards the selection of those solutions in each Big Data management lifecycle phase (focused in this article) that satisfy in a better way their non-functional application requirements. The analysis finally concludes with the identification of certain gaps. Based on the latter, a set of challenges for the two Big Data management phases covered as well as for Big Data management as a whole are supplied towards assisting in the evolution of respective solutions and paving the way for the actual directions that the research should follow.

Based on the above analysis, it is clear that this article aims at providing guidance to potential adopters concerning the most appropriate solution for both placing and storing Big Data (according to the distinctive requirements of the application domain). To this end, our work can be considered as complementary to other relevant surveys that attempt to review Big Data technologies. In particular, the past surveys have focused on the deployment of data-intensive applications in the Cloud [10], on assessing various database management tools for storing Big Data [11], on evaluating the technologies developed for Big Data applications [12], on Cloud-centric distributed database management systems (primarily on NoSQL storage models) [13], on design principles for in-memory Big Data management and processing [14] and on research challenges related

to Big Data in the Cloud ecosystem [15]. However, the primary focus of these surveys is mainly on functional aspects examined under the prism of analysing different dimensions and technology types related to Big Data. Further, there is no clear discussion on management aspects in the context of the whole Big Data management lifecycle as usually the focus seems to be merely on the Big Data storage phase. Interestingly, our survey deeply analyses those phases in the Big Data management lifecycle that are the most crucial in the context of satisfying application non-functional requirements.

The remaining part of this manuscript is structured as follows: "Data lifecycle management (DLM)" section explicates how data modelling can be performed, analyses various data management lifecycle models and comes up with an ideal one which is presented along with the proper architecture to support it. Next, "Methodology" section attempts to explain this survey's main methodology. "Non-functional data management features" section details the main non-functional features of focus in this article. Based on these features, the review of Big Data storage systems and distributed file systems are supplied in "Data storage systems" section. Similarly, the review of state-of-the-art data placement techniques is performed in "Data placement techniques" section. Next, "Lessons learned and future research directions" section presents relevant lessons learned as well as certain directions for future research work and finally "Concluding remarks" section concludes the survey paper.

Data lifecycle management (DLM)

Data lifecycle models

There exist two types of data lifecycle models focusing on either general data or Big Data management. The generic data management lifecycles usually cover activities such as generation, collection (curation), storage, publishing, discovery, processing and analysis of data [16].

In general, Big Data lifecycle models primarily comprises activities (such as data collection, data loading, data processing, data analysis and data visualisation [17, 18]). It is worth to note that apart from the data visualisation, they do share many identical activities with the generic ones. However, such models do not mention the value of data.

To counter this, the NIST reference model [19] suggests four data management phases: *collection*, *preparation*, *analysis* and *action*, where the *action* phase is related to using synthesised knowledge to create value (represents analytics and visualisation of knowledge). Furthermore, focusing more on the data value, OECD [20] has proposed a data value cycle model comprising six phases: *datafication and data collection*, *Big Data*, *data analytics*, *knowledge base*, *decision making* and *valued-added* for growth and well-being. The model forms an iterative, closed feedback loop where results from Big Data analytics are fed back to the respective database. Later, the work in [21] exposed the main drawbacks of OECD and proposed a new reference model that adds two additional components, the business intelligence (BI) system and the environment, into the OECD model. The data interaction and analysis formulates a short closed loop in the model. A greater loop is also endorsed via the BI's iterative interaction and observation of its environment. Finally, it is claimed that the management of Big Data for value creation is also linked to the BI management. In this way, Big Data management is related

directly to the activities of data integration, analysis, interaction and effectuation along with the successful management of the emergent knowledge via data intelligence.

Data modelling

The data needs to be described in an appropriate form prior to any kind of usage. The information used for the data description is termed as metadata (i.e., data about data) [22–24]. The use of metadata enriches data management so that it can properly support and improve any data management activity. Two major issues related to metadata management are:

- *How should metadata be described (or characterised)?* The description of a metadata schema which can be exploited to efficiently place a certain Big Data application in multiple Clouds by respecting both user constraints and requirements. Such a metadata schema has been proposed partially in [25] or completely in [26].
- *How should metadata be efficiently managed and stored for better retrieval and exploitation?* The design of appropriate languages [27, 28] that focus on the description of how Big Data applications and data should be placed and migrated across different multiple Cloud resources.

For a better description of metadata, the authors in [22] identify available Cloud services and analyse some of their main characteristics following a tree-structured taxonomy. Another relevant effort is the DICE project [25] that focuses on the quality-driven development of Big Data applications. It offers a UML profile along with the appropriate tools that may assist software designers to reason about the reliability, safety and efficiency of data-intensive applications. Specifically, it has introduced a metamodel for describing certain aspects of Big Data-intensive applications.

Most of these efforts do not offer a direct support for expressing significant aspects of Big Data, such as data origin, location, volume, transfer rates or even aspects of the operations that transfer data between Cloud resources. One effort that tries to cover the requirements for a proper and complete metadata description is the Melodic metadata schema [26]. This schema refers to a taxonomy of concepts, properties and relationships that can be exploited for supporting Big Data management as well as application deployment reasoning. The schema is clustered into three parts: (i) one focusing on specifying Cloud service requirements and capabilities to support application deployment reasoning; (ii) another focusing on defining Big Data features and constraints to support Big Data management; (iii) a final one concentrating on supplying Big Data security-related concepts to drive the data access control.

With respect to the second direction of work, although several languages are currently used for capturing application placement and reconfiguration requirements (e.g., TOSCA [27]), a lack of distinct support for describing placement and management requirements for Big Data can be observed. However, if such languages are extended through the possible use of a metadata schema, then they could be able to achieve this purpose. This has been performed in [26], where a classical, state-of-the-art Cloud description language called CAMEL [29] has been extended to enable the description of Big Data placement and management requirements by following a feature-model-based

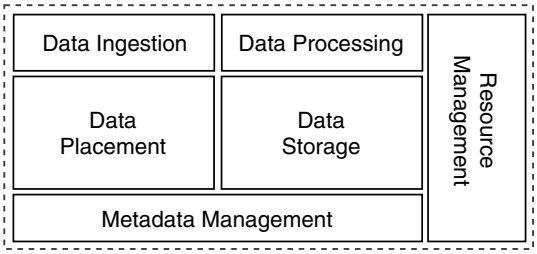


Fig. 1 A high-level block diagram of a Big Data management system

approach where requirements are expressed as features or attributes that are annotated via elements from the metadata schema.

Data lifecycle management systems

Traditional data lifecycle management systems (DLMSs) focus more on the way data is managed and not on how they are processed. In particular, the actual main services that they offer are data storage planning (and provisioning) and data placement (and execution support) via efficient data management policies. On the other hand, it seems that data processing is covered by other tools or systems as it is regarded as application-specific. Traditionally in Cloud, Big Data processing is offered as a separate service, while the resource management is usually handled by other tools, such as Apache Mesos or YARN. Figure 1 depicts the architecture of a system that completely addresses the data management lifecycle, as inscribed in the previous sub-section. This system comprises six primary components.

- *Metadata management* takes care of maintaining information which concerns both the static and dynamic characteristics of data. It is the cornerstone for enabling efficient data management.
- *Data placement* encapsulates the main methods for efficient data placement and data replication while satisfying user requirements.
- *Data storage* is responsible for proper (transactional) storage and efficient data retrieval support.
- *Data ingestion* enables importing and exporting the data over the respective system.
- *Big Data processing* supports the efficient and clustered processing of Big Data by executing the main logic of the user application(s).
- *Resource management* is responsible for the proper and efficient management of computational resources.

In this article, our focus is mainly on the *Data storage* and *Data placement* parts of the above architecture. Our rationale is that the integration of such parts (or Big Data lifecycle management phases) covers the core of a DLMS. An application's data access workflow in the Cloud is presented in Fig. 2. As a first step, the application checks the availability of the input data. In general, the data needs to be known by the system to optimally handle it. It maps to two main cases: (i) data already exist and have been registered; (ii) data do not exist and must be registered. In the latter case, metadata is needed

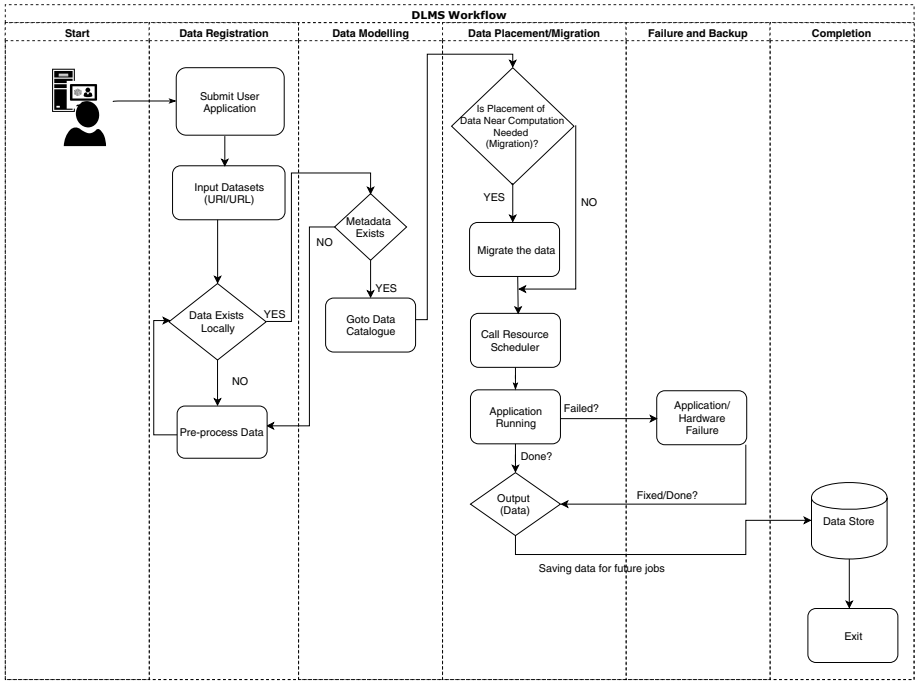


Fig. 2 Standard workflow of application data lifecycle

to register the data into the system (thus mapping to the data-registration process). During the data modelling (see "Data modelling" sub-section), the metadata are maintained via a data catalogue (i.e., a special realisation of *Metadata management* component). Such an approach can guarantee the efficient maintenance of application data throughout the application's lifecycle by both knowing and dynamically altering the values of data features (such as data type, size, location, data format, user preference, data replica numbers, cost constraints) whenever needed. In the next phase, based on the employed data placement methodology, the data is placed/migrated next to the application or both the data and application code is collocated. Here, the underlying scheduler (realising the *Data placement* component) acquires the up-to-date data knowledge to achieve an efficient data placement during both the initial application deployment and its runtime. Such an approach can restrain unnecessary data movement and reduces cost (at runtime) [30–32]. Next, during the application execution, two situations may arise: (i) new data sets are generated; (ii) data sets are transformed into another form (such as data compression). Furthermore, temporary data may also need to be handled. Finally, once application execution ends, the generated or transformed data needs to be stored (or backed up) as per user instructions.

In general, a hierarchical storage management [33] could be considered as a DLMS tool. In recent times, cognitive data management (CDM) has gained industrial support for automated data management together with high-grade efficiency. The CDM

(e.g., Stronglink[1]) is generally the amalgamation of intelligent (artificial-intelligence[2]/ machine learning-based approach) distributed storage including resource management together with a more sophisticated DLMS component. The CDM works on the database-as-a-service (DBaaS) layer which instructs the data to be used by the scheduler with an efficient management approach including the exploitation of the data catalogue via data modelling.

Methodology

We have conducted a systematic literature review (SLR) on Big Data placement and storage methods in the Cloud, following the guidelines proposed in [34]. Such an SLR comprises three main phases: (i) SLR planning, (ii) SLR conduction and (iii) SLR reporting. In this section, we briefly discuss the first two phases. While the remaining part of this manuscript focuses on the presenting the survey, the identification of the remaining research issues and the potential challenges for current and future work.

SLR planning

This phase comprises three main steps: (i) SLR need identification, (ii) research questions identification and (iii) SLR protocol formation.

SLR need identification

Here, we are advocating to add more focus on the Big Data storage and placement phases of the respective Big Data management lifecycle. Thus be able to confront the respective challenges that Big Data place on them. Such phases are also the most crucial in the attempt to satisfy the non-functional requirements of Big Data applications. The primary focus of this survey is over storage and placement phases. It is an attempt to examine if they are efficiently and effectively realised by current solutions and approaches. The twofold advantage of identifying the efficient ways to manage and store Big Data are: (i) practitioners can select the most suitable Big Data management solutions for satisfying both their functional and non-functional needs; (ii) researchers can fully comprehend the research area and identify the most interesting directions to follow. To this end, we are countering both the data placement and the storage issues focusing on the Big Data management lifecycle and Cloud computing under the prism of non-functional aspects. In contrast to previous surveys that have concentrated mainly on the Big Data storage issues in the context of functional aspects.

Research questions identification

This survey has the ambition to supply suitable and convincing answers to:

1. What are the most suitable (big) data storage technologies and how do they compete with each other according to certain criteria related to non-functional aspects?
2. What are the most suitable and sophisticated (big) data placement methods that can be followed to (optimally) place and/or migrate Big Data?

[1] https://strongboxdata.com/products/stronglink/.

[2] https://www.ibm.com/services/artificial-intelligence.

Table 1 Search query

(Big Data) AND (METHODOLOGY OR METHOD OR ALGORITHM OR APPROACH OR SURVEY OR STUDY) AND (MANAGEMENT OR PLACEMENT OR POSITION OR ALLOCATION OR STORAGE) WITH TIME SPAN:2010–2018

SLR protocol formation

It is a composite step related to the identification of (i) (data) sources—here we have primarily consulted the Web of Science and Scopus, and (ii) the actual terms for querying these (data) sources—here, we focus on population, intervention and outcome as mentioned in [34]. It is worth to note that such data sources supply nice structured searching capabilities which enabled us to better pose the respective query terms. The population mainly concerns target user groups in the research area or certain application domains. The intervention means the specific method employed to address a certain issue (used terms include: methodology, method, algorithm, approach, survey and study). Lastly, the outcome relates to the final result of the application of the respective approach (such as management, placement, positioning, allocation, storage). Based on these terms, the abstract query concretised in the context of the two data sources can be seen in Table 1.

SLR conduction

Systematic literature review conduction includes the following steps: (i) study selection criteria; (ii) quality assessment criteria; (iii) study selection procedure. All these steps are analysed in the following three paragraphs.

Study selection

The study selection was performed via a certain set of inclusion and exclusion criteria. The inclusion criteria included the following:

- Peer-reviewed articles.
- Latest articles only (last 8 years).
- In case of equivalent studies, only the one published in the highest rated journal or conference is selected to sustain only a high-quality set of articles on which the review is conducted.
- Articles which supply methodologies, methods or approaches for Big Data management.
- Articles which study or propose Big Data storage management systems or databases.
- Articles which propose Big Data placement methodologies or algorithms.

While the exclusion criteria were the following:

- Inaccessible articles.
- Articles in a different language than English.
- Short papers, posters or other kinds of small in contribution articles.
- Articles which deal with the management of data in general and do not focus on Big Data.

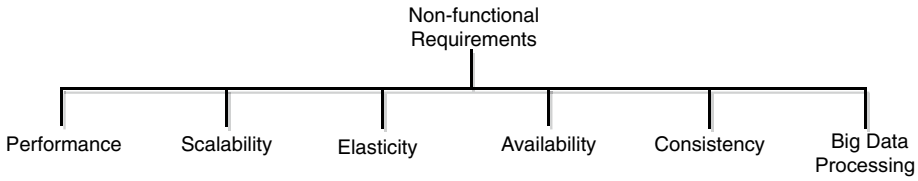


Fig. 3 Non-functional features of data management systems

- Articles that focus on studying or proposing normal database management systems.
- Articles that focus on studying or proposing normal file management systems.
- Articles that focus on the supply of Big Data processing techniques or algorithms. As the focus in this article is mainly on how to manage the data and not how to process them to achieve a certain result.

Quality assessment criteria

Apart from the above criteria, quality assessment criteria were also employed to enable prioritising the review as well as possibly excluding some articles not reaching certain quality standards. In the context of this work, the following criteria were considered:

- Presentation of the article is clear and there is no great effort needed to comprehend it.
- Any kind of validation is offered especially in the context of the proposal of certain algorithms, methods, systems or databases.
- The advancement over the state-of-the-art is clarified as well as the main limitations of the proposed work.
- The objectives of the study are well covered by the approach that is being employed.

Study selection procedure

It has been decided to employ two surveyors for each main article topic which were given a different portion of the respective reviewing work depending on their expertise. In each topic, the selection results of one author were assessed by the other one. In case of disagreement, a respective discussion was conducted. If this discussion was not having a positive outcome, the respective decision was delegated to the principal author which has been unanimously selected by all authors from the very beginning.

Non-functional data management features

For effective Big Data management, current data management systems (DMSs), including distributed file systems (DFSs) and distributed database management systems (DDBMSs) need to provide a set of non-functional features to cater the storage, management and access of the continuously growing data. This section introduces a classification of the non-functional features (see Fig. 3) of DMSs in the Big Data domain extracted from [10, 13, 35–37].

Figure 3 provides an overview of the relevant non-functional features while the following subsections attempt to analyse each of them.

Performance

Performance is typically referred to as one of the most important non-functional features. It directly relates to the execution of requests by the DMSs [38, 39]. Typical performance metrics are *throughput* and *latency*.

Scalability

Scalability focuses on the general ability to process arbitrary workloads. A definition of scalability for distributed systems in general and with respect to DDBMSs is provided by Agrawal et al. [40], where the terms scale-up, scale-down, scale-out and scale-in are defined focusing on the management of growing workloads. Vertical as well as horizontal scaling techniques are applied to distributed DBMSs and can also be applied to DFSs. Vertical scaling applies by adding more computing resources to a single node. While horizontal scaling applies by adding nodes to a cluster (or in general to the instances of a certain application component).

Elasticity

Elasticity is tightly coupled to the horizontal scaling and helps to overcome the sudden workload fluctuations by scaling the respective cluster without any downtime. Agrawal et al. [40] formally define it by focusing on DDBMSs as follows *"Elasticity, i.e. the ability to deal with load variations by adding more resources during high load or consolidating the tenants to fewer nodes when the load decreases, all in a live system without service disruption, is therefore critical for these systems"*. While elasticity has become a common feature for DDBMSs, it is still in an early stage for DFSs [41].

Availability

The availability tier builds upon the scalability and elasticity as these tiers are exploited to handle request fluctuations [42]. Availability represents the degree to which a system is operational and accessible when required. The availability of a DMS can be affected by *overloading at the DMS layer* and/or *failures at the resource layer*. During overloading, a high number of concurrent client requests overload the system such that these requests are either handled with a non-acceptable latency or not handled at all. On the other hand, a node can fail due to a resource failure (such as network outage or disk failure). An intuitive way to deal with overload is to scale-out the system. Distributed DMSs apply data replication to handle such resource failures.

Consistency

To support high availability (HA), consistency becomes an even more important and challenging non-functional feature. However, there is a trade-off among consistency, availability and partitioning guarantees, inscribed by the well-known CAP theorem [43]. This means that different kinds of consistency guarantees could be offered by a DMS. According to [44] consistency can be considered from both the client and data perspectives (i.e., from the DMS administrator perspective). The client-centric consistency can

be classified further into staleness and ordering [44]. Staleness defines the lagging of replica behind its master. It can be measured either in time or versions. Ordering defines that all requests must be executed on all replicas in the same chronological order. Data-centric consistency focuses on the synchronization processes among replicas and the internal ordering of operations.

Big Data processing

The need of native integration of (big) data processing frameworks into the DMSs arises along with the number of recently advanced Big Data processing frameworks, such as Hadoop MapReduce, Apache Spark, and their specific internal data models. Hence, the DMSs need to provide native drivers for Big Data processing frameworks which can automate the transformation of DMS data models into the respective Big Data processing framework storage models. Further, these native drivers can exploit data locality features of the DMSs as well. Please note that such a feature is also needed based on the respective DLMS architecture that has been presented in "Data lifecycle management (DLM)" section as a Big Data processing framework needs to be placed on top of the data management component.

Data storage systems

A DLMS in the Big Data domain requires both the storage and the management of heterogeneous data structures. Consequently, a sophisticated DLMS would need to support a diverse set of DMSs. DMSs can be classified into file systems for storing unstructured data and DBMSs (database management systems) for storing semi-structured and structured data. However, the variety of semi-structured and structured data requires suitable data models (see Fig. 4) to increase the flexibility of DBMSs. Following these requirements, the DBMS landscape is constantly evolving and becomes more heterogeneous.[3] The following sub-sections provides (i) an overview of related work on DBMS classifications; (ii) a holistic and up-to-date classification of current DBMS data models; (iii) a qualitative analysis of selected DBMSs; (iv) a classification and analysis of relevant DFSs.

Database management systems

The classification of the different data models (see Fig. 4) for semi-structured data has been in the focus since the last decade [37] as heterogeneous systems (such as Dynamo, Cassandra [45] and BigTable [46]) appeared on the DBMS landscape. Consequently, the term NoSQL evolved, which summarizes the heterogeneous data models for semi-structured data. Similar, the structured data model evolved with the NewSQL DBMSs [13, 47].

Several surveys have reviewed NoSQL and NewSQL data models over the last years and analyze the existing DBMS with respect to their data models and the specific non-functional features [11, 13, 35–37, 48, 49]. In addition, dedicated surveys focus explicitly specific data models (such as the time series data model [50, 51]) or specific DBMS architectures (such as in-memory DBMS [14]).

[3] http://nosql-database.org/ lists over 225 DBMS for semi-structured data.

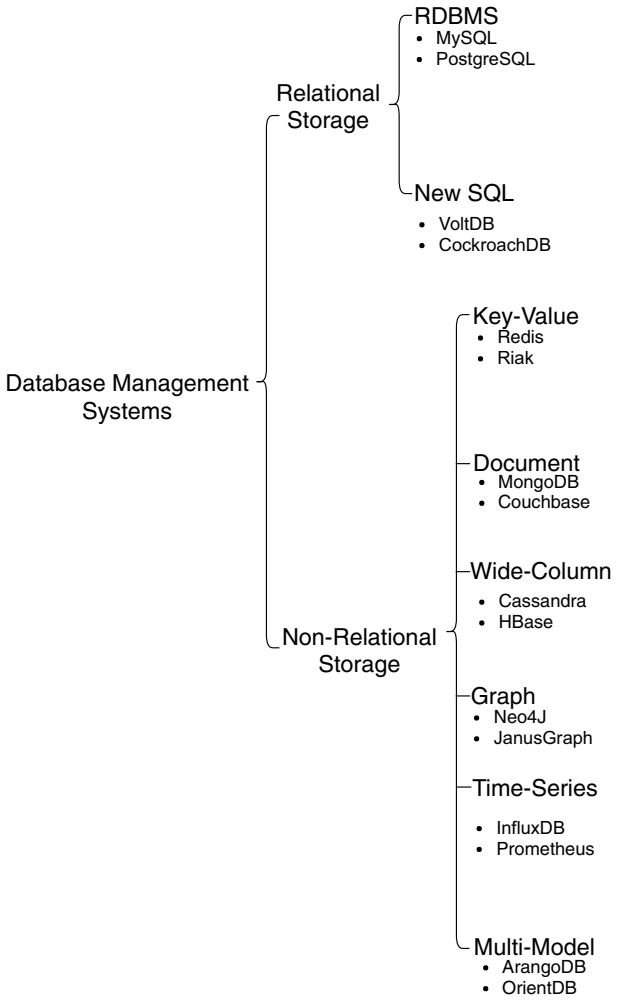


Fig. 4 DBMS data model classification

Cloud-centric challenges for operating distributed DBMS are analysed by [13], considers the following: horizontal scaling, handling elastic workload patterns and fault tolerance. It also classifies nineteen DDBMSs against features, such as partitioning, replication, consistency and security.

Recent surveys on NoSQL-based systems [35, 49] derive both, the functional and the non-functional NoSQL and NewSQL features and correlated them with distribution mechanisms (such as sharding, replication, storage management and query processing). However, the implications of Cloud resources or the challenges of Big Data applications were not considered. Another conceptual analysis of NoSQL DBMS is carried out by [48]. It outlines many storage models (such as key-value, document, column-oriented and graph-based) and also analyses current NoSQL implementations against persistence, replication, sharding, consistency and query capability. However, recent DDBMSs (such as time-series DBMSs or NewSQL DBMSs) are not analysed from Big Data as well as the Cloud context. A survey on DBMS support for Big Data with the focus on data storage models, architectures and consistency models is presented by [11]. Here, the

relevant DBMSs are analysed towards their suitability for Big Data applications, but the Cloud service models and evolving DBMSs (such as time-series databases) are also not considered.

An analysis of the challenges and opportunities for DBMSs in the Cloud is presented by [52]. Here, the relaxed consistency guarantees (for DDBMS) and heterogeneity, as well as the different level of Cloud resource failures are explained. Moreover, it is also explicated that HA mechanism is needed to overcome failures. However, the HA and horizontal scalability come with the weaker consistency model (e.g., BASE [53]) compared to ACID [43].

In the following, we distil and join existing data model classifications (refer to Fig. 4) into an up-to-date classification of the still-evolving DBMS landscape. Hereby, we select relevant details for the DLMS of Big Data applications, while we refer the interested reader to the presented surveys for an in-depth analysis of specific data models. Analogously, we apply a qualitative analysis of currently relevant DBMS based on the general DLMS features (see "Non-functional data management features" section), while in-depth analysis of specific features can be found in the presented surveys. Hereby, we select two common DBMS[4] of each data model for our analysis.

Relational data models

The relational data model stores data as tuples forming an ordered set of attributes; which can be extended to extract more meaningful information [54]. A relation forms a table and tables are defined using a static, normalised data schema. SQL is a generic data definition, manipulation and query language for relational data. Popular representative DBMSs with a relational data model are MySQL and PostgreSQL.

NewSQL

The traditional relational data model provides limited data partitioning, horizontal scalability and elasticity support. NewSQL DBMSs [55] aim at bridging this gap and build upon the relational data model and SQL. However, NewSQL relaxes relational features to enable horizontal scalability and elasticity [13]. It is worth to note that only a few NewSQL DBMSs, such as VoltDB[5] and CockroachDB,[6] are built upon such architectures with the focus on scalability and elasticity as most NewSQL DBMSs are constructed out of existing DBMSs [47].

Key-value

The key-value data model relates to the hash tables of programming languages. The data records are tuples consisting of key-value pairs. While the key uniquely identifies an entry, the value is an arbitrary chunk of data. Operations are usually limited to simple put or get operations. Popular key-value DBMSs are Riak[7] and Redis.[8]

[4] https://db-engines.com/en/ranking.

[5] https://www.voltdb.com/.

[6] https://www.cockroachlabs.com/.

[7] http://basho.com/products/riak-kv/.

[8] https://redis.io/.

Document

The document data model is similar to the key-value data model. However, it defines a structure on the values in certain formats, such as XML or JSON. These values are referred to as documents, but usually without fixed schema definitions. Compared to key-value stores, the document data model allows for more complex queries as document properties can be used for indexing and querying. MongoDB[9] and Couchbase[10] represent the common DBMSs with a document data model.

Wide-column

The column-oriented data model stores data by columns rather than by rows. It enables both storing large amounts of data in bulk and efficiently querying over very large structured data sets. A column-oriented data model does not rely on a fixed schema. It provides nestable, map-like structures for data items which improve flexibility over fixed schema [46]. The common representatives of column-oriented DBMSs are Apache Cassandra[11] and Apache HBase.[12]

Graph

The graph data model primarily uses graph structures, usually including elements like nodes and edges, for data modelling. Nodes are often used for the main data entities, while edges between nodes are used to describe relationships between entities. Querying is typically executed by traversing the graph. Typical graph-based DBMS are Neo4J[13] and JanusGraph.[14]

Time-series

The time-series data model [50] is driven by the needs of sensor storage within the Cloud and Big Data context. The time-series DBMSs are typically built upon existing non-relational data models (preferably key-value or column-oriented), and add a dedicated time-series data model on top. The data model is built upon data points which comprise a time stamp, an associated numeric value and a customisable set of metadata. Time-series DBMSs offers analytical query capabilities, which cover statistical functions and aggregations. Well-known time-series DBMSs are InfluxDB[15] and Prometheus.[16]

Multi-model

A multi-model address the problem of polyglot persistence [56] which signifies that each of the existing non-relational data models addresses a specific use case. Hence, multi-model DBMSs combine different data models into a single DBMS while build upon one storage backend to improve flexibility (e.g., providing the document and graph data

[9] https://www.mongodb.com/.
[10] https://www.couchbase.com/.
[11] http://cassandra.apache.org/.
[12] https://hbase.apache.org/.
[13] https://neo4j.com/.
[14] http://janusgraph.org/.
[15] https://www.influxdata.com/
[16] https://prometheus.io/.

model via a unified query interface). Common multi-model DBMSs are ArangoDB[17] and OrientDB.[18]

Comparison of selected DBMSs

In this section, we analyse already mentioned DBMSs in the context of Big Data applications (see Table 2). To perform this, we first analyse already mentioned DBMS (of the previously introduced data models) with respect to their features and supported Cloud service models. Next, we provide a qualitative analysis with respect to the non-functional features of the DMSs (refer to "Non-functional data management features" section). For quantitative analysis of these non-functional requirements, we refer the interested reader to the existing work focused on DBMS evaluation frameworks [44, 57–60] and evaluation results [42, 61, 62].

Qualitative criteria

In the Table 2, the first three columns present each DBMS and its data model, followed by the technical features and the service models supported. The analysis only considers the standard version of a DBMS.

In the following, we attempt to explicate each of the technical features considered. The DBMS *architecture* is classified into single, master–slave and multi-master architectures [56]. The *sharding* strategies are analysed based on the DBMS architectures; they can be supported manually as well as automatically in a hash- or range-based manner. The *elasticity* feature relies on a distributed architecture and relates to whether a DBMS supports adding and/or removing nodes from the cluster at runtime without a downtime. For consistency and availability guarantees, each DBMS is analysed with respect to its consistency (C), availability (A) and partition tolerance (P) properties within the *CAP* theorem (i.e., CA, CP, AC or AP) [43]. However, it should be highlighted that we did not consider fine-grained configuration options that might be offered for a DBMS to vary the CAP properties. Next, the *replication* mechanisms are analysed in terms of both cluster and cross-cluster replication (also known as geo-distribution). Consequently, a DBMS supporting cross-cluster replication implicitly supports cluster replication. The interested reader might consider [63] for more fine-grained analysis of replication mechanisms of DDBMSs. The *Big Data adapter* is analysed by evaluating native and/or third-party drivers for Big Data processing frameworks. Finally, the DDBMSs are classified based on their offering as *community* editions, *enterprise* commercial editions or managed *DBaaS*. One exemplary provider is presented if the DBMS is offered as a DBaaS.

Qualitative analysis

The resulting Table 2 represents the evolving landscape of the DBMSs. The implemented features of existing DBMSs significantly differ (except the RDBMSs) even within one data model. The heterogeneity of analysed DBMSs is even more obvious across data models. Further, the heterogeneous DBMS landscape offers a variety of potential DBMS solutions for Big Data.

[17] https://www.arangodb.com/.

[18] https://orientdb.com/.

Table 2 Technical feature analysis of selected DBMS

DBMS	Version	Data model	Technical features						Service model		
			Architecture	Sharding	Elasticity	CAP	Replication	Big Data adapter	Community	Enterprise	DBaaS
MySQL	8.0.11	RDBMS	Single/master–slave	Manual	No	CA	Cluster	3rd party (SQL-based)	Yes	Yes	https://Cloud.oracle.com/ mysql
PostgreSQL	10.4	RDBMS	Single/master–slave	Manual	No	CA	Cluster	3rd party (SQL-based)	Yes	Yes	https://aws.amazon.com/rds/postgresql/
VoltDB	8.1.2	NewSQL	Multi-master	Hash	Yes (commercial)	CP	Cross-cluster (commercial)	3rd party (SQL-based)	Yes	Yes	No
CockroachDB	2.0.3	NewSQL	Multi-master	Hash	Yes	CP	Cross-cluster (commercial)	3rd party (SQL-based)	Yes	Yes	No
Riak	2.2.3	Key-value	Multi-master	Hash	Yes	AP	Cross-cluster	Native	Yes	Yes	No
Redis	4.0	Key-value	Multi-master	Hash	Yes	AC	Cluster	Native	Yes	Yes	https://redislabs.com/
MongoDB	4.0.0	Document	Multi-master	Hash/range	Yes	CP	Cross-cluster	Native	Yes	Yes	https://www.mongodb.com/Cloud/atlas
Couchbase	5.0.1	Document	Multi-master	Hash	Yes	CP	Cross-cluster	Native	Yes	Yes	https://www.couchbase.com/products/Cloud/managed-Cloud
Cassandra	3.11.2	Wide-column	Multi-master	Hash/range	Yes	AP	Cross-cluster	Native	Yes	Yes, By DataStax	https://www.instaclustr.com/solutions/managed-apache-cassandra/
HBase	2.0.1	Wide-column	Multi-master	Hash	Yes	CP	Cross-cluster	3rd party	Yes	Yes, By Cloudera	No
Neo4J	3.4.1	Graph	Master–slave	No	Yes	CA	Cross-cluster	Native	Yes	Yes	https://www.graphstory.com/
JanusGraph	0.2.0	Graph	Multi-master	Manual	Yes	AP/CP	Cluster	3rd party	Yes	No	No

Table 2 (continued)

DBMS	Version	Data model	Technical features						Service model		
			Architecture	Sharding	Elasticity	CAP	Replication	Big Data adapter	Community	Enterprise	DBaaS
ArangoDB	3.3.11	Multi-model (key-value, document, graph)	Multi-master	Hash	Yes	CP	Cross-cluster	Native	Yes	Yes	No
OrientDB	3.0.2	Multi-model (key-value, document, graph)	Multi-master	Hash	Yes	–	Cross-cluster (commercial)	Native	Yes	Yes	No
InfluxDB	1.5.4	Time-series	Multi-master (commercial)	Range	Yes (commercial)	AP/CP	Cross-cluster (commercial)	3rd party	Yes	Yes	https://cloud.influxdata.com/
Prometheus	2.3	Time-series	Master–slave	Manual	No	–	Cluster	3rd party	Yes	Yes	No

The feature analysis provides a baseline for the qualitative analysis of the non-functional features. From the (horizontal) scalability point-of-view, a DBMS with a multi-master architecture is supposed to provide scalability for write and read workloads, while a master–slave architecture is supposed to provide read scalability. Due to the differences between the DBMSs, the impact of elasticity requires additional qualitative evaluations [42].

The consistency guarantees correlate to the classification in the CAP theorem. Table 2 clearly shows the heterogeneity compared to the consistency guarantees. Generally, the single-master or master–slave architectures provide strong consistency guarantees. Multi-master architectures cannot be exactly classified into the CAP theorem as their consistency guarantees heavily depend on the DBMS runtime configuration [64]. Additional evaluations of the consistency for the selected DBMSs are required for strong consistency (so as to ensure scalability, elasticity and availability) [44, 62].

Providing HA directly relates to the supported replication mechanisms to overcome failures. The analysis shows that all DBMSs support cluster-level replication, while cross-cluster replication is supported by ten out of the sixteen DBMSs. Big Data processing relates to the technical feature of Big Data adapters. Table 2 clearly shows that seven DBMSs provide native adapters and nine DBMS enable it via third-party adapters to support Big Data processing. The service model of all the DBMSs is either available as a self-hosted community or enterprise version. In addition, both RDBMS and six NoSQL DBMS are offered as managed DBaaS. While the DBaaS offerings are abstracting all operational aspects of the DBMS, an additional analysis might be required with respect to their non-functional features and cost models [65].

Cloudification of DMS

Traditional on-premise DBMS offerings are still popular, but the current trend shows that DDBMSs running in the Cloud are also well-accepted. Especially, as Big Data imposes new challenges such as scalability, the diversity of data management or the usage of Cloud resources, towards the massive storage of data [66]. In general, the distributed architecture of DMSs evolved their focus over exploiting the Cloud features and catering the 5Vs of Big Data [67]. Data-as-a-service (DaaS) mostly handles the data aggregation and management via appropriate web-services, such as RESTful APIs. While database-as-a-service (DBaaS) offers database as a service which can include (distributed) a relational database or a non-relational one. In most of the cases, storage-as-a-service (STaaS) includes both DaaS and the DBaaS [68]. Furthermore, BDaaS [3] is a Cloud service (such as Hadoop-as-a-service) where traditional applications are migrated from local installations to the Cloud. BDaaS wraps three primary services. They are (i) IaaS (for underlying resources), (ii) STaaS (a sub-domain of platform-as-s-service (PaaS)) for managing the data via dynamic scaling and (iii) data management (such as data placement, replica management).

Distributed file systems

A distributed file systems (DFS) is an extended networked file system that allows multiple distributed nodes to internally share data/files without using remote call methods or procedures [69]. A DFS offers scalability, fault-tolerance, concurrent file access and

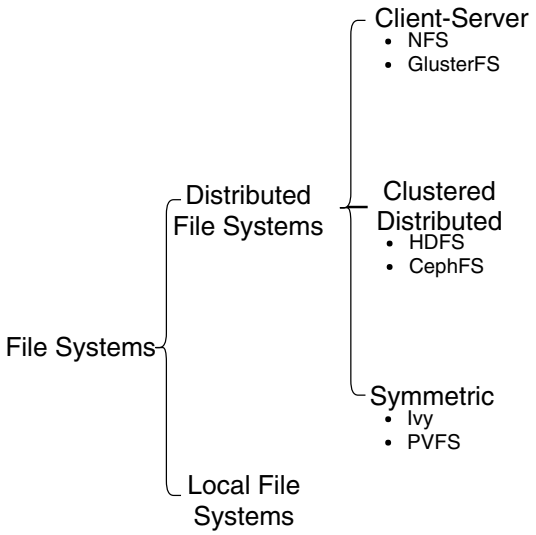


Fig. 5 Distributed file systems classification

metadata support. However, the design challenges (independent of data size and storage type) of a DFS are transparency, reliability, performance, scalability, and security. In general, DFSs do not share storage access at the block level but rather work at the network level. In DFSs, security relies on either access control lists (ACLs) or respectively defined capabilities, depending on how the network is designed. DFSs can be broadly classified into three models and respective groups (see Fig. 5). First, client–server architecture based file systems which supply a standardized view of a local file system. Second, clustered-distributed file systems which offer multiple nodes to enable concurrent access to the same block device. Third, symmetric file systems, where all nodes have a complete view of the disk structure. Below, we briefly analyse each category in a separate sub-section while we also supply some strictly open-source members for it.

Client–server model

In the client–server architecture based file system, all communications between servers and clients are conducted via remote procedure calls. The clients maintain the status of current operations on a remote file system. Each file server provides a standardized view of its local file system. Here, the file read-operations are not mutually exclusive but the write operations are. File sharing is based on mounting operations. Only the servers can mount directories exported from other servers. Network File System (NFS) and GlusterFS[19] are two popular open source implementations of the client–server model.

Clustered-distributed model

Clustered-distributed based systems organize the clusters in an application-specific manner and are ideal for DCs. The model supports a huge amount of data; the data is stored/partitioned across several servers for parallel access. By design, this DFS model

[19] https://docs.gluster.org/en/latest/.

Table 3 Feature analysis of selected DFSs

DFS	Version	FileSystem	Technical features					
			Architecture	Sharding	Elasticity	CAP	Replication	Big Data adapter
NFS	4.2	Client–server	Fully-centralized	Index/range	No	CA	Block level	3rd party
GlusterFS	4.0	Client–server	Fully-centralized	Automatic	Yes	CA	Node level	Native
HDFS	3.0.1	Clustered-distributed	Less-centralized	Fixed size	Yes	AP	Block level	Native
CephFS	12.2.5	Clustered-distributed	Less-centralized	Index/range	Yes	CP	Cluster-level	Native/3rd party
Ivy	0.3	Symmetric	Fully-distributed	DHash	Yes	AP	Block-level	–
PVFS	2.0	Symmetric	Fully-distributed	Hash	Yes	AP	–	3rd party

is fault tolerant as it enables the hosting of a number of replicas. Due to the huge volume of data, data are appended instead of overwritten. In general, the DNS servers map (commonly using round-robin fashion) access requests to the clusters for load-balancing purposes. The Hadoop distributed file system (HDFS) and CephFS[20] are two popular implementations of such a DFS model.

Symmetric model

Symmetric is a DFS that supports a masterless architecture, where each node has the same set of roles. It mainly resembles a peer-to-peer system. In general, the symmetric model employs a distributed hash table approach for data distribution and replication across systems. Such a model offers higher availability but reduced performance. Ivy [70] and the parallel virtual file system (PVFS) [71] are examples of a symmetric DFS model.

DFS evaluation

Similar to DDBMSs, we also compare the open source implementations of DFSs according to the same set of technical features or criteria. A summary of this comparison is depicted in Table 3. In general, most of the file systems are distributed in nature (except NFS and GlusterFS). However, they do exhibit some architectural differences. NFS and GlusterFS are both developed focusing on a master–slave approach, while Ivy and PVFS are based on the masterless model. Data partitioning (or sharding) is also supported dynamically (featured by Ivy and PVFS) or statically via a fixed size (as in case of HDFS) by these DFSs. Elasticity or supporting the data scaling is a very important feature for many Big Data applications (especially hosted at Cloud). We can thus observe that except NFS all mentioned DFSs support scalability. Further, HDFS, CephFS, Ivy and PVFS are fault tolerant as well. Replication, highly needed for not losing data, is well supported by all DFSs. However, their granularity differs from the block to the cluster level. Finally, these DFSs also offer some form of hooks (either native or third-party supplied) to be used with Big Data frameworks.

[20] http://docs.ceph.com/docs/mimic/cephfs/.

Data placement techniques

In the Cloud ecosystem, traditional placement algorithms incur a high cost (including the time) on storing and transferring data [72]. Placing data while data is partitioned and distributed across multiple locations is a challenge [23, 73]. Runtime data migration is an expensive affair [30–32] and the complexity increases due to the frequent change of applications as well as DCs' behaviour (i.e., resources or latencies) [74]. Placing a large amount of data across the Cloud is complex due to issues, such as (i) data storage and transfer cost optimisation while maintaining data dependencies; (ii) data availability and replication; (iii) privacy policies, such as restricted data storage based on geo-locations. Data replication can influence consistency, while it also enhances the scalability and higher availability of data. In general, the existing data placement strategies can be grouped based on user-imposed constraints, such as data access latency [75], fault tolerance [76], energy-cost awareness [77], data dependency [78, 79] and robustness or reliability [80, 81].

Formal definition

The data placement in a distributed computing domain is an instance of NP-hard problem [82], while it can be reduced to a bin-packing problem instance. Informally, the data placement problem can be described as follows: given a certain workflow, the current data placement, and a particular infrastructure, find the right position(s) of data within the infrastructure to optimise one or more certain criteria, such as the cost of the data transfer.

A formal representation of this problem as follows: suppose that there are N datasets, represented as d_i (where $i = 1, \ldots, N$). Each dataset has a certain size s_i. Further, suppose that there are M computational elements represented as V_j (where $j = 1, \ldots, M$). Each computational element has a certain storage capacity denoted as c_j. Finally, suppose that there is a workflow W with T tasks which are represented as t_k (where $k = 1, \ldots, T$). Each task has a certain input $t_k.input$ and output $t_k.output$, where each maps to a set of datasets.

The main set of decision variables is c_{ij} representing the decisions (e.g., based on privacy or legal issues) of whether a certain dataset i should be stored in a certain computational element j. Thus, there is a need to have $c_{ij} == 1$ for each i and a certain j. Two hard constraints need to hold: (i) a dataset should be stored in one computational element which can be represented as follows: $\sum_j c_{ij} = 1$ for each i. It is worth to note that this constraint holds when no dataset replication is allowed. Otherwise, it would take the following form: $\sum_j c_{ij} >= r$, where r is the replication factor; (ii) the capacity of a computational element should be sufficient for hosting the respective dataset(s) assigned to it. This is represented as follows: $\sum_i c_{ij} * s_i <= c_j$ for each j.

Finally, suppose that the primary aim is to reduce the total amount of data transfers for the whole workflow. In this respect, this optimisation objective can be expressed as follows:

$$\text{minimise} \sum_{k} \sum_{d_i, d_{i'} \in t_k.input} \left(\text{m}(d_i) <> \text{m}\left(d_{i'}\right)\right) \tag{1}$$

where $\text{m}(d_i)$ (which supplies as the output a value in $[1, M]$) indicates the index of the computational element that has been selected for a certain dataset. This objective adds the amount of data transfers per each workflow task which relates to the fact that the task will be certainly placed in a specific resource mapping to one of the required input

datasets. Thus, during its execution, the data mapping to the rest of the input datasets will need to be moved in order to support the respective computation needed.

Data placement methodologies

We broadly classify the proposed data placement methods into data dependency, holistic task and data scheduling and graph-based methods. The methods in each category are analysed in the following subsections.

Data dependency methods

A data-group-aware placement scheme is proposed in [83] by employing the bond energy algorithm (BEA) [84] to transform the original data dependency matrix into a Hadoop cluster. It exploits access patterns to find an optimal data grouping to achieve better parallelism and workload balancing. In [85], a data placement algorithm is proposed for solving the data inter-dependency issue at the VM level. Scalia [86] proposes a Cloud storage brokerage scheme that optimises the storage cost by exploiting the real-time data access patterns. Zhao et al. [87] proposed data placement strategies for both initial data placement and relocation using a particle swarm optimization (PSO) algorithm. For fixed data set placement, this method relies on hierarchical data correlation and performs data re-allocation during the storage saturation. Yuan et al. [78] propose a k-means based dataset clustering algorithm to construct a data dependency matrix by exploiting the data dependency and the locality of computation. Later, the dependency matrix is transformed by applying the BEA while items are clustered based on their dependencies by following a recursive binary partitioning algorithm. In general, the preservation of time locality can significantly impact caching performance while the efficient re-ordering of jobs can improve the resource usage. In [79] authors propose a file grouping policy for pre-staging data by preserving time locality and enforcing the role of job re-ordering via extracting access patterns.

Task and data scheduling methods

In [88], the authors propose an adaptive (based on multi-objective optimization model) data management middleware which collects system-state information and abstracts away the complexities of multiple Cloud storage systems. For internet-of-things (IoT) data streaming support, Lan et al. [89] proposed a data stream partitioning mechanism by exploiting statistical feature extraction. Zhang et al. [90] propose a mixed-integer linear programming model for modelling the data placement problem. It considers both the data access cost as well as the storage limitations of DCs. Hsu et al. [91] proposed a Hadoop extension by adding dynamic data re-distribution (by VM profiling) before the map phase and VM mapping for reducers based on partition size and VM availability. Here, high capacity VMs are assigned for high workload reducers. Xu et al. [92] proposes a genetic programming approach to optimise the overall number of data transfers. However, this approach does not consider the DCs' capacity constraints and the non-replication constraints of data sets. In [93], a policy engine is proposed for managing both the number of parallel streams (between origin and destination nodes) and the priorities for data staging jobs in scientific workflows. The policy engine also considers data transfers, storage allocation and network resources.

The storage resource broker [23] provides seamless access to the different distributed data sources (interfacing multiple storages) via its APIs. It works as a middleware between the multiple distributed data storages and applications. BitDew [94] offers a programmable environment for data management via metadata exploitation. The data scheduling (DS) service takes care of implicit data movement. Pegasus [95] provides a framework that maps complex scientific applications onto distributed resources. It stores the newly generated data and also registers them in the metadata catalogue. The replica location service [96] is a distributed, scalable, data management service that maps the logical data names to target names. It supports both centralized as well as distributed resource mapping. Kosar and Livny [81] proposes a data placement that consists of a scheduler, a planner and a resource broker. The resource broker is responsible for matching resources, data identification and decisions related to data movement. The scheduling of data placement jobs relies on the information given by the workflow manager, the resource broker and the data miner. A very interesting feature of the proposed sub-system is that it is able to support failure recovery through the application of retry semantics.

Graph-based data placement

Yu and Pan [72] proposes the use of *sketches* to construct a hyper-graph sparsifier of data traffic to lower the data placement cost. Such sketches represent data structures that approximate properties of a data stream. LeBeane et al. [97] proposed on-line graph-partitioning multiple strategies to optimise data-ingress across heterogeneous clusters. SWORD [98] handles the partitioning and placement for OLTP workloads. Here, the workload is represented as a hypergraph and a hyper-graph compression technique is employed to reduce the data partitioning overhead. An incremental data re-partitioning technique is also proposed that modifies data placement in multiple steps to support workload changes. Kayyoor et al. [99] propose how to map nodes to a subset of clusters via satisfying user constraints. It minimises the query span for query workloads by applying replica selection and data placement algorithms. The query-based workload is represented as hyper-graphs and a hypergraph partitioning algorithm is used to process them. Kaya et al. [100] model the workflow as a hypergraph and employ a partitioning algorithm to reduce the computational and storage load while trying to minimise the total amount of file transfers.

Comparative evaluation

In this section, we have carefully selected a set of criteria to evaluate the methods analysed in "Data placement methodologies" section. The curated criteria are: (i) *fixed data sets*—whether the placement of data can be a priori fixed in sight of, e.g., regulations, (ii) *constraint satisfaction*—which constraint solving technique is used, (iii) *granularity*—what is the granularity of the resources considered, (iv) *intermediate data handling*—whether intermediate data, produced by, e.g., a running workflow, can be also handled, (v) *multiple application handling*—whether the data placement over multiple applications can be supported, (vi) *increasing data size*—whether the growth rate of data is taken into account, (vii) *replication*—whether data replication is supported, (viii) *optimisation criteria*—which optimisation criteria are exploited, (ix) *additional system related*

Table 4 Comparative summary of existing data placement algorithms

Approach	Fixed DS	Constraint satisfaction	Granul.	Interm. DS	Mult. appl.	Data size	Repl.	Opt. criteria	Add. info.
BDAP [85]	Yes	Meta-heuristic	Fine	Yes	No	No	No	Comm. cost	No
Xu [92]	No	Meta-heuristic	Coarse	No	No	No	No	Data transf. number	No
Yuan [78]	Yes	Recursive binary part.	Coarse	Yes	Yes	Yes	No	Data transf. number	No
Kaya [100]	No	Hypergraph part.	Coarse	No	No	No	No	Data transf. number	No
Zhao [87]	Yes	Hierarchical part. clust. + PSO	Fine	Yes	No	No	No	Data transf. number	No
Wang [83]	No	Recursive clust. + ODPA	Fine	No	No	No	No	Data transf. number	Yes
Yu [72]	No	Hypergraph part.	Fine	No	No	No	No	Cut weight	Yes
Zhang [90]	No	Lagrance MIP relaxation	Coarse	No	No	No	No	Data access cost	No
Hsu [91]	No	–	Fine	No	No	No	No	Profiling-related metric	Yes
LeBeane [97]	No	Hypergraph part.	Fine	No	No	No	No	Skew factor	Yes
Lan [89]	No	Clustering-based PSO search	Fine	No	No	No	No	Volatility AMA, hurst distance	Yes
BitDew [94]	No		Fine	Yes	Yes	No	Yes	Data dep. repl., fault tol.	Yes
Kayoor [99]	No	Hypergraph part.	Coarse	No	No	No	Yes	Avg. query span	Yes
Kosar [81]	Yes		Fine	Yes	Yes	No	Yes		Yes
Scalia [86]	No	Multi-dimensional Knapsack problem	Fine	No	Yes	Yes	No	Storage cost	Yes
SWORD [98]	Yes	Graph partition	Fine		No		Yes	Conflicting transactions	Yes

information—whether additional knowledge is captured which could enable the production of better data placement solutions. An overview of the evaluation based on these criteria can be observed in comparison Table 4. First of all, we can clearly see that there is no approach that covers all the criteria considered. Three approaches (Yuan et al. [78], BitDew [94] and Kosar [81]) can be distinguished, considered also as complementary to each other. However, only in [78] a suitable optimisation/scheduling algorithm for data placement has been realised.

Considering now each criterion in isolation, we can observe in Table 4 that very few approaches consider the existence of a fixed or semi-fixed location of data sets. Further, such approaches seem to prescribe a fixed a-priori solution to the data placement problem which can lead to a sub-optimal solution. Especially as optimisation opportunities are lost in sight of more flexible semi-fixed location constraints. For instance, fixing the placement of a dataset to a certain DC might be sub-optimal in case that multiple DCs in the same location exist.

Three main classes of data placement optimisation techniques can be observed: (i) meta-search (like PSO)/genetic programming) to more flexibly inspect the available solution space and efficiently find a near-optimal solution; (ii) hierarchical partition algorithms (based on BEA) that attempt to group data recursively based on data dependencies either to reduce the number or the cost of data transfers. BEA is used as the baseline for many of these algorithms. BEA also supports dynamicity. In particular, new data sets are handled by initially encoding them in a reduced table-based form before applying the BEA. After the initial solution is found, the modification can be done by adding cluster/VM capacity constraints into the model. (iii) a Big Data placement problem can also be encoded via a hypergraph. Here, nodes are data and machines while hyper-edges attempt to connect them together. Through such modelling, traditional or extended hypergraph partitioning techniques can be applied to find the best possible partitions. There can be a trade-off between different parameters or metrics that should be explored by all the data placement algorithms irrespectively of the constraint solving technique used. However, such a trade-off is not usually explored as in most cases only one metric is employed for optimisation.

Granularity constitutes the criterion with less versatility as most of the approaches have selected a fine-grained approach for data-to-resource mapping, which is suitable for the Cloud ecosystem.

The real-world applications are dynamic and can have varying load at different points of time. Furthermore, applications can produce additional data which can be used for next computation steps. Thus, data placement should be a continuous process to validate decisions taken at different points in time. However, most approaches in data placement, focus mainly on the initial positioning of Big Data and do not interfere with the actual runtime of the applications.

There seems also to exist a dependency between this criterion and the fixed data sets one. The majority of the proposed approaches satisfying this criterion also satisfy the fixed data set one. This looks like a logical outcome as dynamicity is highly correlated to the need to better handle some inherent data characteristics. Further, a large volume of intermediate data can also have a certain gravity effect that could resemble the one concerning fixed data.

The multi-application criterion is not supported at all. This can be due to the following facts: (i) multi-application support can increase the complexity and the size of the problem; (ii) it can also impact the solution quality and solution time which can be undesirable especially for approaches that already supply sub-optimal solutions.

Only the approach in [78] caters for data growth via reserving additional space in already allocated nodes based on statically specified margins. However, such an approach is static in nature and faces two unmet challenges: the support for dynamic data growth

monitoring, suitable especially in cases where data can grow fast, and dynamic storage capacity determination, through, e.g., data growth prediction, for better supporting proactive data allocation. However, if we consider all dynamicity criteria together, we can nominate the approach in [78] as the one with the highest level of dynamicity, which is another indication of why it can be considered as prominent.

Data replication has been widely researched in the context of distributed systems but has not been extensively employed in data placement. Thus, we do believe that there exists a research gap here. Especially as those few approaches (such as SWORD [98], Kosar [81], Kayoor [99], BitDew [94]) that do support replication still lack suitable details or rely on very simple policies driven by user input.

We can observe that the minimisation of data transfer number or cost is a well-accepted optimisation criterion. Furthermore, data partitioning related criteria, such as *skew factor* and *cut weight*, have been mostly employed in the hypergraphs based methods. In some cases, we can also see multiple criteria to be considered which are: (i) either reduced to an overall one; (ii) not handled through any kind of optimisation but just considered in terms of policies that should be enforced. In overall, we are not impressed by the performance of the state-of-the-art in this comparison criterion. So, there is a huge room for potential improvement here.

Finally, many of the methods also consider additional input to achieve a better solution. The most common extra information that is exploited is data access patterns and nodes (VMs or PMs) profiling to, e.g., inspect their (data) processing speed. However, while both are important, usually only one from these two is exploited in these methods.

Lessons learned and future research directions

To conclude our survey, in this section we will discuss the issues of the current state-of-the-art and the research gaps or opportunities related to data storage and placement. Further, we also supply research directions towards a complete DLMS system in the Big Data-Cloud ecosystem.

Data lifecycle management

Challenges and issues

This subsection refers to how the discussed data storage and placement challenges can be combined and viewed from the perspective of a holistic DLMS of the future. Such a DLMS should be able to cope with the optimal data storage and placement in a way that considers the Big Data processing required, along with the functional and non-functional variability space of the given Cloud resources at hand, in each application scenario. It implies the ability to consider both private and public Clouds, offered by one or several Cloud vendors, according to the specifics of each use cases, while making the appropriate decisions on how the data should be stored, placed, processed and eventually managed.

Just considering the cross-Cloud application deployment for fully exploiting the benefits of the Cloud paradigm hinders the important challenge of data-awareness. This data-awareness refers to the need to support an application deployment process that considers the locations of data sources, their volume and velocity characteristics, as well as any security and privacy constraints applicable. Of course, from the DLM perspective,

this means that there should also be a consideration of the dependencies between application components and all data sources. This has the reasonable implication that the components requiring frequent accesses to data artefacts, found at rest in certain data stores, cannot be placed in a different Cloud or even in a certain physical and network distance from the actual storage location. If such aspects are ignored then application performance certainly degrades, as expensive data migrations may incur while legislation conformance issues might be applicable.

Future research directions

Among the most prominent research directions, we highlighted the design and implementation of a holistic DLMS, able to cope with all of the above-mentioned aspects on the data management, while employing the appropriate strategies for benefiting from the multi-Cloud paradigm. It is important to note that data placement in virtualized resources is generally subjected to long-term decisions as any potential data migrations generally incur immense costs which may be amplified by data gravity aspects that may result in subsequent changes in the application placement. Based on this, we consider the following aspects that should sketch the main functionality of the DLMS of the future that is able to cope with Big Data management and processing by really taking advantage of the abundance of resources in the Cloud computing world:

- Use of advanced modelling techniques that consider metadata schemas for setting the scope of truly exploitable data modelling artefacts. It refers to managing the modelling task in a way that covers the description of all V's (e.g. velocity, volume, value, variety, and veracity) in the characteristics of Big Data to be processed. The proper and multi-dimensional data modelling will allow for an adequate description of the data placement problem.
- Perform optimal data placement across multiple Cloud resources based on the data modelling and user-defined goals, requirements and constraints.
- Use of efficiently distributed monitoring functionalities for observing the status of the Big Data stored or processed and detect any migration or reconfiguration opportunities.
- Employ the appropriate replication, fail-over and backup techniques by considering and exploiting at the same time the already offered functionalities by public Cloud providers.
- According to such opportunities, continuously make reconfiguration and migration decisions by consistently considering the real penalty for the overall application reconfiguration, always in sight of the user constraints, goals and requirements that should drive the configuration of computational resources and the scheduling of application tasks.
- Design and implement security policies in order to guarantee that certain regulations (e.g., General Data Protection Regulation) are constantly and firmly respected (e.g., data artefacts should not be stored or processed outside the European Union) while at the same time the available Cloud providers' offerings are exploited according to the data owners' privacy needs (e.g., exploit the data sanitization service when migrating or just removing data from a certain Cloud provider).

Data storage

In this section, we highlight the challenges for holistic data lifecycle management with respect to both the current DBMS and DFS systems and propose future research directions to overcome such challenges.

Challenges and issues

In the recent decade, the DBMS landscape has significantly evolved with respect to the data models and supported non-functional features, driven by Big Data and the related requirements of Big Data applications (see "Non-functional data management features" section). The resulting heterogeneous DBMS landscape provides a lot of new opportunities for Big Data management while it simultaneously imposes new challenges as well. The variety of data models offers domain-specific solutions for different kinds of data structures. Yet, the vast number of existing DBMSs per data model leads to a complex DBMS selection process. Hereby, functional features of potential DBMSs need to be carefully evaluated (e.g., NoSQL DBMSs do not offer a common query interface even within the same data model). For the non-functional features, the decision process is twofold: (i) a qualitative analysis (as carried out in "Comparison of selected DBMSs" section) should be conducted to narrow down the potential DBMSs; (ii) quantitative evaluations should be performed over the major non-functional features based on existing evaluation frameworks.

While collecting data from many distributed and diverse data sources is a challenge [8] modern Big Data applications are typically built upon multiple different data structures. Consequently, current DBMSs cater for domain-specific data structures due to the variety of data models supported, (as shown in our analysis Table 2). However, exploiting the variety of data models typically leads to the integration of multiple different DBMSs in modern Big Data applications. Consequently, the operation of a DBMS needs to be abstracted to ease the integration of different DBMSs into Big Data applications and to fully exploit the required features (such as scalability or elasticity). Hereby, research approaches in Cloud-based application orchestration can be exploited [101, 102]. While the current DBMS landscape already moves towards the Big Data domain, the optimal operation of large-scale or even geo-distributed DBMSs still remains a challenge as the non-functional features significantly differ for different DBMSs (especially by using Cloud resources [42, 61, 103]).

In general, DFS provides scalability, network transparency, fault tolerance, concurrent data (I/O) access, and data protection [104]. It is worth noting that in Big Data domain, the scalability must be achieved without increasing the degree of replication of stored data (particularly for the Cloud ecosystem while combined with the private/local data storage systems). The storage system must increase user data availability but not the overheads. While resource sharing is a complex task and the severity can increase manyfolds while managing the Big Data. In today's Cloud ecosystem, we lack a single/unified model that offers a single interface to connect multiple Cloud-based storage models (such as Amazon S3 objects) and DFSs. Apart from that, the synchronization in DFS is also a well-known issue and as the degree of data access concurrency is increasing, synchronization could certainly be a performance bottleneck. Moreover, in some cases,

it has also been observed that the performance of DFSs is low compared to the local file systems [105, 106]. Furthermore, network transparency is also a crucial process related to the performance, especially while handling Big Data (because now the Big Data is distributed across multiple Clouds). Although most DFSs uses transmission control protocol or user datagram protocol during the communication process, however, a smarter way needs to be devised. In DFS, the fault-tolerance is achieved by lineage, checkpoint, and replicating metadata (and data objects) [104]. While the state-less based DFSs are having fewer overheads regarding managing the file states while reconnecting after failures, the state-full approach is also in use. For DFSs, the failure must be handled very fast and seamlessly across the Big Data management infrastructure. On the other side, there is no well-accepted approach to data access optimization methods. The methods such as data locality, multi-level caches are used case by case. Finally, securing the data in the DFS-Cloud ecosystem is a challenge due to the interconnection of so many diverse hardware as well as software components.

Future research directions

To address the identified challenges for the data storage in Big Data lifecycle management, novel Big Data-centric evaluations are required that ease the selection and operation of large-scale DBMS.

- The growing domain of hybrid transaction/analytical processing workloads needs to be considered for the existing data models. Moreover, comparable benchmarks for different data models need to be established [107] and qualitative evaluations need to be performed across all data model domains as well.
- To select an optimal combination of a distributed DBMS and Cloud resources, evaluation frameworks across different DBMS, Cloud resource and workload domains are required [108]. Such frameworks ease the DBMS selection and operation for Big Data lifecycle management.
- Holistic DBMS evaluation frameworks are required to enable the qualitative analysis across all non-functional features in a comparable manner. In order to achieve this, frameworks need to support complex DBMS adaptation scenarios, including scaling and failure injection.
- DBMS adaptation strategies need to be derived and integrated into the orchestration frameworks to enable the automated operation (to cope with workload fluctuations) of a distributed DBMS.
- Qualitative DBMS selection guidelines need to be extended with respect to operational and adaptation features of current DBMS (i.e., support for orchestration frameworks to enable automated operation and adaptation and the integration support into Big Data frameworks).

Similar to the above research directions for DBMSs, we also mention below the research directions for DFSs.

- For efficient, resource sharing among multiple Cloud service providers/components, a single/unified interface must handle the complex issues, such as seamless

workload distribution, improved data access experience and faster read-write synchronizations, together with the increased level of data serialization for DFSs.
- We also advocate for using smarter replica-assignment policies to achieve better workload balance and efficient storage space management.
- To counter the synchronization issue in DFSs, a generic solution could be to cache the data in the client or in the local server's side, but such an approach can become the bottleneck for the Big Data management scenario as well. Thus, exploratory research must be done in this direction.
- As the data diversity and the networks heterogeneity is increasing, an abstract communication layer must be in place to address the issue of network transparency. Such abstraction can handle different types of communications easily and efficiently.
- The standard security mechanisms are in place (such as ACLs) for data security. However, after the Cloudification of the file system, the data become more vulnerable due to the interconnection of diverse distributed, heterogeneous computing components. Thus, proper security measures must be built-in features of tomorrow's DFSs.

Data placement

The following data placement challenges and corresponding research directions are in line with our analysis in "Comparative evaluation" section.

Challenges and issues

Fixed data set size We have observed data placement methods able to fix the location of data sets based on respective (privacy) regulations, laws or user requirements. Such requirements indicate that data placement should be restrained within a certain country, sets of countries or even continents. However, this kind of semi-fixed constraints is handled in a rather static way by already pre-selecting the right place for such data sets.

Constraint solving Exhaustive solution techniques are efficient to reach optimal solutions but suffer from scalability issues and higher execution time (especially for medium/big-sized problem instances). On the other hand, meta-heuristics (such as PSO) seems more promising as they can produce near-optimal solutions faster by also achieving better scalability. However, they need proper configuration and modelling which can be a time-consuming task while it is not always guaranteed that near-optimal solutions can be produced.

Granularity Most of the evaluated methods support a fine-grained approach for dataset placement. However, all such methods consider that resources are fixed in number. Such assumptions are inflexible in the sight of the following issues: (i) a gradual data growth can saturate the resources assigned to data. In fact, a whole private storage infrastructure could be saturated for this reason; (ii) data should be flexibly (re-)partitioned to tackle the workload variability.

Multiple applications Only three from the evaluated methods (see Table 4) can handle multiple applications but also in a very limited fashion. Such handling is challenging, especially when different applications are assorted with conflicting requirements. It must also be dynamic due to the changes brought by application execution as well as other factors (e.g., application requirement and Cloud infrastructure changes).

Data growth Data sets can grow over time. Only one method [78] in the previous analysis is able to handle the data size change. It employs a threshold-based approach to check when data needs to be moved or when resources are adequate for storing the data to also handle their growth. However, no detailed explanation is supplied concerning how the threshold is computed.

Data replication It is usually challenging to find the best possible trade-off between cost and replication degree to enable cost-effective data replication.

Optimisation criteria Data transfer and replication management is a complex process [109] due to the completely distributed nature of the Cloud ecosystem. It further gets complicated due to the unequal data access speed. Data transfer number or cost is a well-accepted criterion for optimising data placement. However, it can be also quite restrictive. First, as there can be cases where both of these two metrics need to be considered. For instance, suppose that we need to place two datasets, initially situated in one VM, to other VMs as this VM will become soon unavailable. If we just consider the transfer number, this can lead to the situation where the movement is performed in an arbitrary way even migrating data to another DC while there is certainly a place in the current one. In the opposite direction, there can be cases where cost could be minimised but this could lead to increasing the number of transfers which could impact application performance. Second, data placement has been mainly seen in an isolated manner without examining user requirements. However, it can greatly affect application performance and cost.

Additional information Apart from extracting data access patterns and node profiles, we believe that more information is needed for a better data placement solution.

Future research directions

- Fixed data set size: To guarantee the true, optimal satisfaction of the user requirements and optimisation objectives, we suggest *the use of semi-fixed constraints in a more suitable and flexible manner as a respective non-static part of the location-aware optimisation problem to be solved.*
- Constraint solving: We propose *the use of hybrid approaches* (i.e., combining exhaustive and meta-search heuristic techniques) so as to rapidly get (within an acceptable and practically employable execution time) optimal or near-optimal results in a scalable fashion. For instance, constraint programming could be combined with local search. The first could be used to find a good initial solution, while the latter could be used for neighbourhood search to find a better result. In addition, it might be possible that *a different and more scalable modelling of the optimisation problem* could enable to run standard exhaustive solution techniques even with medium-

sized problem instances. Finally, *solution learning from history* could be adopted to fix parts of the optimisation problem and thus substantially reduce the solution space to be examined.

- Granularity: There is a need for *dynamic approaches for data placement* which do take into account the *workload fluctuation* and the *data growth* to both partition data as well as optimally place them in a set of resources with a *size that is dynamically identified*.
- Multiple applications: To handle applications conflicting requirements and the dynamicity of context (e.g., change of infrastructure, application requirements), *different techniques to solve the (combined) optimisation problem* are required. First, *soft constraints* could be used to solve this problem, even if it is over-constrained (e.g., producing a solution that violates the least number of these preferences). Next, we could *prioritise the applications and/or their tasks*. Third, *distributed solving techniques* could be used to produce application-specific optimisation problems of reduced complexity. This would require a *transformation of the overall problem into sub-problems which retains as much as possible the main constraints and requirements of each relevant application*. Finally, complementary to these distributed solving techniques, the *measure of replication* could also be employed. By using such a measure, we enable each application to operate over its own copy of the data originally shared. This could actually enable to *have complete independence of applications* which would then allow us to solve data placement individually for each of these applications.
- Data growth: There is a need to employ a *more sophisticated approach* which exploits the *data (execution) history* as well as *data size prediction and data (type) similarity techniques* to solve the data growth issue. Similarity can be learned by knowing *the context of data* (e.g., by assuming the same context has been employed for similar data over time by multiple users), while *statistical methods* can predict the data growth. Such an approach can also be used for new data sets for which no prior knowledge exists (known as the *cold-start problem*).
- Data replication: For data replication, we suggest to dynamically *compute the replication degree by considering the application size, data size, data access pattern, data growth rate, user requirements, and the capabilities of Cloud services*. Such a solution could also rely on *a weight calculation method* for the determination of the relative importance of each of these factors.
- Optimisation criteria: An interesting research direction compiles into *exploring ways via data placement and task scheduling could be either solved in conjunction or in a clever but independent way such that they do take into account the same set of (high-level) user requirements*. This could lead to producing solutions which are in concert and also optimal according to both aspects of data and computation.
- Additional information: We advocate that the additional information required to be collected or derived include: (i) *co-locating frequently accessing tasks and data*; (ii) *exploiting data dependencies to have effective data partitioning*. A similar approach is employed by Wang et al. [83] where data are grouped together at a finer granularity. There are also precautions in not storing different data blocks from the same data in the same node; (iii) *data variability* data can be of different forms. Each form might require a different machine configuration for optimal storage and processing.

In this case, *profiling should be extended to also capture this kind of machine performance variation* which could be quite beneficial for *more data-form-focused placement*. In fact, we see that whole approaches are dedicated to dealing with different data forms. For instance, graph analytics-oriented data placement algorithms exploit the fact that data are stored in the form of graphs to more effectively select the right techniques and algorithms for solving the data placement problem. While special-purpose approaches might be suitable for different data forms, they are not the right choice for handling different kinds of data. As such, we believe that an important future direction should be the ability to *more optimally handle data of multiple forms* to enhance the applicability of a data placement algorithm and make it suitable for handling different kinds of applications instead of a single one.

Concluding remarks

The primary aim of this survey is to provide a holistic overview of the state of the art related to both data storage and placement in the Cloud ecosystem. We acknowledge that there do exist some surveys on various aspects of Big Data, which focus on the functional aspect and mainly on Big Data storage issues. However, this survey plays a complementary role with respect to them. In particular, we cover multiple parts of the Big Data management architecture (such as DLM, data storage systems, data placement techniques), which were neglected in the other surveys, under the prism of non-functional properties. Further, our contribution to Big Data placement is quite unique. In addition, the in-depth analysis of each main article section is covered by a well-designed set of evaluation criteria. Such an analysis also assists in a better categorization of the respective approaches (or technologies, involved in each part).

Our survey enables readers to better understand which solution could be utilized under which non-functional requirements. Thus, assisting towards the construction of user-specific Big Data management systems according to the non-functional requirements posted. Subsequently, we have described relevant challenges that can pave the way for the proper evolution of such systems in the future. Each challenge prescribed in "Lessons learned and future research directions" section has been drawn from the conducted analysis. Lastly, we have supplied a set of interesting and emerging future research work directions concerning both the functionalities related to the Big Data management (i.e., Big Data storage and placement), as well as the Big Data lifecycle management as a whole, in order to address the identified challenges.

Abbreviations

ACL: access control list; BDaaS: Big Data-as-a-service; BEA: bond energy algorithm; BI: business intelligence; CDM: cognitive data management; DaaS: data-as-a-service; DBaaS: database-as-a-service; DCs: data centers; DDBMS: distributed database management system; DFS: distributed file system; DLMS: data lifecycle management system; DMS: data management system; HA: high availability; HDFS: Hadoop distributed file system; IoT: internet-of-thing; NFS: Network File System; PaaS: platform-as-a-service; PSO: particle swarm optimization; PVFS: parallel virtual file system; QoS: quality of service; SLR: systematic literature review; STaaS: storage-as-a-service.

Authors' contributions

"Introduction" is contributed by SM, DS, KK and YV; "Data lifecycle management (DLM)" is contributed by SM, KK and YV; "Methodology" is contributed by KK and SM; "Non-functional data management features" is contributed by DS and SM; "Data storage systems" is contributed by DS, SM and YV; "Data placement techniques" is contributed by KK and SM; and "Lessons learned and future research directions" is contributed by YV, SM, DS, KK. All authors read and approved the final manuscript.

Author details
[1] Simula Research Laboratory, 1325 Lysaker, Norway. [2] Ulm University, Ulm, Germany. [3] ICS-FORTH, Heraklion, Crete, Greece. [4] Institute of Communication and Computer Systems (ICCS), 9 Iroon Polytechniou Str., Athens, Greece.

Acknowledgements
The research leading to this survey paper has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 731664. The authors would like to thank the partners of the MELODIC project (http://www.melodic.cloud/) for their valuable advices and comments.

Competing interests
The authors declare that they have no competing interests.

Availability of data and materials
Not applicable.

Funding
This work is generously supported by the Melodic project (Grant Number 731664) of the European Union H2020 program.

Publisher's Note

Received: 4 October 2018 Accepted: 22 January 2019
Published online: 11 February 2019

References

1. Khan N, Yaqoob I, Hashem IAT, et al. Big data: survey, technologies, opportunities, and challenges. Sci World J. 2014;2014:712826.
2. Kaisler S, Armour F, Espinosa JA, Money W. Big data: issues and challenges moving forward. In: System sciences (HICSS), 2013 46th Hawaii international conference on, IEEE. 2013. pp. 995–1004.
3. Zheng Z, Zhu J, Lyu MR. Service-generated big data and big data-as-a-service: an overview. In: Big Data (BigData Congress), 2013 IEEE international congress on, IEEE. 2013. pp. 403–10.
4. Chen M, Mao S, Liu Y. Big data: a survey. Mob Netw Appl. 2014;19(2):171–209.
5. Inukollu VN, Arsi S, Ravuri SR. Security issues associated with big data in cloud computing. Int J Netw Secur Appl. 2014;6(3):45.
6. Wang C, Wang Q, Ren K, Lou W. Privacy-preserving public auditing for data storage security in cloud computing. In: Infocom, 2010 proceedings IEEE, IEEE. 2010. pp. 1–9.
7. Chaudhuri S. What next?: a half-dozen data management research goals for big data and the cloud. In: PODS, Scottsdale, AZ, USA. 2012. pp. 1–4.
8. Najafabadi MM, Villanustre F, Khoshgoftaar TM, Seliya N, Wald R, Muharemagic E. Deep learning applications and challenges in big data analytics. J Big Data. 2015;2(1):1.
9. Verma D. Supporting service level agreements on IP networks. Indianapolis: Macmillan Technical Publishing; 1999.
10. Sakr S, Liu A, Batista DM, Alomari M. A survey of large scale data management approaches in cloud environments. IEEE Commun Surv Tutor. 2011;13(3):311–36.
11. Wu L, Yuan L, You J. Survey of large-scale data management systems for big data applications. J Comput Sci Technol. 2015;30(1):163.
12. Oussous A, Benjelloun FZ, Lahcen AA, Belfkih S. Big data technologies: a survey. J King Saud Univ Comput Inf Sci. 2017;30(4):431–48.
13. Grolinger K, Higashino WA, Tiwari A, Capretz MA. Data management in cloud environments: NoSQL and NewSQL data stores. J Cloud Comput Adv Syst Appl. 2013;2(1):22.
14. Zhang H, Chen G, Ooi BC, Tan KL, Zhang M. In-memory big data management and processing: a survey. IEEE Trans Knowl Data Eng. 2015;27(7):1920–48.
15. Hashem IAT, Yaqoob I, Anuar NB, Mokhtar S, Gani A, Khan SU. The rise of "big data" on cloud computing: review and open research issues. Inf Syst. 2015;47:98–115.
16. Ball A. Review of data management lifecycle models. Bath: University of Bath; 2012.
17. Demchenko Y, de Laat C, Membrey P. Defining architecture components of the big data ecosystem. In: International conference on collaboration technologies and systems. 2014. pp. 104–12.
18. Pääkkönen P, Pakkala D. Reference architecture and classification of technologies, products and services for big data systems. Big Data Res. 2015;2(4):166–86.
19. NBD-PWG. NIST big data interoperability framework: volume 2, big data taxonomies. Tech. rep., NIST, USA 2015. Special Publication 1500-2.
20. Organisation for Economic Co-operation and Development. Data-driven innovation: big data for growth and well-being. Paris: OECD Publishing; 2015.
21. Kaufmann M. Towards a reference model for big data management. Research report, University of Hagen. 2016. Retrieved from https://ub-deposit.fernuni-hagen.de/receive/mir_mods_00000583. Retrieved 15 July 2016.
22. Höfer C, Karagiannis G. Cloud computing services: taxonomy and comparison. J Internet Serv Appl. 2011;2(2):81–94.
23. Baru C, Moore R, Rajasekar A, Wan M. The sdsc storage resource broker. In: CASCON first decade high impact papers, IBM Corp.; 2010. pp. 189–200.

24. Chasen JM, Wyman CN. System and method of managing metadata data 2004. US Patent 6,760,721.
25. Gómez A, Merseguer J, Di Nitto E, Tamburri DA. Towards a uml profile for data intensive applications. In: Proceedings of the 2nd international workshop on quality-aware DevOps, ACM. 2016. pp. 18–23.
26. Verginadis Y, Pationiotakis I, Mentzas G. Metadata schema for data-aware multi-cloud computing. In: Proceedings of the 14th international conference on INnovations in Intelligent SysTems and Applications (INISTA). IEEE. 2018.
27. Binz T, Breitenbücher U, Kopp O, Leymann F. Tosca: portable automated deployment and management of cloud applications. In: Advanced web services. Springer; 2014. pp. 527–49.
28. Kritikos K, Domaschka J, Rossini A. Srl: a scalability rule language for multi-cloud environments. In: Cloud computing technology and science (CloudCom), 2014 IEEE 6th international conference on, IEEE. 2014. pp. 1–9.
29. Rossini A, Kritikos K, Nikolov N, Domaschka J, Griesinger F, Seybold D, Romero D, Orzechowski M, Kapitsaki G, Achilleos A. The cloud application modelling and execution language (camel). Tech. rep., Universität Ulm 2017.
30. Das S, Nishimura S, Agrawal D, El Abbadi A. Albatross: lightweight elasticity in shared storage databases for the cloud using live data migration. Proc VLDB Endow. 2011;4(8):494–505.
31. Lu C, Alvarez GA, Wilkes J. Aqueduct: online data migration with performance guarantees. In: Proceedings of the 1st USENIX conference on file and storage technologies, FAST '02. USENIX Association 2002.
32. Stonebraker M, Devine R, Kornacker M, Litwin W, Pfeffer A, Sah A, Staelin C. An economic paradigm for query processing and data migration in mariposa. In: Parallel and distributed information systems, 1994., proceedings of the third international conference on, IEEE. 1994. pp. 58–67.
33. Brubeck DW, Rowe LA. Hierarchical storage management in a distributed VOD system. IEEE Multimedia. 1996;3(3):37–47.
34. Kitchenham BA, Pfleeger SL, Pickard LM, Jones PW, Hoaglin DC, Emam KE, Rosenberg J. Preliminary guidelines for empirical research in software engineering. IEEE Trans Softw Eng. 2002;28(8):721–34.
35. Gessert F, Wingerath W, Friedrich S, Ritter N. NoSQL database systems: a survey and decision guidance. Comput Sci Res Dev. 2017;32(3–4):353–65.
36. Sakr S. Cloud-hosted databases: technologies, challenges and opportunities. Clust Comput. 2014;17(2):487–502.
37. Cattell R. Scalable SQL and NoSQL data stores. Acm Sigmod Rec. 2011;39(4):12–27.
38. Gray J. Database and transaction processing performance handbook. In: The benchmark handbook for database and transaction systems. 2nd ed. Digital Equipment Corp. 1993.
39. Traeger A, Zadok E, Joukov N, Wright CP. A nine year study of file system and storage benchmarking. ACM Trans Storage. 2008;4(2):5.
40. Agrawal D, El Abbadi A, Das S, Elmore AJ. Database scalability, elasticity, and autonomy in the cloud. In: International conference on database systems for advanced applications. Springer. 2011. pp. 2–15.
41. Séguin C, Le Mahec G, Depardon B. Towards elasticity in distributed file systems. In: Cluster, cloud and grid computing (CCGrid), 2015 15th IEEE/ACM international symposium on, IEEE. 2015. pp. 1047–56.
42. Seybold D, Wagner N, Erb B, Domaschka J. Is elasticity of scalable databases a myth? In: Big Data (Big Data), 2016 IEEE international conference on, IEEE. 2016. pp. 2827–36.
43. Gilbert S, Lynch N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. Acm Sigact News. 2002;33(2):51–9.
44. Bermbach D, Kuhlenkamp J. Consistency in distributed storage systems. In: Networked systems. Springer. 2013. pp. 175–89.
45. Lakshman A, Malik P. Cassandra: a decentralized structured storage system. ACM SIGOPS Oper Syst Rev. 2010;44(2):35–40.
46. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE. Bigtable: a distributed storage system for structured data. ACM Trans Comput Syst. 2008;26(2):4.
47. Pavlo A, Aslett M. What's really new with newsql? ACM Sigmod Rec. 2016;45(2):45–55.
48. Corbellini A, Mateos C, Zunino A, Godoy D, Schiaffino S. Persisting big-data: the NoSQL landscape. Inf Syst. 2017;63:1–23.
49. Davoudian A, Chen L, Liu M. A survey on NoSQL stores. ACM Comput Surv. 2018;51(2):40.
50. Jensen SK, Pedersen TB, Thomsen C. Time series management systems: a survey. IEEE Trans Knowl Data Eng. 2017;29(11):2581–600.
51. Bader A, Kopp O, Falkenthal M. Survey and comparison of open source time series databases. In: BTW (Workshops). 2017. pp. 249–68.
52. Abadi JD. Data management in the cloud: limitations and opportunities. IEEE Data Eng Bull. 2009;32:3–12.
53. Pritchett D. Base: an acid alternative. Queue. 2008;6(3):48–55.
54. Codd EF. Extending the database relational model to capture more meaning. ACM Trans Database Syst. 1979;4(4):397–434.
55. Aslett M. How will the database incumbents respond to nosql and newsql. The San Francisco. 2011;451:1–5.
56. Sadalage PJ, Fowler M. NoSQL distilled. 2012. ISBN-10 321826620
57. Seybold D, Hauser CB, Volpert S, Domaschka J. Gibbon: an availability evaluation framework for distributed databases. In: OTM confederated international conferences "On the Move to Meaningful Internet Systems". Springer. 2017. pp. 31–49.
58. Seybold D, Domaschka J. Is distributed database evaluation cloud-ready? In: Advances in databases and information systems. Springer. 2017. pp. 100–8.
59. Barahmand S, Ghandeharizadeh S. BG: a benchmark to evaluate interactive social networking actions. In: CIDR. Citeseer. 2013.
60. Cooper BF., Silberstein A, Tam E, Ramakrishnan R, Sears R. Benchmarking cloud serving systems with ycsb. In: Proceedings of the 1st ACM symposium on Cloud computing, ACM. 2010. pp. 143–54.
61. Kuhlenkamp J, Klems M, Röss O. Benchmarking scalability and elasticity of distributed database systems. Proc VLDB Endow. 2014;7(12):1219–30.
62. Bermbach D, Tai S. Benchmarking eventual consistency: lessons learned from long-term experimental studies. In: Cloud engineering (IC2E), 2014 IEEE international conference on, IEEE. 2014. pp. 47–56.

63. Domaschka J, Hauser CB, Erb B. Reliability and availability properties of distributed database systems. In: Enterprise distributed object computing conference (EDOC), 2014 IEEE 18th international, IEEE. 2014. pp. 226–33.
64. Brewer E. Cap twelve years later: How the "rules" have changed. Computer. 2012;45(2):23–9.
65. Klems M, Bermbach D, Weinert R. A runtime quality measurement framework for cloud database service systems. In: Quality of information and communications technology (QUATIC), 2012 eighth international conference on the, IEEE. 2012. pp. 38–46.
66. Abadi D, Agrawal R, Ailamaki A, Balazinska M, Bernstein PA, Carey MJ, Chaudhuri S, Chaudhuri S, Dean J, Doan A. The beckman report on database research. Commun ACM. 2016;59(2):92–9.
67. Group NBDPW, et al. Nist big data interoperability framework. Special Publication 2015. pp. 1500–6.
68. Kachele S, Spann C, Hauck FJ, Domaschka J. Beyond iaas and paas: an extended cloud taxonomy for computation, storage and networking. In: Utility and cloud computing (UCC), 2013 IEEE/ACM 6th international conference on, IEEE. 2013. pp. 75–82.
69. Levy E, Silberschatz A. Distributed file systems: concepts and examples. ACM Comput Surv. 1990;22(4):321–74.
70. Muthitacharoen A, Morris R, Gil TM, Chen B. Ivy: a read/write peer-to-peer file system. ACM SIGOPS Oper Syst Rev. 2002;36(SI):31–44.
71. Ross RB, Thakur R, et al. PVFS: a parallel file system for Linux clusters. In: Proceedings of the 4th annual Linux showcase and conference. 2000. pp. 391–430.
72. Yu B, Pan J. Sketch-based data placement among geo-distributed datacenters for cloud storages. In: INFOCOM, San Francisco: IEEE. 2016. pp. 1–9.
73. Greene WS, Robertson JA. Method and system for managing partitioned data resources. 2005. US Patent 6,922,685.
74. Greenberg A, Hamilton J, Maltz DA, Patel P. The cost of a cloud: research problems in data center networks. ACM SIGCOMM Comput Commun Rev. 2008;39(1):68–73.
75. Hardavellas N, Ferdman M, Falsafi B, Ailamaki A. Reactive nuca: near-optimal block placement and replication in distributed caches. ACM SIGARCH Comput Archit News. 2009;37(3):184–95.
76. Kosar T, Livny M. Stork: making data placement a first class citizen in the grid. In: Distributed computing systems, 2004. Proceedings. 24th international conference on, IEEE. 2004. pp. 342–9.
77. Xie T. Sea: a striping-based energy-aware strategy for data placement in raid-structured storage systems. IEEE Trans Comput. 2008;57(6):748–61.
78. Yuan D, Yang Y, Liu X, Chen J. A data placement strategy in scientific cloud workflows. Future Gener Comput Syst. 2010;26(8):1200–14.
79. Doraimani S, Iamnitchi A. File grouping for scientific data management: lessons from experimenting with real traces. In: Proceedings of the 17th international symposium on High performance distributed computing, ACM. 2008. pp. 153–64.
80. Cope JM, Trebon N, Tufo HM, Beckman P. Robust data placement in urgent computing environments. In: Parallel & distributed processing, 2009. IPDPS 2009. IEEE international symposium on, IEEE. 2009. pp. 1–13.
81. Kosar T, Livny M. A framework for reliable and efficient data placement in distributed computing systems. J Parallel Distrib Comput. 2005;65(10):1146–57.
82. Bell DA. Difficult data placement problems. Comput J. 1984;27(4):315–20.
83. Wang J, Shang P, Yin J. Draw: a new data-grouping-aware data placement scheme for data intensive applications with interest locality. IEEE Trans Magnetic. 2012;49(6):2514–20.
84. McCormick W, Schweitzer P, White T. Problem decomposition and data reorganisation by a clustering technique. Oper Res. 1972;20:993–1009.
85. Ebrahimi M, Mohan A, Kashlev A, Lu S. BDAP: a Big Data placement strategy for cloud-based scientific workflows. In: BigDataService, IEEE computer society. 2015. pp. 105–14.
86. Papaioannou TG, Bonvin N, Aberer K. Scalia: an adaptive scheme for efficient multi-cloud storage. In: Proceedings of the international conference on high performance computing, networking, storage and analysis. IEEE Computer Society Press. 2012. p. 20.
87. Er-Dun Z, Yong-Qiang Q, Xing-Xing X, Yi C. A data placement strategy based on genetic algorithm for scientific workflows. In: CIS, IEEE computer society. 2012. pp. 146–9.
88. Rafique A, Van Landuyt D, Reniers V., Joosen W. Towards an adaptive middleware for efficient multi-cloud data storage. In: Proceedings of the 4th workshop on CrossCloud infrastructures & platforms, Crosscloud'17. 2017. pp. 1–6.
89. Lan K, Fong S, Song W, Vasilakos AV, Millham RC. Self-adaptive pre-processing methodology for big data stream mining in internet of things environmental sensor monitoring. Symmetry. 2017;9(10):244.
90. Zhang J, Chen J, Luo J, Song A. Efficient location-aware data placement for data-intensive applications in geo-distributed scientific data centers. Tsinghua Sci Technol. 2016;21(5):471–81.
91. Hsu CH, Slagter KD, Chung YC. Locality and loading aware virtual machine mapping techniques for optimizing communications in mapreduce applications. Future Gener Comput Syst. 2015;53:43–54.
92. Xu Q, Xu Z, Wang T. A data-placement strategy based on genetic algorithm in cloud computing. Int J Intell Sci. 2015;5(3):145–57.
93. Chervenak AL, Smith DE, Chen W, Deelman E. Integrating policy with scientific workflow management for data-intensive applications. In: 2012 SC companion: high performance computing, networking storage and analysis. 2012. pp. 140–9.
94. Fedak G, He H, Cappello F. Bitdew: a programmable environment for large-scale data management and distribution. In: 2008 SC—international conference for high performance computing, networking, storage and analysis. 2008. pp. 1–12.
95. Deelman E, Singh G, Su MH, Blythe J, Gil Y, Kesselman C, Mehta G, Vahi K, Berriman GB, Good J. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Sci Program. 2005;13(3):219–37.

96. Chervenak A, Deelman E, Foster I, Guy L, Hoschek W, Iamnitchi A, Kesselman C, Kunszt P, Ripeanu M, Schwartzkopf B, et al. Giggle: a framework for constructing scalable replica location services. In: Proceedings of the 2002 ACM/IEEE conference on supercomputing. IEEE computer society press. 2002. pp. 1–17.
97. LeBeane M, Song S, Panda R, Ryoo JH, John LK. Data partitioning strategies for graph workloads on heterogeneous clusters. In: SC, Austin: ACM; 2015. pp. 1–12.
98. Quamar A, Kumar KA, Deshpande A. Sword: scalable workload-aware data placement for transactional workloads. In: Proceedings of the 16th international conference on extending database technology, EDBT '13, ACM. 2013. pp. 430–41.
99. Kumar KA, Deshpande A, Khuller S. Data placement and replica selection for improving co-location in distributed environments. CoRR 2012. arXiv:1302.4168.
100. Catalyurek UV, Kaya K, Uçar B. Integrated data placement and task assignment for scientific workflows in clouds. In: Proceedings of the Fourth International Workshop on Data-intensive Distributed Computing, New York, NY, USA: ACM; 2011. pp. 45–54.
101. Baur D, Seybold D, Griesinger F, Tsitsipas A, Hauser CB, Domaschka J. Cloud orchestration features: are tools fit for purpose? In: Utility and Cloud Computing (UCC), 2015 IEEE/ACM 8th international conference on, IEEE. 2015. pp. 95–101.
102. Burns B, Grant B, Oppenheimer D, Brewer E, Wilkes J. Borg, omega, and kubernetes. Queue. 2016;14(1):10.
103. Schad J, Dittrich J, Quiané-Ruiz JA. Runtime measurements in the cloud: observing, analyzing, and reducing variance. Proc VLDB Endow. 2010;3(1–2):460–71.
104. Thanh TD, Mohan S, Choi E, Kim S, Kim P. A taxonomy and survey on distributed file systems. In: Networked computing and advanced information management, 2008. NCM'08. Fourth international conference on, vol. 1, IEEE. 2008. pp. 144–9.
105. Ananthanarayanan G, Ghodsi A, Shenker S, Stoica I. Disk-locality in datacenter computing considered irrelevant. In: HotOS. 2011. p. 12.
106. Nightingale EB, Chen PM, Flinn J. Speculative execution in a distributed file system. In: ACM SIGOPS operating systems review, vol. 39, ACM. 2005. pp. 191–205.
107. Coelho F, Paulo J, Vilaça R, Pereira J, Oliveira R. Htapbench: Hybrid transactional and analytical processing benchmark. In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, ACM; 2017. pp. 293–304.
108. Seybold D, Keppler M, Gründler D, Domaschka J. Mowgli: Finding your way in the DBMS jungle. In: Proceedings of the 2019 ACM/SPEC international conference on performance engineering. ACM. 2019.
109. Allen MS, Wolski R. The livny and plank-beck problems: studies in data movement on the computational grid. In: Supercomputing, 2003 ACM/IEEE conference, IEEE. 2003. pp. 43.

Chapter 9

[core2] Cloud orchestration features: Are tools fit for purpose?

This article is published as follows:

Daniel Baur, Daniel Seybold, Frank Griesinger, Athanasios Tsitsipas, Christopher B Hauser, and Jörg Domaschka. "Cloud orchestration features: Are tools fit for purpose?" in *IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, 2015, IEEE, pp. 95–101, DOI: https://doi.org/10.1109/UCC.2015.25.

Cloud Orchestration Features: Are Tools Fit for Purpose?

Daniel Baur, Daniel Seybold, Frank Griesinger, Athanasios Tsitsipas, Christopher B. Hauser, Jörg Domaschka
Institute of Information Resource Management
University of Ulm, Germany
Email: {firstname.lastname,joerg.domaschka}@uni-ulm.de

***Abstract*—Even though the cloud era has begun almost one decade ago, many problems of the first hour are still around. Vendor lock-in and poor tool support hinder users from taking full advantage of main cloud features: dynamic and scale. This has given rise to tools that target the seamless management and orchestration of cloud applications. All these tools promise similar capabilities and are barely distinguishable what makes it hard to select the right tool. In this paper, we objectively investigate required and desired features of such tools and give a definition of them. We then select three open-source tools (Brooklyn, Cloudify, Stratos) and compare them according to the features they support using our experience gained from deploying and operating a standard three-tier application. This exercise leads to a fine-grained feature list that enables the comparison of such tools based on objective criteria as well as a rating of three popular cloud orchestration tools. In addition, it leads to the insight that the tools are on the right track, but that further development and particularly research is necessary to satisfy all demands.**

I. Introduction

The last decade has been dominated by the Cloud in both research and industry. Nevertheless, many problems have been around since the beginning of the cloud era and are substantially hindering the move forward. These include vendor lock-in, the incomparability of cloud providers with respect to performance per price, the weak adoption of cloud standards from the providers, etc.

Particularly, researchers from domains that make use of clouds such as software engineering and data mining, but also software architects and DevOps teams need robust and powerful mechanisms to bring their applications and innovations to the Cloud—ideally to many cloud providers. In order to perform evaluations and test the elasticity of their application, these users need to be able to seamlessly change the distribution of their application across multiple cloud providers. They also have to be able to quickly change the scaling factor of individual components. Similar considerations hold for start-ups whose demands change with the growth of the business.

In order to satisfy these demands, a powerful and reliable cloud orchestration and operation platform is needed. Indeed, there are multiple commercial and open-source tools available that promise to solve above mentioned issues. Yet, it is hard to find a visible distinguishing factor. Also, there are barely any sources available that report on the success of using these tools. This document aims at closing this information gap by the following contributions: *(i)* We establish a fine-grained feature list that enables the comparison of such orchestration tools based on objective criteria. We also provide a guideline on how to grade tools according to the features. *(ii)* We identify tools that promise to realise some or multiple of the above mentioned features. *(iii)* We rate three tools according to our feature list.

Based on our feature extraction, it becomes possible to compare existing and upcoming orchestration tools: What is more, the identified lack of supported features shall drive future research and development with respect to cloud orchestration. The remainder of this document is structured as follows: Section II introduces the terminology of the document as well as the methodology of our approach. Section III introduces background and basic capabilities of the tools. Section IV introduces the feature list and does the comparison for each introduced feature. Section V presents the comparison results from applying the list in tabular I. Section VI discusses related work before we conclude.

II. Background

In the following, we briefly introduce the terminology that we use throughout the document and further classify the methodology of our comparison.

A. Terminology

Even though more advanced terminologies have been published (e.g. [1]), we follow the well-known NIST standard [2] defining the three service models IaaS, PaaS, and SaaS.

We define *(cloud) application* as a possibly distributed application consisting of multiple interlinked application components. For illustration consider a blog application B that may consist of the three components load balancer lb, application server together with the business logic as, and a database db.

We depict cloud orchestration tool, as a software component that manages the complete lifecycle of a cloud application. It therefore needs to fulfil the following criteria: *(1)* application specification (definition); *(2)* deployment of specified applications: the tool has to acquire virtual machines and then distribute component instances over them; *(3)* collection of monitoring data from deployed instances. This data further has to be consolidated and aggregated and provided to the operators. *(4)* If application management shall be (semi-)automatic, the tool has to allow the definition of rules that capture when to execute which management task (e.g. scaling). We therefore explicitly exclude pure deployment tools like Chef.

B. Methodology

While the primary goal of this document is to present a feature list that helps rating cloud development tools a secondary goal is to show its applicability to existing software tools. For this purpose we select three tools fulfilling the above mentioned four criteria: Apache Brooklyn[1], Cloudify[2], and Apache Stratos[3].

For the evaluation we install the tools in our local data centre and apply the defined features. As a sample application we select a three-tier application consisting of *NGINX* as a load balancer, *Node.js* as an application server running the *Ghost* blogging application, and *PostgreSQL* as a database management system. The used application descriptions are available on Github[4].

III. Introduction to Tools

All three selected tools make use of third party IaaS platforms for running applications and all of them offer a PaaS-like interface to operators. Below we introduce the tools in particular with respect to their terminology and define the versions used for the evaluation. As a rule of thumb, we used the latest stable version available at the time writing this document.

Apache Brooklyn is compared using milestone release 0.7.0-M2-incubating. Brooklyn's application description is based on *blueprints* written in a *domain specific language* (DSL) in YAML format. A blueprint contains global properties (e.g. name, cloud provider configuration) as well as a services block defining application components. The definition of the components is split into abstract types and entities. The type captures the structure for entities such as defining the properties and interfaces. The entities refer to those types and provide the concrete configuration. Each type is linked to a Java implementation. Besides deployment, Brooklyn also supports a monitoring interface and elastic adaptation at runtime.

Cloudify by *GigaSpaces Technologies* is offered in a free open-source edition and a commercial Pro edition. Our comparison is based on the open-source edition version 3.2. Cloudify uses a TOSCA-aligned modelling language for describing the topology of the application which is then deployed to allocated virtual machines in the cloud environment. TOSCA-like, Cloudify splits the blueprint in a type and template definition. Types define abstract reusable entities and are to be referenced by templates. The types therefore define the structure of the template, by e.g. defining the properties that a template can have/must provide. The template then provides the concrete values. This mechanism is used for nodes as well as for relationships.

Apache Stratos is compared using the release candiate 4.1.0-RC2. Stratos makes use of an abstract virtual machine description, named *cartridge*, with an application component type (named *cartridge type*) like an application runtime container (e.g. Tomcat). An application is described by a single cartridge or/and a set of cartridges (*groups*), combined with deployment and scaling policies. The cartridges, applications and other configurations are represented in a Stratos specific JSON format. For the deployment itself, it solely relies on the DevOps tool Puppet. The application itself is subsequent cloned from a Git repository. Stratos is installed as one central controller and in all virtual machines by having a virtual machine image prepared with the necessary software (Stratos and Puppet agents) installed.

[1]https://brooklyn.incubator.apache.org/

[2]http://getcloudify.org/

[3]http://stratos.apache.org/

[4]https://github.com/dbaur/orchestration_comparison

IV. Features and Comparison

In this section we perform the actual comparison. Each of the following two sections introduces a set of features. Section IV-A addresses cloud-related aspects and Section IV-B considers application-related features.

In the individual sections, we introduce the features of the set and for each of them, *(i)* define the feature in detail and argue why it is desirable. In case sub-features exist for a feature, they are introduced as well (highlighted in italics). Moreover, we *(ii)* discuss to what extend each feature and all its sub-features are actually supported by each of the three tools.

A. Cloud Feature Set

The Cloud Feature Set relates to the cloud infrastructure. Hence, its features focus on supported deployment across multiple cloud providers and levels.

1) Multi-Cloud Support Feature: Supporting *multiple cloud providers* is one of the most crucial features for cloud application management tools, as it allows the selection of the best matching cloud offer for an application from a diverse offering landscape. Cloud providers often differ from each other regarding their API. This causes the user to suffer from a vendor lock-in once he depends on the native API of a cloud provider. For that reason cloud orchestration tools should offer a *cloud abstraction layer* which hides differences and avoids the need for provider-specific customisation thus removing the vendor lock-in.

Apache Brooklyn Support: Brooklyn uses Apache jclouds as cloud abstraction layer and therefore supports many public and private cloud providers.

Cloudify Support: Cloudify comes with plugins supporting AWS, Openstack and VMWare vCloud. It also offers a contributed plugin for Apache Cloudstack and two commercial plugins (Pro version) for VMWare vSphere and SoftLayer. Nevertheless, Cloudify does not support an abstraction layer and each model needs to explicitly reference cloud provider specific features.

Apache Stratos Support: Stratos utilises jclouds as a cloud abstraction layer, supporting multiple providers. Support is tested for AWS EC2, Openstack and Google Compute Engine. Yet, the abstraction is imperfect as application specifications still need to refer to cloud specific entities.

2) Cross-Cloud Support Feature: Cross-cloud support enhances the multi-cloud feature such that the user is able to deploy a single application in a way that its component instances are distributed over multiple cloud providers. For instance, the database may be deployed in a private cloud on the user's premises while numerous instances of the application

server run in a public cloud. The advantages of cross-cloud deployment are three-fold: *(i)* It allows a sophisticated per-component instance selection of the best-fitting offer. *(ii)* It leverages the availability of the application as it introduces resilience against the failure of individual cloud providers. *(iii)* It helps coping with privacy issues (private vs. public cloud).

Apache Brooklyn Support: Brooklyn supports cross-cloud deployments on a per-component level: Each component can be bound to a separate cloud provider by referencing its configuration.

Cloudify Support: Cloudify offers cross-cloud support. For each virtual machine defined in the model, the user can reference a different cloud provider.

Apache Stratos Support: Stratos allows the definition of network partitions which are logical groups of IaaS resources such as regions or availability zones. Network partitions enable cross-cloud scaling and deployment using policies like round robin through available network partitions. Cartridges may only be configured for a subset of network partitions.

3) External PaaS Support Feature: In addition to supporting IaaS clouds, the support of PaaS clouds (e.g. Google App Engine) is desirable. For PaaS offers ready-to-deploy application containers, it reduces complexity compared to IaaS. This also reduces the management effort for the user. On the downside, it comes at the cost of reduced flexibility, it is the provider that defines the container configuration.

Tool Support: None of the three tools allows the use of external PaaS clouds.

4) Support of Cloud Standards Feature: In addition to supporting multiple provider APIs (cf. Section IV-A1) the support of cloud API standards such as CIMI [3] or OCCI [4] enables support for any cloud provider adapting such a standard.

Tool Support: None of the three tools supports any cloud interface standard.

5) Bring Your Own Node (BYON) Feature: BYON captures the ability to use already running servers for application deployment. It enables the use of servers not managed by a cloud platform or virtual machines on unsupported cloud providers.

Apache Brooklyn Support: Brooklyn supports BYON by providing an IP address and login credentials for the server.

Cloudify Support: Cloudify supports BYON through an externally installable Host-Pool Service that works as a cloud middleware mock-up. When enabled, Cloudify requests IP addresses and login credentials from this service whenever it needs to provision a new server.

Apache Stratos Support: Stratos does not support BYON, despite the general ability of jclouds to do so.

B. Application Feature Set

This section discusses features related to the deployment and automation of applications. It starts with features related to the application description language and deployment, continues with features related to runtime adaptation, and finally discusses additional features such as support of the Windows operating system.

1) Application Standards Feature: Supporting open standards such as TOSCA [5] and CAMP [6] for modelling the application topology, the component life cycles, and the interaction with the cloud management tool facilitates the usage of the tool and further increases the reusability of the topology definition, as it avoids moving the vendor lock-in from cloud provider level to management tool level (cf. Section IV-A1). Moreover, it reduces the initial effort and costs to learn a new DSL.

Apache Brooklyn Support: Brooklyn's YAML format follows the CAMP specification, but uses some custom extensions. Yet, it is possible to deploy CAMP YAML plans with Brooklyn and via the separately provided CAMP server. Support for TOSCA is planned for a future release.

Cloudify Support: While Cloudify's DSL for the deployment description is strongly aligned with the TOSCA modelling standard it does not directly reference the standard types, but instead defines its own profile following the *TOSCA Simple Profile in YAML*. Full TOSCA support is planned.

Apache Stratos Support: Stratos does not implement any standard.

2) Resource Selection Feature: The resource selection is part of the application topology description. It defines the resources used for the deployment of a component instance in an IaaS cloud. Hence, a resource will commonly refer to the virtual machine type/flavour, an image, and a provider-specific location: $\langle location, hardware, image \rangle$. A tool has mainly four possibilities to define or derive such a tuple: *(i)* In an *manual binding* the user provides the concrete unique identifiers of the cloud entities. *(ii)* In an *automatic binding* the user defines abstract requirements regarding the defined tuple (e.g. number of cores). These are then bound to a concrete offer at runtime by the tool. Automatic binding can be enhanced by offering an *iii) optimised binding*. Here, the specification of optimisation criteria based on attributes of the cloud provider such as price or location is possible. Finally, *(iv) dynamic binding* offers a solving system that enables changes to the binding based on runtime information, e.g. metric data collected from the monitoring system. Automatic binding is a prerequisite for complex deployment and runtime adaptation scenarios, as it allows the cloud management platform to dynamically select the concrete offer during runtime. Optimised binding offers optimised selection based on simple criteria like price. Dynamic binding offers the possibility to use a solver applying an optimisation algorithm for selecting the best-fitting offer based on complex criteria (performance or performance per $).

Apache Brooklyn Support: Brooklyn supports manual as well as basic automatic binding. For the latter it supports resource boundaries for the hardware. The resource selection happens either in the global or in the component-specific parts of the blueprint.

Cloudify Support: Cloudify exclusively supports manual binding of the resources used for a virtual machine. The user needs to reference a cloud provider specific node type (e.g. `cloudify.openstack.nodes.Server` for Openstack) to provide the implementation for the chosen cloud provider, as well as the specific properties defined by this type. These include the location, the image and the flavour information.

Automatic binding of resources (like offered by TOSCA's `nodes_filter` requirements specification) is not supported by Cloudify. Due to this shortcoming optimised and dynamic bindings are also not possible.

Apache Stratos Support: The resource selection in Stratos is a manual process when configuring cartridges by referencing to *(i)* an image and *(ii)* a hardware description in an IaaS cloud.

3) Life Cycle Description Feature: The life cycle description defines the actions that need to be executed in order to deploy the application including all its component instances on started virtual machines. The basic approach for the life cycle description of the application is to provide *shell scripts* that are executed in a specific order. This approach can be extended to support *DevOps tools* such as Chef that offer a more sophisticated approach to deployment management and ready to use deployment descriptions.

Apache Brooklyn Support: In Brooklyn each defined type provides basic life cycle actions called effectors. These can be configured in the concrete application component definition. The configuration can either happen with shell scripts or by referencing Chef recipes.

Cloudify Support: Cloudify relies on the interface definition of TOSCA for defining life cycle actions. The base node type defines multiple life cycle actions as interfaces, that are executed during deployment. The actions are defined as shell scripts or by using Chef and Puppet. Support for Salt is in development. Cloudify also has the possibility to provide python scripts. This is evaluated and provides immediate access to Cloudify's API.

Apache Stratos Support: The life cycle description for managing virtual machines is done by Stratos itself, while the software setup is delegated to Puppet. Stratos cartridges have a cartridge type, which is a reference to a Puppet module. During application deployment, Stratos identifies and invokes the needed Puppet modules.

4) Wiring and Workflows: Most cloud applications are distributed applications where components reside on different virtual machines, e.g. the application server resides on a compute optimised host, while the database is on a storage optimised host. Hence, the modelling language needs to support a way to configure those communication relationships between the components by offering a way to pass the endpoint, either before the start of the dependent component (database starts before application server) or after (application server is added to already running load balancer).

A straight-forward approach to resolve those dependencies is *attribute and event passing*. That is, the tool allows the user (life cycle scripts) to lock/wait for attributes to become available or register listeners on topology change events. This is commonly achieved by a global registry shared between all component instances of an application.

Obviously, this approach offloads most complexity to the user who needs to, e.g., make sure that the database URL is only available when it already started. An improvement is a *manual workflow* definition. Here, the user defines a workflow taking care of the deployment order. Finally, the easiest way for the end user is an *automatic workflow* deduction, where the modelling language is sufficiently verbose to allow the system to automatically deduce the correct workflow from the defined life cycle actions on the virtual machines and their relationships.

Additionally, a tool may offer extensions to its model, allowing to refer to *external services* like PaaS (cf. Section IV-A3) or SaaS services, to ensure that the deployment engine is aware of this dependency and e.g., can open ports on firewalls.

Apache Brooklyn Support: Brooklyn supports wiring by attribute-and-event-passing. It offers a locking action, that waits until the dependent service provides a required attribute. The reverse way, where a later starting service needs to reconfigure a running service, is not supported out of the box. Instead, the user has to implement this functionality. Yet, for the commonly used load-balancer scenario, Brooklyn supports predefined static out-of-box load balancing. The tool neither supports workflow scenarios nor access to external services.

Cloudify Support: Cloudify uses the relationship mechanism of TOSCA. It defines a generic `depends_on` relationship type that offers the execution of custom actions on either the source or the target of the relationship on specific events. Combined with a shared configuration space available via e.g. a shell extension, this allows the user to configure endpoints before or after the start of a service. Using python the user can implement custom workflows, making sure that the life cycle actions are executed in the correct order. If the user only uses the basic life cycle actions, Cloudify is capable of automatically deducing the correct execution order. Cloudify does not support external services by default. Yet, the modular communication relationship might allow adding this feature if needed.

Apache Stratos Support: Stratos provides a metadata service where the component instances of an application can export and import variables. This basic but manual wiring using variable exchange must be implemented by the user at application setup. In case of joining or leaving component instances Stratos broadcasts a topology change event, which is used by Stratos core functionality (e.g. notify the user for a successful application deployment) or any load balancers existing in a deployed application setup, to update their state for redirecting client requests. For assessing the overall deployment workflow, support of both Stratos and Puppet have to be considered. Stratos defines a startup order of virtual machines, while Puppet has a more complex dependency expression for each single virtual machine. Puppet modules can be depending on each other and inside of one module, dependencies between different steps can be defined. This rather static deployment workflow is defined in advance of the application deployment. The flow cannot be controlled during application boot or execution time. Stratos does not support external services.

5) Monitoring Feature: Being able to track the behaviour of the application is a key to assessing the quality of the deployment. Consequently, it is necessary to monitor the current resource consumption and the quality-of-service (QoS) parameters of the application. Only if the end-user is aware of current bottlenecks he is able to remove them. The cloud management framework should therefore offer a way to measure *system metrics* like CPU usage and *application specific*

metrics like number of requests. If those predefined metrics are not sufficient, the tool should offer a well defined way to add *custom metrics*.

An *aggregation mechanism* enables users to compute higher-level metrics (e.g. 10 minute average over CPU load) and also to combine multiple metrics (e.g. average over 10 minute CPU average of all instances of a particular component). For helping users in accessing and assessing the current load on his application, it is beneficial to have the gathered metrics presented through a *dashboard*. In order to support higher-level evaluation of monitoring data *access to historical data* is desirable.

Apache Brooklyn Support: Brooklyn's uses a pulling mechanism gathering the data from the virtual machines by either executing remote actions or accessing an external tool. It is the user's responsibility to implement those actions, or to provide an interface to an external monitoring tool. Brooklyn does not store historical data and only supports access to the latest measured value impairing aggregation. The latest value of all metrics is shown in a dashboard.

Cloudify Support: Cloudify's monitoring system relies on the Diamond monitoring daemon that has built-in collectors for the most common system and application metrics. Additionally, it offers an interface for the implementation of custom collectors. The Cloudify user interface for viewing (historical) metrics is only available in the closed-source Pro version. Aggregation of metrics is possible using the policy framework (cf. Section IV-B6).

Apache Stratos Support: Stratos uses a cartridge agent residing within each virtual machine. This agent comprises a *Health Publisher* to avail itself of the machine's health statistics, load average, free memory, and the amount of in-flight requests. It is not possible to define further custom metrics. Monitoring data is sent to a central real-time processing engine where aggregation and evaluation is performed. Support for visualisation of current and historical health statistics through the Web GUI is is planned for the future.

6) Runtime Adaptation Feature: While monitoring (cf. Section IV-B5) lays one of the foundations for adapting the configuration of the application during runtime, the cloud management platform should be able to react upon deviations automatically. For this purpose, the user needs to be able to define *(i)* metric and QoS conditions that trigger *(ii)* actions if violated: For instance, scale component horizontally, if the CPU usage is $> 80\%$. In order to support that, the cloud management tool should at least offer a *simple threshold-based approach* for the detection of violations and support *horizontal scaling*. As extensions, we consider *repair* and the *custom definition* of actions on the actions side, and more complex *rule engines* with respect to QoS.

Another feature related to adaptation is *continuous delivery* of the application. It enables the user to change the topology model of an already deployed and running application (e.g. add a load balancer, or update the software version of a component). This should be possible with as few changes as needed to the running components.

Apache Brooklyn Support: Brooklyn policies enable the specification of metrics/QoS. By default a threshold-based policy is available. Scaling is enacted in so called clusters. By default Brooklyn supports horizontal scaling. Both the policies and the clusters are in general customisable by new implementations, but there is no easy way to plugin such custom extensions. Continuous delivery is exclusively possible on component level, namely by redeploying single components with updated software.

Cloudify Support: Cloudify uses the event stream processor Riemann for the definition of QoS requirements. By default, they provide policies for host failure detection, simple threshold and exponential weighted moving average threshold. It enables the definition of custom aggregations and policies using Clojure and Riemann. On the action side, Cloudify offers workflows for healing the application (uninstall/reinstall) and a scale workflow offering horizontal scaling. Complex scaling scenarios (e.g. cloud bursting, vertical scaling) are not supported out of the box. Instead, the user may define custom workflows using a DSL. This enables support for complex scenarios, but leaves the responsibility with the user. Continuous delivery of the application is currently not supported by Cloudify, meaning that the user has to un-deploy the entire application, even for minor changes in the model. Support for this has been announced for the Pro version.

Apache Stratos Support: Stratos balances the QoS requirements by using policies, that enable a multi-factored autoscaling. Using client requests and system load as health data (cf. Section IV-B5) combined with a complex event processor and the Drools rules engine, Stratos enacts horizontal scaling to the environment. Moreover, repair actions are supported, in case some tasks within virtual machines of an application topology fail (e.g. installation of required software), by automatically destroying and re-creating the affected instance. The implementation of custom actions is not foreseen. Continuous delivery is not supported. Instead, the user has to un-deploy the whole application first and change its definition.

7) Reusability and Sharing of Models Feature: When using cloud management tools, the main task of the user is the creation of the application description based on components. As this imposes a high initial effort, this task needs to be supported by sharing of existing models.

Regarding *reusability* the tool should offer a modularised approach regarding the application description. Generally speaking, each application description consists of components. In order to facilitate re-use, modularisation shall be used to an extend where the description and life cycle handling of components is mostly self-contained and independent from e.g. the application it is embedded in. At the same time the composing mechanism that forms applications from sets of components has to be powerful enough to capture the most common use cases, such as setting the port numbers wiring two components. In an ideal case, exchanging an SQL-based database in an application with a different SQL-based database should neither require changes to the invoking component definition nor to the new database component. Also, the application should be widely untouched except that the configuration parameters for the new database have to be set. Approaches to achieve this modularity include templating, parameterisation, and inheritance.

In order to facilitate easy *sharing* of entire models and parts

thereof, the tool should offer a marketplace where users can exchange their models with other users. If such a marketplace does not exist, the tool provider should at least offer application models for the most important standard services.

Apache Brooklyn Support: Brooklyn achieves the reusability of types by using inheritance. Yet types of the same parent can not be exchanged without modifying the concrete properties of connected types. Types can be shared either locally or in a Git repository.

Cloudify Support: Cloudify uses the same reusability mechanisms for its models as TOSCA: For the model is split into types and templates, defined types are in general usable in other templates. The separation of the server host and the application using the *hosted_on* relationship also decouples the server from the application description. The reusability is further increased by the import mechanism, that allows to define types in another file location as the templates and then import them. Another feature increasing modularity is the relationship mechanism, that allows a custom wiring for each type usage. Another mechanism that Cloudify shares with TOSCA is the inheritance of types. This allows the user to inherit from parent types, meaning that defined elements of the parent type are also defined in the child type. Finally, the input mechanism allows defining parameterised models. Cloudify does not offer a marketplace.

Apache Stratos Support: Since Stratos defines its configurations and applications in JSON, they can be shared as any text file. Yet, the cartridges contain references to IDs of IaaS snapshots and hardware configurations. Thus the reusability is limited to a cloud. Moreover, the definitions of an already deployed application can't be changed dynamically; it needs to be un-deployed first and then edit its definition. Similarly, Puppet is built to be reusable and shareable also. Its marketplace *Puppet Forge* contains more than $3,300$ modules, which can be added to a Stratos setup. The reusability of Puppet modules is gained by dependencies between modules, which allows splitting work in smaller but linked modules.

8) Containerisation Feature: The use of containers such like Docker is a reasonable approach for sharing a virtual machine between several component instances, while keeping them isolated. This leads to better utilisation of the virtual machine [7]. Moreover, the increased isolation offered by containers allows resource consumption to be configured, controlled, and limited on the level of component instances. This feature does not consider whether cloud providers use containers instead of hypervisors, as this is transparent for the users of the platform.

Apache Brooklyn Support: Brooklyn does not support containers out of the box. Yet, the separate project Clocker enables the usage of Docker.

Cloudify Support: Cloudify supports containerisation using Docker. The user can use a docker container node type what allows starting a docker container from a provided Docker image. The Docker container, then can be deployed on a virtual machine node using a *contained_in* relationship.

Apache Stratos Support: Stratos supports containerisation with Docker by using Google Kubernetes as a cluster orchestration framework.

9) Windows Application Support Feature: While Linux is dominating the cloud computing environment, there are many professional companies running their applications on a Windows operating system [8]. Hence, these applications should be supported by cloud management tools.

Apache Brooklyn Support: Brooklyn relies heavily on SSH which excludes native Windows support. Windows support is currently under development, though.

Cloudify Support: Cloudify supports Windows but requires that the virtual machine (image)s have WinRM enabled. Cloudify uses this protocol to install its agents on the machine. The agents then operate in an operating system independent manner.

Apache Stratos Support: Stratos cartridges for deploying applications also support Windows, e.g. for .NET framework applications. Both the Stratos agent running in the applications' virtual machines and Puppet support the Windows operating system.

V. Comparison Result

Table I depicts the achievements of the three cloud tools with respect to the features defined in Section IV. To be able to account for different partial achievement or different achievement quality we use a three-staged marker.

VI. Related Work

To the best of our knowledge there is currently no directly comparable work defining an in-detail comparison framework based on features and applying it to the given tools. The work of [9] and [10] present a general view of cloud computing defining characteristics, features and challenges of the cloud computing environment on a much higher level, from which our features are derived. There is also multiple work defining a taxonomy and doing a comparison of cloud computing systems [11] [12]. However, those comparisons focus on cloud computing in general. They hence put stress on features offered by and comparison of cloud providers. A qualitative and quantitive survey for the two IaaS management tools Openstack and Synnefo[5] is provided by [13]. [14] depicts an elaborate overview of frameworks, projects and libraries with respect to provisioning, deployment and adaptation of cloud systems, but also stays at a much coarser granularity. [15] compares multiple cloud brokerage solutions by first categorising them, and then listing their core capabilities. However, their comparison is also done on a much higher level. Finally, Paasify[6] gives a good overview of existing PaaS and PaaS-like offers doing a feature comparison on higher, quantifiable levels.

VII. Conclusion

In this paper, we have considered basic requirements of cloud orchestration tools and derived a fine-grained list of features any of the tools shall support. We also applied these results by rating three publicly available cloud orchestration tools based on our list: Apache Brooklyn, Cloudify, and Apache Stratos.

[5]https://www.synnefo.org/

[6]http://www.paasify.it

TABLE I. COMPARISON OF BROOKLYN, CLOUDIFY, AND STRATOS.

Features	Tools		
	Brooklyn	Cloudify (Pro)	Stratos
Cloud Features			
Multi-Cloud			
# of Cloud Providers	jclouds	3 (5)	jclouds
Abstraction Layer	✓	✗	**0**
Cross-Cloud	✓	✓	✓
External PaaS	✗	✗	✗
Cloud Standards	✗	✗	✗
BYON	✓	✓	✗
Application Features			
Model Standards	**0**	**0**	✗
Resource Selection			
Manual Binding	✓	✓	✓
Automatic Binding	**0**	✗	✗
Optimised Binding	✗	✗	✗
Dynamic Binding	✗	✗	✗
Life Cycle Description			
Shell Script	✓	✓	✗
# DevOps Tools	1	3	1
Wiring & Workflow			
Attribute & Event Passing	**0**	✓	✓
Manual Workflow	✗	✓	✓
Automatic Workflow	✗	✓	✗
External Services	✗	✗	✗
Monitoring			
System Metrics	✗	✓	✓
Application Metrics	✗	✓	✓
Custom Metrics	✓	✓	✗
Aggregation	**0**	✓	✓
Dashboard	**0**	✗(✓)	✗
Historical Data	✗	✗(✓)	✗
Runtime Adaptation			
Thresholds	✓	✓	✓
Rule Engine	✗	✓	✓
Horizontal Scaling	✓	✓	✓
Repair	**0**	✓	✓
Custom Action	✗	✓	✗
Continuous Delivery	**0**	✗	✗
Reusability and Sharing of Models			
Reusability	**0**	✓	**0**
Sharing	✓	✗	**0**
Containerisation	✗	✓	✓
Windows Support	✗	✓	✓

✗= not fulfilled, **0**= partially fulfilled, ✓= fully fulfilled

When looking at the results it becomes evident, that especially the resource selection feature is underrepresented in all three tools. All tools require the user to manually bind a concrete resource to the components at description time causing vendor lock-in due to missing abstraction and non-optimal placement due to an incorrect initial selection, performance unpredictability and no possibility to change it at runtime. Our own cloud orchestration tool CLOUDIATOR[7] [16], [17] has the goal to close this gap.

Future work will include the finalisation of a first release of our CLOUDIATOR tool based on the experiences gained while working with Brooklyn, Cloudify, and Stratos. In parallel, we will evaluate and rate more tools and we plan to extend this evaluation to ongoing research projects, also considering non-function features like performance, availability or security.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 317715 (PaaSage) and 610711 (CACTOS), and from the European Community's Framework Programme for Research and Innovation HORIZON 2020 (ICT-07-2014) under grant agreement number 644690 (CloudSocket).

[7]https://github.com/cloudiator

REFERENCES

[1] S. Kächele, C. Spann, F. J. Hauck, and J. Domaschka, "Beyond IaaS and PaaS: An extended cloud taxonomy for computation, storage and networking," in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, ser. UCC '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 75–82.

[2] P. M. Mell and T. Grance, "SP 800-145. the NIST definition of cloud computing," National Institute of Standards & Technology, Gaithersburg, MD, United States, Tech. Rep., 2011.

[3] DMTF, "Cloud infrastructure management interface (CIMI) model and RESTful HTTP-based protocol," 2013.

[4] Open Grid Forum, "Open cloud computing interface - core," 2011.

[5] D. Palma and T. Spatzier, "Topology and orchestration specification for cloud applications version 1.0," OASIS Standard, 2013.

[6] J. Durand, A. Otto, G. Pilz, and T. Rutt, "Cloud application management for platforms version 1.1," OASIS Committee Specification, 2014.

[7] K. Razavi, A. Ion, G. Tato, K. Jeong, R. Figueiredo, G. Pierre, and T. Kielmann, "Kangaroo: A tenant-centric software-defined cloud infrastructure," in *Proceedings of the IEEE International Conference on Cloud Engineering*, Tempe, AZ, USA, United States, 2015.

[8] Linux Foundation, "Enterprise end user trends report," 2014.

[9] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[10] R. Buyya, "Market-oriented cloud computing: Vision, hype, and reality of delivering computing as the 5th utility," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, ser. CCGRID '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–.

[11] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC*, ser. NCM '09. IEEE Computer Society, 2009, pp. 44–51.

[12] C. Höfer and G. Karagiannis, "Cloud computing services: taxonomy and comparison," *Journal of Internet Services and Applications*, vol. 2, pp. 81–94, 2011.

[13] E. Qevani, M. Panagopoulou, C. Stampoltas, A. Tsitsipas, D. Kyriazis, and M. Themistocleous, "What can OpenStack adopt from a Ganeti-based open-source IaaS?" in *2014 IEEE 7th International Conference on Cloud Computing, Anchorage, AK, USA, June 27 - July 2, 2014*, 2014, pp. 833–840.

[14] N. Ferry, A. Rossini, F. Chauvel, B. Morin, and A. Solberg, "Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems," in *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, 2013, pp. 887–894.

[15] F. Fowley, C. Pahl, and L. Zhang, "A comparison framework and review of service brokerage solutions for cloud architectures," in *Service-Oriented Computing ICSOC 2013 Workshops*. Springer International Publishing, 2014, pp. 137–149.

[16] D. Baur, S. Wesner, and J. Domaschka, "Towards a model-based execution-ware for deploying multi-cloud applications," in *Advances in Service-Oriented and Cloud Computing*, ser. Communications in Computer and Information Science, G. Ortiz and C. Tran, Eds. Springer International Publishing, 2015, vol. 508, pp. 124–138.

[17] J. Domaschka, D. Baur, D. Seybold, and F. Griesinger, "Cloudiator: A cross-cloud, multi-tenant deployment and runtime engine," in *9th Workshop and Summer School On Service-Oriented Computing 2015*, 2015, in press.

Chapter 10

[core3] Is Distributed Database Evaluation Cloud-Ready?

This article is published as follows:

Material from: Daniel Seybold and Jörg Domaschka, "Is Distributed Database Evaluation Cloud-Ready?", *European Conference on Advances in Databases and Information Systems (ADBIS) - New Trends in Databases and Information Systems (Short Papers)*, published 2017, Springer International Publishing, DOI: https://doi.org/10.1007/978-3-319-67162-8_12

Is Distributed Database Evaluation Cloud-Ready?

Daniel Seybold(✉) and Jörg Domaschka

Institute of Information Resource Management, Ulm University, Ulm, Germany
{daniel.seybold,joerg.domaschka}@uni-ulm.de

Abstract. The database landscape has significantly evolved over the last decade as cloud computing enables to run distributed databases on virtually unlimited cloud resources. Hence, the already non-trivial task of selecting and deploying a distributed database system becomes more challenging. Database evaluation frameworks aim at easing this task by guiding the database selection and deployment decision. The evaluation of databases has evolved as well by moving the evaluation focus from performance to distribution aspects such as scalability and elasticity. This paper presents a cloud-centric analysis of distributed database evaluation frameworks based on evaluation tiers and framework requirements. It analysis eight well adopted evaluation frameworks. The results point out that the evaluation tiers performance, scalability, elasticity and consistency are well supported, in contrast to resource selection and availability. Further, the analysed frameworks do not support cloud-centric requirements but support classic evaluation requirements.

Keywords: NoSQL · Distributed database · Database evaluation · Cloud

1 Introduction

Relational database management systems (RDBMS) have been the common choice for persisting data for many decades. Yet, the database landscape has changed over the last decade and a plethora of new database management systems (DBMS) have evolved, namely NoSQL [20] and NewSQL [15]. These are promising persistence solutions not only for Web applications, but also for new domains such as "BigData" and "IoT". While NewSQL database systems are inspired by the relational storage model, the storage models of NoSQL database system can be further classified into key-value stores, document-oriented stores, column-oriented stores and graph-oriented stores [20]. NoSQL and NewSQL DBMS are designed to satisfy requirements such as high performance or scalability by running on commodity hardware as a distributed database management system (DDBMS), providing a single DBMS, which is spread over multiple nodes. An element of the overall DDBMS is termed *database node*.

An enabler of the DBMS evolvement is cloud computing by providing fast access to commodity hardware via elastically, on-demand, self-service resource

M. Kirikova et al. (Eds.): ADBIS 2017, CCIS 767, pp. 100–108, 2017.
DOI: 10.1007/978-3-319-67162-8_12

provisioning [17]. Infrastructure as a Service (IaaS) is the preferable way to deploy a DDBMS, requiring a high degree of flexibility in compute, storage and network resources [17].

With the number of available NoSQL and NewSQL systems and cloud resource offerings, the database selection and deployment on the cloud is a challenging task. Hence, DBMS evaluation is a common approach to guide these decisions. With the evolvement of the DBMSs, the landscape of database evaluation frameworks (DB-EFs) has evolved as well: from single node evaluation, *e.g.* TPC-E [1] of the Transaction Processing Performance Council (TPC), to DDBMS evaluation, *e.g.* the Yahoo Cloud Serving Benchmark (YCSB) [7], adding new evaluation tiers such as scalability, elasticity or consistency. Yet, these DB-EFs aim at different evaluation tiers and differ towards common DB-EF requirements [6,14], especially with respect to cloud computing.

In order to facilitate the selection and deployment of DDBMS in the cloud, we present an analysis of DB-EF with the focus on exploiting cloud computing. Our contribution is threefold by (1) defining relevant evaluation tiers for DDBMS deployed in the cloud; (2) extending existing requirements towards DDBMS evaluation with cloud specific requirements; (3) analyse existing evaluation frameworks based on the evaluation tiers and evaluation requirements.

The remainder is structured as follows: Sect. 2 introduces the background on DBMS evaluation. Section 3 defines the evaluation tiers while Sect. 4 defines the requirements towards evaluation frameworks. Section 5 analysis and discusses existing frameworks. Section 6 concludes.

2 Background

Evaluating DBMS imposes challenges for the evaluation frameworks itself, which have been discussed over decades. A first, but still valid guideline for evaluating RDBMS defines the requirements such as relevance to an application domain, portability to allow benchmarking of different systems, scalability to support benchmarking large systems, and simplicity to ensure that the results are easy to understand [14]. A more recent guideline adds the DDBMS and the resulting challenges for supporting different deployment topologies and coordination of distributed experiments [6]. By adopting these challenges, several DB-EFs have been established over the years, which are analysed in Sect. 5.

An overview of existing DB-EF focuses on the tiers availability and consistency. Yet, general requirements for DB-EF are not introduced and cloud specific characteristics are not considered [11]. An overview of DB-EFs for NoSQL database systems is provided by [19]. Yet, the focus lies on evaluating the data model capabilities, without taking explicitly into accout DDBMS aspects and the usage of cloud resources. Evaluating the dimensions of consistency in DDBMS, is also analysed by [5], introducing client-centric and data-centric consistency metrics. Related DB-EF for consistency are presented and missing features for fine-grained consistency evaluation are outlined. An overview of DB-EF is included

[1] http://www.tpc.org/tpce/.

in a recommendation compendiums [16] for distributed database selection based on functional and non-functional requirement Yet, the compendium considers only the performance evaluation.

3 Distributed Database Evaluation Tiers

With the evolving heterogeneity in DDBMSs their evaluation becomes even more challenging. DBMS evaluation is driven by **workload domains (WD)**. On a high level WDs can classified into *transactional* (TW) [9], *web-oriented* (WOW) [9], *Big Data* (BDW) [12] and *synthetic* workloads (SW) [7]. These WDs drive the need for considering various evaluation tiers, which are distilled out of database and cloud research.

Resource selection (RS) determines the best matching resources to run a DBMS. For traditional RDBMSs the focus lies on single node resources, (CPU, memory, storage). For DDBMSs network, locality and number of nodes became important factors. By using cloud resources for a DBMS, the cloud providers tend to offer more *heterogeneous resources* such as VMs with dedicated storage architectures[2], container based resources[3], or dedicated resource locations from data center to physical host level.

Performance (P) evaluates the behaviour of a DBMS against a specific kind of workload. Performance metrics are *throughput* and *latency*, which are measured by the evaluation framework.

Scalability (S) defines the capability to process arbitrary workload sizes by adapting the DBMS by scaling *vertically* (scale-up/down) or *horizontally* (scale-in/out) [1]. Scaling vertically changes the computing resources of a single node. Horizontal scaling adds nodes to a DDBMS cluster (scale-out) or removes nodes (scale-in) In the following the term scalability implies horizontal scalability Measuring scalability is performed by correlating throughput and latency for growing cluster sizes and workloads. A high scalability rating is represented by constant latency and proportionally growing throughput with respect to the number of nodes and the workload size [7].

Elasticity (E) defines the ability to cope with sudden workload fluctuations without service disruption [1]. Elasticity metrics are *speedup* and *scaleup* [7]. Speedup refers to the required time for a scaling action, *i.e.* adapting the cluster size, redistributing data and stabilising the cluster. Scaleup refers to the benefit of this action, *i.e.* the throughput/latency development with respect to the workload fluctuation.

Availability (A) represents the degree to which a DBMS is operational and accessible when required for use. The availability of a DBMS can be affected by *overload* (issuing more requests in parallel than theDBMS can handle or *failures on the resource layer* (a node failure). With respect to failures, DDBMSs apply replication of data to multiple database nodes. A common

[2] https://aws.amazon.com/de/ec2/instance-types/.
[3] https://wiki.openstack.org/wiki/Magnum.

metric to measure availability with respect to node failures are the takeover time, and the performance impact.

Consistency (C) Distributed databases offer different consistency guarantees as there is trade-off between consistency, availability and partitioning, *i.e.* the CAP theorem [13]. Consistency can be evaluated *client-centric* (*i.e.* from the application developer perspective) and *data-centric* (*i.e.* from the database administrator perspective) [5] . Here, we only consider client-centric consistency that can be classified into *staleness* and *ordering* [5]. Staleness defines how much a replica lags behind its master. It is measured either in time or versions. Ordering defines all requests must be executed on all replicas in the same chronological order.

4 Evaluation Frameworks Requirements

Besides the evaluation tiers, DBMS evaluation imposes requirements towards the DB-EF itself. We briefly present established requirements [6,14] as well as novel cloud-centric requirements.

Usability (U) eases the framework configuration, execution and extension by providing sufficient documentation and tools to run the evaluation. Hence, the evaluation process has to be transparent to provide objective results [14].

Distribution/Scalability (D/S) is provided by distributed workload generation, *i.e.* the framwork clients can be distributed across multiple nodes in order to increasing the workload by utilising an arbitrary amount of clients [6].

Measurements Processing (MP) defines that measurements are gathered not only in an aggregated but also in a fine-grained manner for further processing [6]. As the amount of measurements can grow rapidly for multiple or long running evaluation runs, file-based persistence might not be sufficient. Hence, advanced persistence options such as time series databases (TSDBS), will ease the dedicated processing and visualisation.

Monitoring (MO) data improves the significance of evaluation results. Hence, monitoring of the involved resources, clients ,and DBMSs should be supported by the evaluation framework to provide the basis of a thorough analysis. Again an advanced persistence solution is beneficial.

Database Abstraction (DA) enables the support of multiple DBMSs by abstracting database driver implementations. Yet, the abstraction degree needs to be carefully chosen as a too high abstraction might limit specific DBMS features and distort results. Therefore, the abstraction interface should be aligned with the specified workload scenarios [6].

Client Orchestration (CO) enables automated evaluation runs. Therefore, the framework should provide tools that orchestrate evaluations, *i.e.* provision (cloud) resources, create, execute and collect the results and clean-up the clients. Hence, CO eases the creation of arbitrary load patterns and the simulation of multi-tenant workload patterns.

Database Orchestration (DO) enables the management of the DDBMSs to facilitate repetitive evaluation for different resources, configurations and the adaptation of the DDBMS based on predefined conditions. Hence, the evaluation framework should provide tools to automatically orchestrate DDBMSs, *i.e.* provision resources, setup, configure and adapt generic DDBMSs.

Multi-phase Workloads (MpW) define the support of multiple workloads that run in parallel This is crucial to execute advanced evaluation scenarios. Further, the specification of the load development over a certain time frame per workload is required to simulate real world scenarios.

Extensibility (E) defines the need to provide an architecture, which eases the extension of the framework capabilities, *e.g.* by adding support for additional DBMSs or workload types.

5 Analysis of Evaluation Frameworks

In this section we analyse DB-EFs, which focus on DDBMSs. Hereby, we consider only DB-EFs, which have been published within the evolvement of DDBMSs, *i.e.* from 2007 on. In addition, we only consider the original evaluation frameworks and no minor extensions or evaluations based on these frameworks.

First, we analyse each framework based on the workload domain and supported evaluation tiers. The results are shown in Table 1. Second, we analyse each frameworks capabilities against the presented DB-EF requirements from Sect. 4. The results are shown in Table 2. The analysis applies ✗= *not supported*, (✓)= *partially supported*, ✓= *supported*. A detailed analysis can be found in an accompanying technical report [21].

Table 1. Distributed database evaluation tiers

Evaluation framework	Evaluation tier						
	WD	RS	P	S	E	A	C
TPC-E	TW	(✓)	✓	✓	✗	✗	✗
YCSB [7]	SW	✗	✓	✓	✓	✗	✗
YCSB++ [18]	TW, BDW, SW	✗	✓	✓	✓	✗	✓
BG [3]	WOW	✗	✓	✓	✗	✗	✓
BigBench [12]	TW, BDW	✗	✓	✗	✗	✗	✗
OLTP-bench [9]	WOW, SW	✗	✓	✓	✗	✗	✗
YCSB-T [8]	TW, SW	✗	✓	✓	✓	✗	✓
LinkBench [2]	WOW	✗	✓	✗	✗	✗	✗

The first insight of our analysis is that performance, scalability, elasticity and consistency tiers are well covered, but the resource selection and availability tier lack support (*cf.* Sect. 5.2). The second insight points out that the traditional DB-EF requirements [14] such as usability, distribution and extensibility are well supported, while monitoring and cloud orchestration are not (*cf.* Sect. 5.2).

Table 2. Distributed database evaluation tiers

Evaluation framework	Evaluation framework requirement								
	U	D/S	MP	MO	DA	CO	DO	MpW	E
TPC-E	✓	✓	(✓)	✗	(✓)	✗	✗	✓	✓
YCSB [7]	✓	✓	(✓)	✗	✓	✗	✗	✗	✓
YCSB++ [18]	(✓)	✓	✓	✓	(✓)	✓	✗	✓	✗
BG [3]	✓	✓	✓	✓	✓	✗	✗	✓	✓
BigBench [12]	✓	✓	✗	✗	(✓)	✗	✗	✗	✓
OLTP-bench [9]	(✓)	✓	(✓)	✓	(✓)	✓	✗	✓	✓
YCSB+T [8]	(✓)	✓	(✓)	✗	✓	✗	✗	✗	✓
LinkBench [2]	✓	(✓)	(✓)	✓	(✓)	✗	✗	✓	✓

5.1 Results for Evaluation Tiers

The resulting table (*cf.* Table 1) shows that the early DB-EFs focus on performance, scalability and elasticity, while newer frameworks focus as well on consistency. Hereby, only the performance tier has established common rating indices such as throughput, latency or SLA based rating [3]. While multiple frameworks target the scalability and elasticity tier, a common methodology and rating index has not yet been established. Yet, the need for a common evaluation methodology [22] and rating index [10] is already carved out.

Currently not supported evaluation tiers are resource selection and availability. While resource selection is partially supported by TPC-E, which considers physical hardware configurations, cloud-centric resource selection is not in the scope of any of the frameworks. With the increasing heterogeneity of cloud resources from diverse virtual machines offering to even container based resources, the consideration of cloud-centric resource selection needs to move into the focus of novel DB-EFs. Yet, existing DB-EFs can be applied to evaluate DDBMS running on heterogeneous cloud resources, but the DB-EFs do not offer an explicit integration with cloud resource offerings. Hence, manual resource management, monitoring and client/DDBMS orchestration hinders cloud-centric evaluations.

As availability is a major feature of DDBMS it is surprising that it is not considered by the analysed DB-EFs. Especially, as cloud resources do fail, availability concepts for applications running on cloud resources is widely discussed topic. Again, the support of DDBMSs orchestration can enable database specific availability evaluations.

5.2 Results for Evaluation Framework Requirements

The analysis of the evaluation framework requirements (*cf.* Table 2) shows that usability, scalability, database abstraction and extensibility are covered by all

frameworks. Measurement processing is covered as well but only a few frameworks support advanced features such as visualisation and none of the frameworks supports advanced storage solutions such as TSDBs. Multi-phase workloads are partially covered by the frameworks, especially by the frameworks from the TW and WOW domains. The monitoring of client resources is partially covered, but only OLTP-bench considers resource monitoring. While all frameworks support the distributed execution of evaluations, only two support the orchestration of clients, which complicates the distributed evaluation runs. Further, none of the frameworks supports DBMS orchestration. This fact leads to high complexity only for setting up the evaluation environment, especially when it comes to heterogeneous cloud resources. Further, dynamic DDBMS transitions for evaluating tiers such as elasticity or availability, always require custom implementations, which impedes the comparability and validity of the results.

6 Conclusion and Future Work

In the last decade the landscape of distributed database systems has evolved and NoSQL and NewSQL database systems appeared. In parallel, cloud computing enabled novel deployment option for database systems. Yet, these evolvements raise the complexity in selecting and deploying an appropriate database system.

In order to ease such decisions, several evaluation frameworks for distributed databases have been developed. In this paper, we presented an analysis of distributed database evaluation frameworks based on evaluation tiers and requirements towards the frameworks itself. The analysis is applied to eight evaluation frameworks and provides a thorough analysis of their evaluation tiers and capabilities. The results of this analysis shows that the performance, scalability, elasticity, and consistency tiers are well covered, while resource selection and availability are not considered by existing evaluation frameworks. With respect to the framework requirements, traditional requirements are covered [14], while cloud-centric requirements such as orchestration are only partially supported.

The analysis shows, that existing frameworks can be applied to evaluate distributed databases in the cloud, but there are still unresolved issues on the evaluation tier side, *i.e.* the support for resource selection and availability evaluation, and on the framework requirement side, *i.e.* the orchestration of clients and databases and exploitation of advanced storage solutions. This hinders repeatability [14] of evaluations on heterogeneous cloud resources as well as dynamic transition in the cluster. Yet, cloud computing research already offers approaches to enable automated resource provisioning and application orchestration in the cloud based on *Cloud Orchestration Tools (COTs)* [4]. Integrating COT into evaluation frameworks can be an option to ease the distributed execution of evaluation runs as well as orchestrating database clusters across different cloud resources. As COTs provide monitoring and adaptation capabilities, they can ease the evaluation of dynamic cluster transitions by defining advanced evaluation scenarios with dynamic database cluster adaptations.

Future work will comprise the analysis of COTs with respect to their exploitation in database evaluation frameworks. In addition, the design and implementation of a cloud-centric database evaluation framework is ongoing.

Acknowledgements. The research leading to these results has received funding from the EC's Framework Programme HORIZON 2020 under grant agreement number 644690 (CloudSocket) and 731664 (MELODIC). We thank Moritz Keppler and the Daimler TSS for their valuable and constructive discussions.

References

1. Agrawal, D., Abbadi, A., Das, S., Elmore, A.J.: Database scalability, elasticity, and autonomy in the cloud. In: Yu, J.X., Kim, M.H., Unland, R. (eds.) DASFAA 2011. LNCS, vol. 6587, pp. 2–15. Springer, Heidelberg (2011). doi:10.1007/978-3-642-20149-3_2
2. Armstrong, T.G., Ponnekanti, V., Borthakur, D., Callaghan, M.: Linkbench: a database benchmark based on the facebook social graph. In: SIGMOD (2013)
3. Barahmand, S., Ghandeharizadeh, S.: Bg: A benchmark to evaluate interactive social networking actions. In: CIDR (2013)
4. Baur, D., Seybold, D., Griesinger, F., Tsitsipas, A., Hauser, C.B., Domaschka, J.: Cloud orchestration features: Are tools fit for purpose? In: UCC (2015)
5. Bermbach, D., Kuhlenkamp, J.: Consistency in distributed storage systems. In: Gramoli, V., Guerraoui, R. (eds.) NETYS 2013. LNCS, vol. 7853, pp. 175–189. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40148-0_13
6. Bermbach, D., Kuhlenkamp, J., Dey, A., Sakr, S., Nambiar, R.: Towards an extensible middleware for database benchmarking. In: Nambiar, R., Poess, M. (eds.) Performance Characterization and Benchmarking: Traditional to Big Data. LNCS, pp. 82–96. Springer, Cham (2015). doi:10.1007/978-3-319-15350-6_6
7. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking cloud serving systems with ycsb. In: SoCC (2010)
8. Dey, A., Fekete, A., Nambiar, R., Rohm, U.: Ycsb+t: Benchmarking web-scale transactional databases. In: ICDEW (2014)
9. Difallah, D.E., Pavlo, A., Curino, C., Cudre-Mauroux, P.: Oltp-bench: An extensible testbed for benchmarking relational databases. VLDB **7**, 277–288 (2013)
10. Dory, T., Mejias, B., Roy, P., Tran, N.L.: Measuring elasticity for cloud databases. In: Cloud Computing (2011)
11. Friedrich, S., Wingerath, W., Gessert, F., Ritter, N., Pldereder, E., Grunske, L., Schneider, E., Ull, D.: Nosql oltp benchmarking: A survey. In: GI-Jahrestagung (2014)
12. Ghazal, A., Rabl, T., Hu, M., Raab, F., Poess, M., Crolotte, A., Jacobsen, H.A.: Bigbench: towards an industry standard benchmark for big data analytics. In: SIGMOD (2013)
13. Gilbert, S., Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM Sigact News **33**, 51–59 (2002)
14. Gray, J.: Benchmark Handbook: For Database and Transaction Processing Systems. Morgan Kaufmann Publishers Inc, San Francisco (1993)
15. Grolinger, K., Higashino, W.A., Tiwari, A., Capretz, M.A.: Data management in cloud environments: Nosql and newsql data stores. JoCCASA **2**, 22 (2013)

16. Khazaei, H., Fokaefs, M., Zareian, S., Beigi-Mohammadi, N., Ramprasad, B., Shtern, M., Gaikwad, P., Litoiu, M.: How do i choose the right NoSQL solution? a comprehensive theoretical and experimental survey. BDIA **2**, 1 (2016)
17. Mell, P., Grance, T.: The nist definition of cloud computing. Technical report, National Institute of Standards & Technology (2011)
18. Patil, S., Polte, M., Ren, K., Tantisiriroj, W., Xiao, L., López, J., Gibson, G., Fuchs, A., Rinaldi, B.: Ycsb++: benchmarking and performance debugging advanced features in scalable table stores. In: SoCC (2011)
19. Reniers, V., Van Landuyt, D., Rafique, A., Joosen, W.: On the state of nosql benchmarks. In: ICPE (2017)
20. Sadalage, P.J., Fowler, M.: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Pearson Education, London (2012)
21. Seybold, D., Domaschka, J.: A cloud-centric survey on distributed database evaluation. Technical report. Ulm University (2017)
22. Seybold, D., Wagner, N., Erb, B., Domaschka, J.: Is elasticity of scalable databases a myth? In: IEEE Big Data (2016)

Chapter 11

[core4] Gibbon: An Availability Evaluation Framework for Distributed Databases

This article is published as follows:

Material from: Daniel Seybold, Christopher B. Hauser, Simon Volpert, and Jörg Domaschka, "Gibbon: An Availability Evaluation Framework for Distributed Databases", *On the Move to Meaningful Internet Systems (OTM)*, published 2017, Springer International Publishing, DOI: `https://doi.org/10.1007/978-3-319-69459-7_3`

Gibbon: An Availability Evaluation Framework for Distributed Databases

Daniel Seybold(✉), Christopher B. Hauser, Simon Volpert, and Jörg Domaschka

Institute of Information Resource Management, Ulm University, Ulm, Germany
{daniel.seybold,christopher.hauser,simon.volpert,
joerg.domaschka}@uni-ulm.de

Abstract. Driven by new application domains, the database management systems (DBMSs) landscape has significantly evolved from single node DBMS to distributed database management systems (DDBMSs). In parallel, cloud computing became the preferred solution to run distributed applications. Hence, modern DDBMSs are designed to run in the cloud. Yet, in distributed systems the probability of failures is the higher the more entities are involved and by using cloud resources the probability of failures increases even more. Therefore, DDBMSs apply data replication across multiple nodes to provide high availability. Yet, high availability limits consistency or partition tolerance as stated by the CAP theorem. As the decision for two of the three attributes in not binary, the heterogeneous landscape of DDBMSs gets even more complex when it comes to their high availability mechanisms. Hence, the selection of a high available DDBMS to run in the cloud becomes a very challenging task, as supportive evaluation frameworks are not yet available. In order to ease the selection and increase the trust in running DDBMSs in the cloud, we present the Gibbon framework, a novel availability evaluation framework for DDBMSs. Gibbon defines quantifiable availability metrics, a customisable evaluation methodology and a novel evaluation framework architecture. Gibbon is discussed by an availability evaluation of MongoDB, analysing the take over and recovery time.

Keywords: Distributed database · Database evaluation · High availability · NoSQL · Cloud

1 Introduction

The landscape of database management systems (DBMSs) has evolved significantly over the last decade, especially when it comes to large-scale DBMSs installations. While relational database management systems (RDBMS) have been the common choice for persisting data over decades, the raise of the Web and new application domains such as *Big Data* and *Internet of Things (IoT)* drive the need for novel DBMS approaches. NoSQL and only recently NewSQL

H. Panetto et al. (Eds.): OTM 2017 Conferences, Part II, LNCS 10574, pp. 31–49, 2017.
https://doi.org/10.1007/978-3-319-69459-7_3

DBMSs [15] have evolved. Such DBMSs are often designed as a distributed database management system (DDBMS) spread out over multiple *database nodes* and supposed to run on commodity or even virtualised hardware.

Due to their distributed architecture, DDBMSs support horizontal scalability and consequently the dynamic allocation and usage of compute, storage and network resources [1] based on the actual demand. This fact makes Infrastructure as a Service (IaaS) cloud platforms a suited choice for running DDBMSs, as it provides elastically, on-demand, self-service resource provisioning [19].

Even though distributed architectures can be used to improve the availability of the overall system, this is not automatically the case. In particular, in distributed systems the probability of failures is the higher the more entities are involved, *i.e.* in the case of DDBMSs the more data nodes are used. When cloud resources are used, the situation is worsened, as failures on lower level may affect multiple virtualised resources and cause mass failures [13,16].

With respect to DDBMSs, the common approach to availability is to replicate data items across multiple database nodes to ensure high availability in case of resource failures. However, the desire for availability is hindered by the CAP theorem stating that availability is achieved by scarifying consistency guarantees or partition tolerance [6]. However, the choice between two of these three attributes is not a binary decision and offers multiple trade-offs [5]. This has led to a very heterogeneous DDBMS landscape not only with respect to the sheer number of systems, but also the availability mechanisms they provide [11].

Hence, we find ourselves in the situation that more DDBMSs exist than ever, each of them promising availability features and often enough even high availability. Further, many of these DDBMS are operated on IaaS infrastructures with an increased risk of mass failures. Yet, for none of the DDBMSs it is known how it actually behaves under failure conditions and how the failure condition affects the availability of the DDBMSs. At the same time, no supportive frameworks for evaluating availability of DDBMSs exist [24] and in consequence, selecting a DDBMS becomes a gamble for users if availability is a major selection criterion.

In order to increase the trust in running DDBMSs on cloud resources and easing the selection of high available DDBMSs, we present Gibbon, a novel framework for evaluating the availability of DDBMSs. It explicitly supports cloud failure scenarios. Hence, our contribution is threefold: (1) we identify quantifiable metrics to evaluate availability; (2) we define an extensible evaluation methodology; (3) we present a novel availability evaluation framework architecture.

The remainder is structured as follows: Sect. 2 introduces the background on availability, DDBMS and cloud computing, while Sect. 3 analyses the impact of failures. Section 4 defines the availability metrics while Sect. 5 presents the evaluation methodology. Section 6 presents the framework architecture. Section 7 discusses the presented framework and Sect. 8 presents related work. Section 9 concludes.

2 Background

In order to consolidate the context of the Gibbon framework, we introduce in this section the background on DDBMSs, availability, and DDBMSs on IaaS.

2.1 Distributed Database Systems

Per definition, a DBMS manages data items grouped in collections or databases. For DDBMS these data items are spread out over multiple database nodes each of which runs management logic. Hence, the databases nodes communicate and cooperate in order to realise the expected functionality.

The use of distribution has two conceptionally unrelated benefits: (i) more data can be stored and processed, as the overall available capacity is the sum of the capacity of the individual database nodes. In this usage scenario the *sharding (partitioning) strategy* defines which data items are stored on which of the database nodes. (ii) data items can be stored redundantly on multiple database nodes protecting them against failures of individual database nodes. A *replication degree* of n denotes that a data item is stored n times in the system.

Replication. When sharding is used without replication, no tolerance against node failures exists. On the other hand, using replication without sharding means that all data is available on all database nodes. This is also referred to as full replication. When sharding and replication is used in parallel, each database node will contain only a subset of all data items [9]. Depending on the *replication strategy* [22] a user is allowed to interact with only one of the replicas (master-slave replication) or all of them (multi-master).

In the master-slave approach one of the n physical copies of a data item has the master role. Only this item can be changed by users. It is the task of the database node hosting this item to synchronize slave nodes. The latter only execute read requests. A failure of the master will require the re-election of a new master amongst the remaining slaves. In the multi-master approach, any replica can be updated and the hosting database nodes have to coordinate changes.

Geo-replication caters for mirroring the entire DDBMS cluster to a different location in order to protect the data against catastrophic events.

Replica Consistency. Having multiple copies of the same data item in the system requires to keep the copies in sync with each other. This is particularly true for multi-master approaches. Here, two approaches exist: synchronous propagation ensures consistency is coordinated amongst all database nodes hosting a replica before any change is confirmed to a client. Asynchronous propagation in turn delays this so that multiple diverging copies of the item can exist in the system and clients can perform conflicting updates unnoticed.

Node and Task Types. Besides storing data items, a DDBMS has several other tasks to do: *management tasks* keep track of the location of data items, routing queries to the right destination, and detecting node failures. *Query tasks* process queries issued by the client and interact with the database nodes according to the

distribution information the management task stores. Depending on the actual DDBMS these tasks are executed by all database nodes in peer-to-peer manner, isolated in separate nodes, or even part of the database driver used by the client.

Storage Models. For DBMS three top-level categories are currently in use [15]: *relational*, *NoSQL* and *NewSQL* data stores. Relational data stores target transactional workloads, providing strong consistency guarantees based on the ACID paradigm [17]. Hence, relational data stores are originally designed as single server DBMS, which have been extended lately to support distribution (*e.g.* MySQL Cluster[1]). NoSQL storage models can be classified into key-value stores, document-oriented stores, column-oriented stores and graph-oriented stores. Compared to relational, their consistency guarantees are weaker and tend to towards BASE [21]. This weakening consistency makes those DDBMS better suited for distributed architectures and eases the realisation of features such as scalability and elasticity. NewSQL data stores are inspired by the relational data model and target strong consistency in conjunction with a distributed architecture.

2.2 Availability for DDBMSs

Generally, *availability* is defined as *the degree to which a system is operational and accessible when required for use* [14]. Besides *reliability* (the *"measure of the continuity of correct service"* [3]), availability is the main pillar of many fault-tolerant implementations [11].

In this paper, we solely focus on the availability aspect and assume that DDBMSs are reliable, *i.e.* work as specified. As our primary metric, we use what Zhong et al. call *expected service availability* and define availability of a DDBMS as the proportion of all successful requests [27] over all requests.

The availability of the DDBMS can be affected by two kinds of conditions [11]: (i) A high number of requests issued concurrently by clients, overloading the DDBMS such that the requests of clients cannot be handled at all or are handled with an unacceptable $latency > \Delta t$. (ii) Failures occur that impact network connectivity or availability of data items. The failure scenarios we consider are subject to Sect. 2.3.

2.3 Cloud Infrastructure

IaaS clouds have become a preferable way to run DDBMSs. Due to its cloud nature, IaaS offers more flexibility than bare metal resources. IaaS provides processing, storage, and network to run arbitrary software [19]. The processing and storage resources are typically encapsulated in a virtual machine (VM) entity that also includes the operating system (OS). VMs run on hypervisors on top of the physical infrastructure of the IaaS provider. As cloud providers typically operate multiple data centres, IaaS eases to span DDBMSs across different geographical locations. The location of cloud resources can be classified

[1] https://www.mysql.com/products/cluster/.

into geographical locations *(regions)*, data centres inside a region *(availability zones)*, *racks* inside a data centre and *physical hosts* inside a rack.

This set-up heavily influences availability as failures on different levels of that stack can have impact on individual database nodes (running in one VM), but also on larger sets of them. For instance, the failure of a hypervisor will lead to the failure of all VMs on that hypervisor.

An exemplary DDBMS deployment on IaaS is depicted in Fig. 1. Here, an 8-node DDBMS is deployed across two regions of one cloud provider. Each database node is placed on a VM and the VMs rely on different availability zones of the respective region. The example also illustrates the use of heterogeneous physical hardware: availability zones B and C are built upon physical servers without disks and dedicated storage servers. Availability zones A and D are built upon servers with built-in disks.

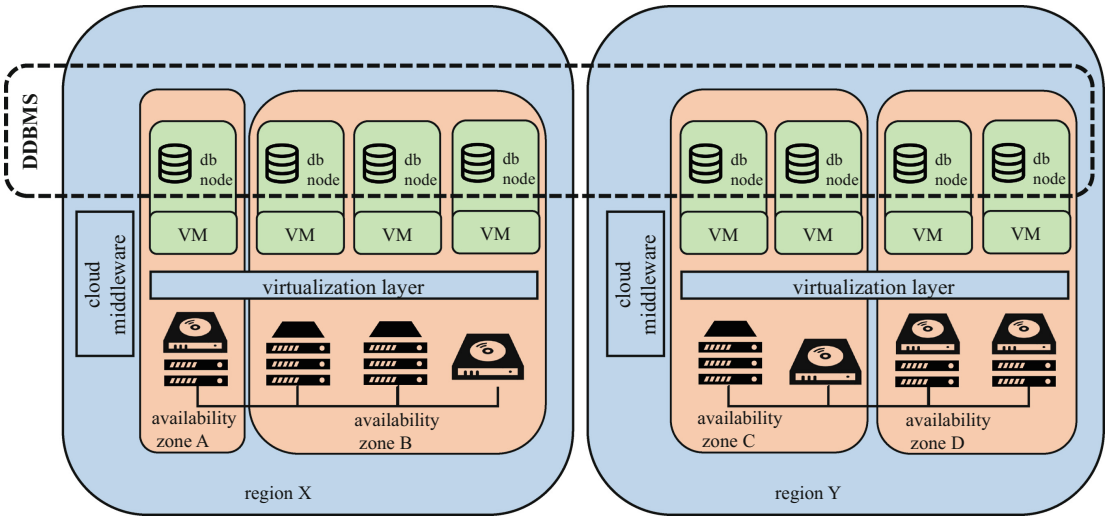


Fig. 1. DDBMS on IaaS

3 Failure Impact Analysis

Section 2.2 stated that two types of events impact the availability of DDBMSs: overload and failures. Dealing with overload has seen much attention in literature (*cf.* Sect. 8) so that this paper will focus on the latter that has barely received attention in the past [24]. In particular, our work addresses the capabilities of DDBMSs availability mechanisms to overcome failures in IaaS environments.

3.1 Replication for Availability

In Sect. 2.1, we introduced replication and partitioning as two major, but basically unrelated concepts used in DDBMSs. This section investigates the impact of their use under failure conditions.

No Replication. Apparently, when a database node fails and no replication is used, all data items stored on that database node become unavailable. The impact on the overall availability of the DDBMS depends on the access pattern of the failed shard, but for uniform distribution, the availability will drop to $\frac{N-1}{N}$ while this database node is unavailable.

Master-Slave Replication. When master-slave replication is used, the failure of a single database node has less impact, as copies of the data item are in the system. Yet, the process of detecting the failure and finding a new master for all data items from the failed database node needs time no matter if the new master is elected manually or automatically [11]. Hence, for uniform distribution of access, availability will still drop to $\frac{N-1}{N}$. Yet, hopefully for a shorter time.

Multi-master Replication. When using multi-master replication the failure of a single database node does not affect the overall availability, as any other database node hosting a replica of the requested data item can be contacted. Nevertheless, depending on the driver implementation and the routing, a small portion of requests may fail, when they are connected with the failed database node at the time of failure.

Functional Nodes. Some DDBMSs make use of additional hosts that function as entry points for clients or as a registry storing the mapping from data item to database node. The failure of these node also has impact on the overall availability. In particular, if configured wrong, the failure of any of those nodes can render the entire DDBMSs unavailable.

3.2 Failures and Recovery

This section investigates failures and recovery of DDBMSs hosted on Clouds. In particular, it derives how to emulate the failure of certain resources for the sake of evaluating availability metrics of different DDBMSs.

From Fig. 1 we can see that a cloud-operated DDBMS sits on top of several layers of hard- and software. Hence, even assuming that both DDBMS code and the operating system surrounding it are correct, the large stack leaves opportunities that can go wrong: On the infrastructure level, servers, network links, network device, power supplies, or cooling may fail. Similarly, on software level, management software and hypervisors can fail; algorithms, network, and devices can be buggy, misconfigured, or in the process of being restarted.

Any of these failures can affect virtual machines and virtual networks, but also physical servers, physical networks, entire racks, complete availability zones, or even entire data centers. Table 1 lists these failure levels with a way to emulate the failure and an action that helps to recovery from the failure.

The failure of a *database node* or a *virtual machine* can be represented as virtual machine unavailability and can be emulated by forcibly terminating the virtual machine. Here, it is important to ensure that no additional clean-up tasks, *e.g.* closing network connections get executed. The failure of a *physical server* leads to the unavailability of all virtual machines hosted on that

Table 1. DDBMS failures in a Cloud Infrastructure

Failure level	Emulate	Recovery
DDBMS node	Forcibly terminate VM	Replace VM
VM	Forcibly terminate VM	Replace VM
Physical server	Forcibly terminate all VMs on server	Replace VMs/move zone
Availability zone	Forcibly terminate all VMs in zone	Move zone or region
Region	Forcibly terminate all VMs in region	Move region/provider
Cloud provider	Forcibly terminate VMs hosted by provider	Move provider

server. The failure of a full *availability zone* or even an entire *region* leads to the unavailability of multiple physical servers and hence unavailability of all hosted virtual machines.

4 Availability Metrics

From the previous sections we derive input parameters and output metrics to evaluate the availability of DDBMSs running on cloud infrastructures. Input parameters describe the deployment and evaluation specifications of the DDBMS, while output metrics describe the experienced availability after the input parameters have been applied. Hence, a tuple of input parameters and output metrics provide the base for the availability evaluation (*cf.* Sect. 5).

4.1 Input Parameters

The input parameters as listed in Table 2 comprise the *deployment* and *evaluation* specification. The deployment specification combines the replication and partitioning characteristics (*cf.* Sect. 3.1), failure and recovery characteristics (*cf.* Sect. 3.2) and deployment information of the DDBMS nodes. The first two input parameters define the *replication* setting of the DDBMS, *i.e.* node replication level and cross data centre geo-replication level. None, one, or both replication levels might be configured for the DDBMS under observance. The replication first is defined by the strategy, defines the amount of replicas the replication will have in normal, healthy state, and the update laziness, how the write requests are synchronized between replicas.

Partitioning defines the setting for data partitioning, if present. If partitioning takes place, the amount of partitions are specified, and how the distribution strategy of data items to partitions is handled (group based, range based, with hashing functions). For accessing the distributed data items, the data access is of importance, namely if the client connects to the correct partition directly, via a proxy or requests are routed automatically.

Table 2. Input parameters

Input parameter		Description
Deployment	Node replication	Strategy (single-/multi-master, selection), replicas, laziness
	Geo-replication	Equals "node replication"
	Partitioning	Number of partitions, strategy (range, hash), data access (client, proxy, routing)
	Resources	Hierarchical infrastructure model of DDBMS (*cf.* Fig. 1)
Evaluation	Failure spec	Number of failing nodes per level (cf. Table 1), number of failing nodes per types
	Recovery Spec	Restart policies, number of database nodes to add (per node type if existent)
	Workload spec	Requests per second, read/write request ratio, number of data items

The *resources* parameter contains a full model of the allocated infrastructure resources for the DDBMS. This model includes all infrastructure entities involved from geographic location, to physical servers, virtual machines and DDBMS nodes (cf. Fig. 1). If the DDBMS differentiates node types, the type is reflected in the resource information as well.

Further, Table 2 also presents the *evaluation specification* parameters. The *failure specification* parameter defines a failure scenario which will be emulated by the Gibbon framework. For each level of potential failures described in Table 1, the amount of failing resource entities and (optionally) the DDBMS node type to fail is defined. The *recovery specification* parameter on the other hand describes the emulated recovery plan such as restarting virtual machines or adding new nodes to the DDBMS. The input parameters can skip the optional failure recovery parameter.

For simulating failure and recovery specification, the DDBMS is continuously under an artificial workload, defined by the *workload specification* parameter. This parameter defines the amount of read and write requests, as well as the total amount of data items stored in the DDBMS.

4.2 Output Metrics

The output metrics presented in Table 3 represent the experienced availability after the input parameters deployment and evaluation specification have been applied. The output metrics are results of continuously monitoring the metrics while input parameters are applied.

The *accessibility* α defines if the database is still reachable by clients (accepting incoming connections) and accepts read and write requests. While accessibility represents `boolean` values over time, the *performance impact* ϕ represents

Table 3. Output metrics

Output metric		Description
Statistics	Accessibility	DDBMS is accessible for client requests (read and write)
	Performance impact	Throughput (read/write requests per second), latency of requests
	Request error rate	Amount of failed requests due to data unavailability
Times	Take over time	Time until the failure spec is being masked by the DDBMS
	Recovery time	Time until the recovery spec is applied by the DDBMS

the throughput the DDBMS can handle during the evaluation scenario, including the amount of requests and the latency for request handling. For instance the performance may be decreased if replicas are down. In case of node failures, not all data partitions of a DDBMS may be available until the failure is handled. The *request error rate* ϵ describes as output metric the amount of failed requests due to data unavailability over the evaluation time.

The output metrics *take over time* and *recovery time* specify the measured time the DDBMS required to identify the applied failure specification, and the time it takes to apply the recovery specification. The accessibility, the performance impact and the data loss rate are time series values over the time the evaluation scenario is being applied. These values are measured periodically at runtime, are then aggregated and statistically represented in an percentile ranking over the time separately for the time it takes to apply (i) the failure specification and (ii) the recovery specification.

From the described output metrics, the overall availability metric of the evaluated DDBMS can be calculated. From the output metrics, only a configurable amount of percentiles (e.g. > 90) are considered. Each of the three metrics accessibility, performance impact and data loss rate gets its configurable weighting factor $W_\alpha, W_\epsilon, W_\phi$ resulting in the overall availability metric defined as $\Theta = \alpha * W_\alpha + \epsilon * W_\epsilon + \phi * W_\phi$.

5 Availability Evaluation

In this section we present an extensible methodology to evaluate the availability capabilities of DDBMSs based on the defined input parameters and output metrics (*cf.* Sect. 4). Therefore we define an adaptable evaluation process, which emulates the previous failure levels and enacts the respective recovery actions. This process enables the monitoring of the defined availability metrics in order to analyse the high availability efficiency of the evaluated DDBMS. First, we introduce the evaluation process, defining the required evaluation states and

transitions. Second, we present an algorithm to inject cloud resource failures on different levels, based on a predefined failure specification.

5.1 Evaluation Process

Emulating cloud resource failures and monitoring the defined availability metrics, requires the definition of a thorough and adaptable evaluation process, from the DDBMS deployment in the cloud over the simulation of cloud resource failures to the recovering of the DDBMS. A fine-grained evaluation process is depicted in Fig. 2, where the monitoring periods of the availability metrics are presented in the white box, the evaluation process state in the yellow box and the framework components in the blue box. In the following we introduce the evaluation process states and the monitoring periods, while the framework components are described in Sect. 6.

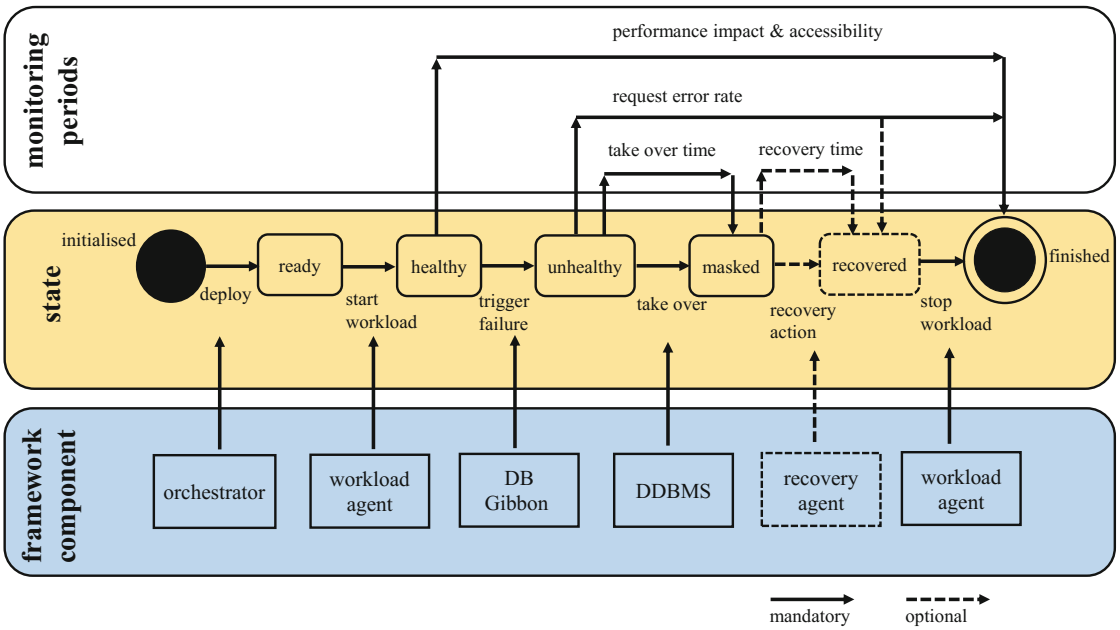


Fig. 2. Evaluation process

The depicted evaluation process in Fig. 2 illustrates an exemplary evaluation, which runs through all defined states exactly once by executing the respective transitions (*cf.* Tables 4 and 5). Yet, it is also possible to leave out dedicated transitions such as *recovery action*. The Gibbon framework executes each evaluation process and it also supports the combination of multiple evaluation processes into an *evaluation scenario*.

Throughout each evaluation process, the defined availability metrics (*cf.* Sect. 4) need to be monitored. Yet, the monitoring period for each availability metric depends on the evaluation process state as depicted in Fig. 2. The *performance impact* and *accessibility* is monitored from the *healthy* to the finished state to analyse the performance development over the intermediate states and compare it with the performance at the beginning. The *request error rate*

Table 4. Evaluation states

State	Description
Initialised	A new evaluation process is triggered
Ready	All nodes of the DDBMS deployed in the cloud and the configuration is finished, *i.e.* the DDBMS is operational
Healthy	All nodes of DDBMS are serving requests
Unhealthy	n nodes of the DDBMS are not operational due to cloud resource failures, the DDBMS has not yet started take over actions.
Masked	The DDBMS initiated automatically the take over of n replicas to reestablish the availability of all data records. The DDBMS is operational again but with $-n$ nodes
Recovered	The number of nodes complies again with the initial number of nodes, all nodes of DDBMS are serving requests
Finished	The evaluation process is finished

Table 5. Evaluation transitions

Transition	Description
Deploy	The DDBMS is being deployed and configured, *i.e.* dedicated VMs are allocated in specified locations
Start workload	A constant workload is started against the deployed DDBMS
Trigger failure	A specified cloud resource failure is emulated by the DB Gibbon component (*cf.* Sect. 5.2, provoking the failure of n nodes
Take over	DDBMS recognizes the failure of n nodes and initialises the take over of the remaining replicas by propagating the new locations for the currently unavailable data records
Recovery action	the DDBMS is restored to its actual number or nodes by adding n new nodes to the DDBMS. In this process the new nodes are integrated in the running DDBMS and the data is redistributed
Stop workload	The workload is stopped

is monitored during the *unhealthy* to the *recovered* state to analyse the development of the failed request rate. The *take over time* is monitored in the transition from the *unhealthy* to the *masked* state while the *recovery time* is monitored in the transition from the *masked* to the *recovered* state.

5.2 DB Gibbon Algorithm

In order to emulate failing cloud resources on the different levels, we build upon the concepts of Netflix's Simian Army [26] and adapt these concepts

for DDBMS in the cloud. Therefore, we define an algorithm, which is able to enact the introduced cloud failure levels (cf. Sect. 3.2) for DDBMSs. Following the Simian Army [26] concepts, we name our algorithm the *DB Gibbon*. The algorithm is depicted in Listing 1.1 and requires as input parameters a list of `failures <List<failure >>` and the `dbDeployment`. The `failures` list contains n `failure` objects with the attributes `failureLevel`, indicating the cloud resource failure level (*cf.* Table 1), `failureQuantity` specifying the number of failures to be enacted by the DB Gibbon and `nodeType` specifying the failing nodes type. Possible `nodeTypes` are `<ANY, DATA, MANAGEMENT, QUERY>` The `dbDeployment` parameter contains the information of the deployed DDBMS topology, *i.e.* the mapping of each node to the cloud resource. Based on these parameters the DB Gibbon algorithm enacts the specified failures in the transition to the *unhealthy* state as depicted in Fig. 2. After enacting all failures, the algorithm returns the updated deployment, which is used to trigger the *recovery action* by calculating the difference to the initial deployment and deriving the required recovery actions.

Listing 1.1. DB Gibbon Algorithm

```
input: failures<List<failure>>, dbDeployment
output: dbDeployment
begin
 for each failure in failures
  if failure.failureLevel == failureLevel.VM
   def VMs List<VM> ← dbDeployment.getVMsOfNodeType(failure.nodeType)
   for (int i : failure.failureQuantity)
    def failedVM VM ← failRandomVM(VMs)
    dbDeployment ← updateDeployment(dbDeployment, failedVM)
   end

  else if failure.failureLevel == failureLevel.AVAILABILITY_ZONE
   for (int i : failure.failureQuantity)
    def VMs List<VM> ← dbDeployment
    .availabilityZone(i).getVMsOfNodeType(failure.nodeType)
    def failedVMs List<VM> ← failVMs(VMs)
    dbDeployment ← updateDeployment(dbDeployment, failedVMs)
   end

  else if failure.failureLevel == failureLevel.REGION
   for (int i : failure.failureQuantity)
    def VMs List<VM> ← dbDeployment.region(i)
    .getVMsOfNodeType(failure.nodeType)
    def failedVMs List<VM> ← failVMs(VMs, failureSeverity)
    dbDeployment ← updateDeployment(dbDeployment, failedVMs)
   end

  //only private cloud deployments
  else if failure.failureLevel == failureLevel.PHYSICAL_HOST
   for (int i : failure.failureQuantity)
    def VMs List<VM> ← dbDeployment.physicalHost(i)
    .getVMsOfNodeType(failure.nodeType)
    def failedVMs List<VM> ← failVMs(VMs, failureSeverity)
    dbDeployment ← updateDeployment(dbDeployment, failedVMs)
   end
  else
   return FAIL
 end
 return dbDeployment
end
```

6 Architecture

This section presents the architecture of the novel Gibbon Framework for executing the introduced evaluation methodology (*cf.* Fig. 2). A high-level view on the architecture is depicted in Fig. 3, introducing the technical framework components and their interactions between each other. The entry point to the Gibbon framework represents the evaluation API, expecting the evaluation scenario specification. This specification comprises four sub specifications for the respective framework components. In the following each framework component is explained with respect to the required specification and its technical details.

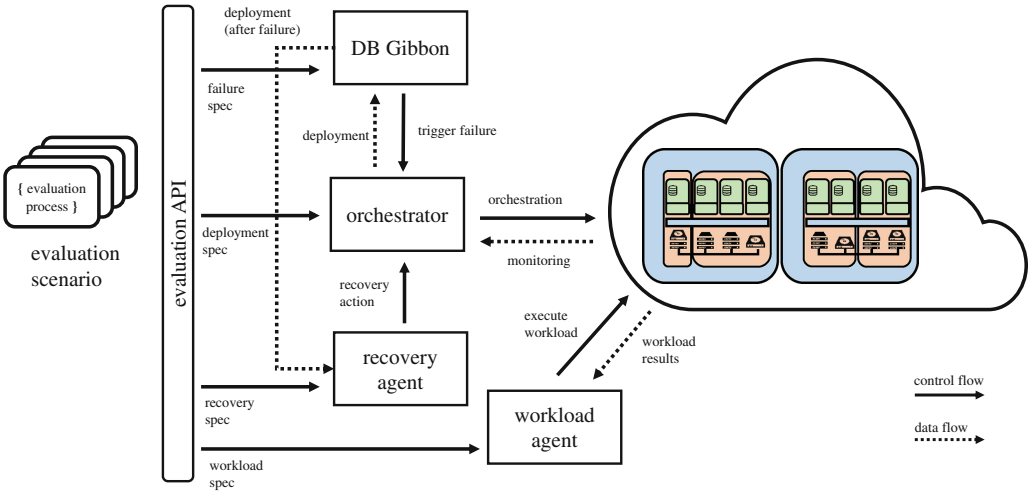


Fig. 3. Gibbon evaluation framework architecture

Orchestrator. Orchestrator receives the *deployment specification*, which comprises the description required cloud resources and the actual DDBMS with its configuration. Hence, the orchestrator interacts with the cloud provider APIs to provision the cloud resources and to orchestrate the DDBMS on these resources. As carved out in our previous work, the usage of advanced Cloud Orchestration Tools (COTs) over basic DevOps tools is preferable as COTs abstract cloud provider APIs and provide monitoring and adaptation capabilities during runtime [4]. The monitoring capabilities comprise general system metrics as well as customisable application specific metrics. Hence, the monitoring capabilities of COTs can be exploited to measure metrics such as the DDBMS nodes state or ongoing maintenance operations, which are required to express the availability metric (*cf.* Sect. 4). COTs offer run-time adaptations such as deleting or suspending cloud resources or DDBMS nodes, adding new cloud resources and DDBMS nodes or the execution of additional applications on the existing DDBMS nodes. These capabilities are used by the DB Gibbon and recovery agent components. The Gibbon framework builds upon the Cloudiator COT[2] [10], which supports

[2] http://cloudiator.org/.

the main public IaaS providers (Amazon EC2[3], Google Compute[4], Microsoft Azure[5]) as well as private clouds built upon OpenStack[6] and provides advanced cross-cloud monitoring and adaptation capabilities [12].

DB Gibbon. This component is responsible to emulate cloud resources failures based on the provided *failure specification*. It queries the COT for the current DDBMS deployment information, *i.e.* the mapping of nodes to cloud resources and their location. Based on the deployment information and the failure specification, it executes the DB Gibbon algorithm (*cf.* Sect. 5.2). In the execution phase of the algorithm the DB Gibbon interacts with the COT to enact the actual failures on cloud resource level, *e.g.* deleting all VMs of the DDBMS which rely in availability zone A.

Recovery Agent. The component receives the *recovery specification*, which comprises the defined recovery actions (*cf.* Sect. 3.2). It collects the DDBMS deployment state from the DB Gibbon after its execution and map the recovery actions to the failed resources. To enact the actual recovery actions, the recovery agent interacts with the COT, which executes the cloud provider API calls and orchestrates the DDBMS nodes.

Workload Agent. It keeps the DDBMS under constant workload during the evaluation process to measure the performance development during the different evaluation states by receiving a *workload specification*, describing the targeted operation types, the data set size and the number of operations. Further, the workload agent is responsible to measure the performance metrics and store them for further analysis in the context of the availability metrics. Our framework uses the Yahoo Cloud Serving Benchmark (YCSB) as workload agent [8] as the YCSB supports distributed execution, multiple DDBMSs and easy extensibility.

7 Discussion

In this section, we discuss the effectiveness of the Gibbon framework based on a concrete evaluation scenario for MongoDB[7]. First, we describe the applied deployment, failure, recovery, and workload specifications and second, we discuss early evaluation results of the accessibility, take over and recovery time.

7.1 Evaluation Scenario: MongoDB

We select MongoDB as the most prevalent document-oriented data store[8] as the preliminary DDBMS to evaluate. MongoDB is classified as CP in the context of

[3] https://aws.amazon.com/ec2/.
[4] https://cloud.google.com/compute/.
[5] https://azure.microsoft.com.
[6] https://www.openstack.org/.
[7] https://www.mongodb.com/.
[8] http://db-engines.com/en/ranking_trend.

the CAP theorem but still provides mechanisms to enable high availability [11], such as automatic take over in case of node failures. MongoDB's architecture is built upon three different services: (1) `mongos` act as query router, providing an interface between clients and a sharded cluster, (2) `configs` store the metadata of a sharded cluster, (3) `shards` persist the data. A group of shards build a replica set with one `primary` handling all requests and n `secondaries`, synchronizing the primary data and taking over in case the `primary` becomes unavailable.

The applied evaluation scenario builds upon a single evaluation process as depicted in Fig. 2. Yet, in this preliminary evaluation process we do not apply a constant workload during the evaluation but use three different data set sizes which are inserted in MongoDB during the deployment of MongoDB. Hence, we only measure the metrics *take over time* and *recovery time*. Accessibility is measured by periodic connection attempts by a MongoDB client.

The *deployment specification* comprises one `mongos` and three `shards` nodes, building a MongoDB replica set with one `primary` and two `secondaries` with full-replication and no sharding. For the sake of simplicity we did not deploy a production-ready setup with multiple `mongos` and `config` nodes. All nodes are provisioned on a private cloud based on OpenStack version Kilo with full and isolated access to all physical and virtual resources. All nodes are provisioned within the region *Ulm* and the availability zone *University*. Each node runs on a VM with 2 vCPUs, 4GB RAM, 40GB disk and Ubuntu 14.04.

As *failure specification* we applied one `failure` object with the attributes `failureLevel=VM, failureQunatity=1 nodeType=data` to the DB Gibbon.

The *recovery specification* defines to add of a new data node (*i.e.* secondary in MongoDB), as soon as MongoDB reaches the *masked* state after a node failure. MongoDB is configured to elect a new primary if the recent primary node failed.

As stated above, we do not apply a constant workload during the evaluation, the *workload specification* only defines the data set by number of `records=100K, 400K, 800K` and `record size=10KB`.

7.2 Evaluation Results

The preliminary evaluation results reveals two insights: the behaviour in case of node failures and in the case of adding a new replica to the system.

In case of a node failure, the behaviour depends on the failed node type. If a secondary fails, connected clients will loose their read-only connections and have to reconnect to another secondary or to the primary node. In this case we can assume, that at least the primary node is still *accessible*, so clients can reconnect immediately. If the primary fails, clients will loose their read/write connections, but may connect immediately to a secondary node for read requests. Remaining secondaries will recognize that the primary fails and will elect the new primary after a configurable timeout. During this timeout and election phase, no clients can issue write requests, the DDBMS is hence only *accessible* for read requests and not for write requests. Per default, the primary failover timeout is configured to ten seconds. In repeated experiments an average duration for election and

primary take over of five seconds (± one second) is measured. Hence, the overall *take over time* is 15 s.

Whenever a node failure happens, the evaluation scenario adds a new replica after the remaining nodes elected a primary. The new secondary node has to synchronize the stored data from other nodes, for consistency reasons from the primary node. The replication time depends on the size of stored data. For 100 k, 400 k, and 800 k items the median for replication time with 30 runs takes 31 s, 300 s, and 553 s with a standard deviation of 7s, 15 s, and 25 s. The *recovery time* hence depends on the amount of data to replicate, plus a fixed amount of time it takes to allocate new resources on the Cloud. The DDBMS is *accessible* throughout the replication, yet with reduced resources due to the ongoing synchronisation.

8 Related Work

The view on availability in distributed systems evolved over the last two decades. The CAP theorem published in 2000, states that any networked shared-data system can only have two of the three properties consistency, availability and partition tolerance [6]. A revisited view on the CAP theorem is presented in 2012 [5], reflecting how emerging distributed systems such as DDBMSs adapted their consistency, availability and partition tolerance properties as the decision for two out of the three properties is not a binary decision [5].

The classification of DDBMSs according to their CAP properties in CA or CP DDBMSs is a widely discussed topic in database research. A first overview and analysis of emerging DDBMSs is provided by [7], analysing various DDBMSs with respect to their availability capabilities in the context of the CAP theorem.

DDBMSs mechanism to provide high availability are discussed by [15], breaking down the technical replication strategies from master-slave replication to masterless, asynchronous replication of 19 DDBMSs. Yet, the high availability mechanisms are only discussed on a theoretical level and no evaluation of their efficiency is proposed. Further, cloud resources are introduced as the preferable resources to run DDBMSs but the different failure levels and their implication to the availability of the DDBMS are not considered.

A similar approach is followed by [23], adding a dedicated classification of common DDBMSs with respect to their CAP properties, *i.e.* AP or CP. Further, the usage of cloud resources is discussed with respect to virtualisation and data replication across multiple regions. Yet, the focus relies on enabling consistency guarantees of wide-area DDBMSs while side-effects by using cloud resources that effect as well as ensure availability in DDBMS are not considered in detail.

An availability- and reliability-centric classification of DDBMSs is presented by [11]. Hereby, the challenges to provide non-functional requirements such as replication, consistency, conflict management, and partitioning are broken down in a fine grained classification schema and a set of 11 DDBMS are analysed. Two availability affecting issues and solutions are presented, overloading and node failures: *(i)* Is a DDBMS not available due to overloading, the DDBMS

needs to be scaled out. *(ii)* Replicas need to be in place to overcome database node failures. Yet, the proposed solutions are on an architectural level and the actual capabilities of DDBMSs are not evaluated in real-world scenarios.

While theoretical classifications of DDBMSs provide a valuable starting point for a first selection of DDBMSs, the final selection still remains challenging due to the heterogeneous DDBMSs landscape. Hence, database evaluation frameworks provide additional insights in DDBMS capabilities by evaluation dedicated evaluation tiers based on different workload domains. While available evaluation frameworks such as YCSB [8] or YCSB++ [20], focus on the evaluation performance, scalability, elasticity and consistency, the availability tier is not yet considered by these frameworks [24]. First approaches in availability evaluation are based on the YCSB and focus the on decreased availability due to an overloaded database [18,25]. While an evaluation framework focusing on the availability of cloud-hosted DDBMSs is not yet available, an approach to enact synthetic failures on cloud resources is described by [26] and implemented at Netflix. Yet, this approach only describes the failure scenarios in the cloud and does not propose evaluation metrics or an evaluation methodology for cloud applications in general and DDBMS in particular. A first approach towards the availability metrics is presented by [2], yet the focus relies on the resilience of DBMSs, while DDBMSs and their availability mechanism are not considered. Existing failure-injection tools such as the DICE fault injection tool[9] or jepsen[10] either inject failures on node or DBMS level but do not support the injection of resource failures in different granularity.

9 Conclusion and Future Work

In the last decade the database management system (DBMS) landscape grew fast, resulting in a very heterogeneous DBMSs landscape, especially when it comes to distributed database management systems (DDBMSs). As cloud computing is the preferable way to run distributed applications, cloud computing seems to be the choice to run DDBMSs. Yet, the probability of failures increases with the number of distributed entities and cloud computing adds another layer of possible failures. Hence, DDBMSs apply data replication across multiple DDBMS nodes to provide high availability in case of node failures. Yet, providing high availability comes with limitations with respect to consistency or partition tolerance as stated by the CAP theorem. As these limitations are not binary, a vast number of high availability implementations in DDBMSs exist. This makes the selection of a DDBMS to run in the cloud a complex task, especially as supportive availability evaluation frameworks are missing.

Therefore, we present Gibbon, a novel availability evaluation framework for DDBMSs to increase the trust in running DDBMSs in the cloud. We describe levels of cloud resource failures, existing DDBMS concepts to provide high availability and distill a set of five quantifiable availability metrics. Further, we

[9] https://github.com/dice-project/DICE-Fault-Injection-Tool.
[10] https://github.com/jepsen-io/jepsen.

derive the DDBMSs specific technical details, affecting the availability evaluation processes. Building upon these findings, we introduce the concept of extensible evaluation scenarios, comprising n evaluation processes. Further, we present the DB Gibbon, which emulates cloud resource failures on different levels.

The Gibbon framework executes the defined evaluation scenarios for generic DDBMSs and cloud infrastructures. Its architecture comprises an orchestrator to deploy the DDBMS in the cloud, a workload agent, a recovery agent and the DB Gibbon to inject cloud resources failures. As preliminary evaluation, we evaluate the take over and recovery time of MonogDB in a private cloud, by injecting failures on the virtual machine level.

Future work will comprise an in-depth evaluation of multiple well-adopted DDBMSs (See footnote 8) based on the Gibbon framework. Further, the definition of a minimal evaluation scenario to derive a significant availability rating, is in progress. In this context, the statistical calculations of the overall availability rating index will be refined. Finally, the portability of Gibbon evaluate the availability of generic applications running in the cloud will be evaluated.

Acknowledgements. The research leading to these results has received funding from the EC's Framework Programme HORIZON 2020 under grant agreement numbers 644690 (CloudSocket) and 731664 (MELODIC). We also thank Daimler TSS for the encouraging and fruitful discussions on the topic.

References

1. Abadi, D., Agrawal, R., Ailamaki, A., Balazinska, M., Bernstein, P.A., Carey, M.J., Chaudhuri, S., Chaudhuri, S., Dean, J., Doan, A., et al.: The Beckman report on database research. Commun. ACM **59**(2), 92–99 (2016)
2. Almeida, R., Neto, A.A., Madeira, H.: Resilience benchmarking of transactional systems: experimental study of alternative metrics. In: PRDC (2017)
3. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. TDSC **1**(1), 11–33 (2004)
4. Baur, D., Seybold, D., Griesinger, F., Tsitsipas, A., Hauser, C.B., Domaschka, J.: Cloud orchestration features: are tools fit for purpose? In: UCC (2015)
5. Brewer, E.: Cap twelve years later: how the "rules" have changed. Computer **45**(2), 23–29 (2012)
6. Brewer, E.A.: Towards robust distributed systems. In: PODC (2000)
7. Cattell, R.: Scalable SQL and NoSQL data stores. ACM Sigmod Rec. **39**(4), 12–27 (2011)
8. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking cloud serving systems with YCSB. In: SoCC (2010)
9. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: Amazon's highly available key-value store. ACM SIGOPS Oper. Syst. Rev. **41**(6), 205–220 (2007)
10. Domaschka, J., Baur, D., Seybold, D., Griesinger, F.: Cloudiator: a cross-cloud, multi-tenant deployment and runtime engine. In: SummerSOC (2015)
11. Domaschka, J., Hauser, C.B., Erb, B.: Reliability and availability properties of distributed database systems. In: EDOC (2014)

12. Domaschka, J., Seybold, D., Griesinger, F., Baur, D.: Axe: a novel approach for generic, flexible, and comprehensive monitoring and adaptation of cross-cloud applications. In: ESOCC (2015)
13. Ford, D., Labelle, F., Popovici, F.I., Stokely, M., Truong, V.A., Barroso, L., Grimes, C., Quinlan, S.: Availability in globally distributed storage systems. In: OSDI (2010)
14. Geraci, A., Katki, F., McMonegal, L., Meyer, B., Lane, J., Wilson, P., Radatz, J., Yee, M., Porteous, H., Springsteel, F.: IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries: 610. IEEE Press (1991)
15. Grolinger, K., Higashino, W.A., Tiwari, A., Capretz, M.A.: Data management in cloud environments: NoSQL and NewSQL data stores. In: JoCCASA (2013)
16. Gunawi, H.S., Hao, M., Suminto, R.O., Laksono, A., Satria, A.D., Adityatama, J., Eliazar, K.J.: Why does the cloud stop computing? Lessons from hundreds of service outages. In: SoCC (2016)
17. Haerder, T., Reuter, A.: Principles of transaction-oriented database recovery. In: CSUR (1983)
18. Konstantinou, I., Angelou, E., Boumpouka, C., Tsoumakos, D., Koziris, N.: On the elasticity of NoSQL databases over cloud management platforms. In: CIKM (2011)
19. Mell, P., Grance, T.: The NIST definition of cloud computing. Technical report, National Institute of Standards & Technology (2011)
20. Patil, S., Polte, M., Ren, K., Tantisiriroj, W., Xiao, L., López, J., Gibson, G., Fuchs, A., Rinaldi, B.: YCSB++: benchmarking and performance debugging advanced features in scalable table stores. In: SoCC (2011)
21. Pritchett, D.: Base: an acid alternative. Queue **6**, 48–55 (2008)
22. Sadalage, P.J., Fowler, M.: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Pearson Education, London (2012)
23. Sakr, S.: Cloud-hosted databases: technologies, challenges and opportunities. Cluster Comput. **17**(2), 487–502 (2014)
24. Seybold, D., Domaschka, J.: Is distributed database evaluation cloud-ready? In: ADBIS (2017)
25. Seybold, D., Wagner, N., Erb, B., Domaschka, J.: Is elasticity of scalable databases a myth? In: IEEE Big Data (2016)
26. Tseitlin, A.: The antifragile organization. Commun. ACM **56**(8), 40–44 (2013)
27. Zhong, M., Shen, K., Seiferas, J.: Replication degree customization for high availability. In: EuroSys (2008)

Chapter 12

[core5] Mowgli: Finding Your Way in the DBMS Jungle

This article is published as follows:

Daniel Seybold, Moritz Keppler, Daniel Gründler, and Jörg Domaschka. "Mowgli: Finding Your Way in the DBMS Jungle", *10th ACM/SPEC International Conference on Performance Engineering (ICPE)*, published 2019, ACM, DOI: https://doi.org/10.1145/3297663.3310303

Mowgli: Finding Your Way in the DBMS Jungle

Daniel Seybold
Institute of Information Resource Management
Ulm University, Germany
daniel.seybold@uni-ulm.de

Moritz Keppler
Daimler TSS
Ulm, Germany
moritz.keppler@daimler.com

Daniel Gründler
Daimler TSS
Ulm, Germany
daniel.gruendler@daimler.com

Jörg Domaschka
Institute of Information Resource Management
Ulm University, Germany
joerg.domaschka@uni-ulm.de

ABSTRACT

Big Data and IoT applications require highly-scalable database management system (DBMS), preferably operated in the cloud to ensure scalability also on the resource level. As the number of existing distributed DBMS is extensive, the selection and operation of a distributed DBMS in the cloud is a challenging task. While DBMS benchmarking is a supportive approach, existing frameworks do not cope with the runtime constraints of distributed DBMS and the volatility of cloud environments. Hence, DBMS evaluation frameworks need to consider DBMS runtime and cloud resource constraints to enable portable and reproducible results. In this paper we present Mowgli, a novel evaluation framework that enables the evaluation of non-functional DBMS features in correlation with DBMS runtime and cloud resource constraints. Mowgli fully automates the execution of cloud and DBMS agnostic evaluation scenarios, including DBMS cluster adaptations. The evaluation of Mowgli is based on two IoT-driven scenarios, comprising the DBMSs Apache Cassandra and Couchbase, nine DBMS runtime configurations, two cloud providers with two different storage backends. Mowgli automates the execution of the resulting 102 evaluation scenarios, verifying its support for portable and reproducible DBMS evaluations. The results provide extensive insights into the DBMS scalability and the impact of different cloud resources. The significance of the results is validated by the correlation with existing DBMS evaluation results.

CCS CONCEPTS

• **Information systems → Database performance evaluation**;
• **Computer systems organization → Cloud computing**;

KEYWORDS

benchmarking, cloud, NoSQL, scalability, distributed database

ACM Reference Format:
Daniel Seybold, Moritz Keppler, Daniel Gründler, and Jörg Domaschka. 2019. Mowgli: Finding Your Way in the DBMS Jungle. In *Tenth ACM/SPEC International Conference on Performance Engineering (ICPE '19), April 7–11, 2019, Mumbai, India.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3297663.3310303

ICPE '19, April 7–11, 2019, Mumbai, India

ACM ISBN 978-1-4503-6239-9/19/04…$15.00
https://doi.org/10.1145/3297663.3310303

1 INTRODUCTION

IoT and Big Data drive the need for highly scalable and (geo-) distributed data management. The NoSQL landscape provides distributed database management systems (DBMS) that promise to fulfil the need for both scale and distribution and in some cases even geo-distribution, together with non-functional properties such as elasticity and high-availability. Additionally, cloud computing offers the necessary mechanisms to enable scalability on resource level. Yet, choosing the right DBMS set-up in the jungle of available solutions is a complex task that is not done with the selection of a well-suited DBMS[1], but continues with the selection of a cloud provider, and ends with the choice of the right size and amount of virtual machines. The three choices influence each other [35], so that making independent decisions may lead to sub-optimal results. Additionally, runtime parameters, including the expected workload, consistency requirements, and availability considerations, are influencing the set-up and depend on each other: for instance, the type of workload can influence whether a user should pay for having a local SSD attached to their virtual machines or not [18].

While benchmarking is an established approach to select software systems as well as hardware platforms, existing DBMS benchmarking frameworks cannot cope with the volatility of cloud environments [35], particularly as volatile environments demand for reliable and reproducible benchmarking [29]. Based on these observations, we claim that even with the knowledge of the workload and non-functional constraints, a manual selection of DBMS, cloud provider(s), and virtual machine types cannot deliver satisfactory results and that suitable tool support is strongly needed.

Only by this approach we are able to find an appropriate initial solution, but also keep up with DBMS version upgrades and new DBMSs entering the market, as well as to address new cloud providers and virtual machine types. This paper presents *Mowgli*, a novel DBMS evaluation and benchmarking framework that fully automates the whole evaluation flow from the cloud resource allocation, DBMS deployment, workload execution and the DBMS cluster adaptation. Its underlying orchestration engine is cloud provider-agnostic and supports cross-cloud evaluation scenarios [5].

[1] in June 2018, http://nosql-databases.org lists more than 225 NoSQL database projects

In particular, Mowgli helps answering the DBMS runtime centric question Q1 and the cloud resource allocation centric question Q2:

Q1: "How much throughput for workload W_x can DBMS D_x achieve with cluster size CS_x, replication factor RF_x, and ensures the write-consistency of WC_x if operated on VM type VM_x in cloud C_x?"

Q2: "Which cluster size CS_x of DBMS D_x achieves the highest throughput for workload W_x if operated with replication factor RF_x, ensuring write-consistency WC_x and running on VM types VM_x in cloud C_x if the maximum number of available cloud resources is $CR - MAX_x$?

The motivation for Mowgli is our need to find a DBMS-cloud set-up that is capable of handling an IoT scenario with a growing number of sensors where each sensor s_i would issue its current state every t seconds and no message is allowed to be dropped. This leads to the requirement that the chosen DBMS needs to provide a constant write throughput even in the case of failures. While this paper does not present the final outcome of the selection process, we apply this use case as a frame for validating Mowgli. In particular, we validate Mowgli by applying an evaluation of two DBMSs over three different cloud environments for write-intensive workloads.

The remainder of this paper is structured as follows: Section 2 details the challenges of DBMS evaluation while Section 3 describes Mowgli. Section 4 presents the evaluation scenarios we use to validate our approach and Section 5 analyses the evaluation results. Section 6 discusses the usability and significance of Mowgli. Section 7 presents related work, before Section 8 concludes.

2 DBMS EVALUATION CHALLENGES

In order to guide the DBMS selection process, the introduced questions Q1 and Q2 need to be addressed by evaluating potential DBMS. Yet, a significant evaluation needs to consider multiple domains as the results are affected by the applied *cloud resource*, *DBMS runtime* and *workload* constraints. Hence, the evaluation approach requires the specification of multi-domain evaluation scenarios as depicted in Figure 1. Each evaluation domain comprises its own set of domain specific constraints, which affect the results for the specified evaluation objectives [35]. Consequently, domain knowledge in each evaluation domain is required, which makes the DBMS evaluation a complex and error prone task. In order to reduce this complexity and enable the portable and reproducible *evaluation scenario execution*, dedicated tool support is required. In the following, we introduce each evaluation domain with respect to relevant constraints and present the challenges with respect to execute multi-domain evaluation scenarios in a portable, reproducible and consistent manner.

2.1 DBMS Runtime Domain

With the rise of the NoSQL data models [8, 12, 22], the usage of distributed architectures for shared-nothing DBMS has become a common approach to provide scalability, elasticity and availability [32]. In this context, the extent of DBMS runtime constraints has increased as distributed DBMS aim to provide flexibility for multiple usage scenarios [16, 20]. Common configurable runtime properties of distributed DBMS are the replication factor, read/write consistency settings and the sharding strategy, while there is an extensive number of DBMS-specific runtime constraints such as storage engines or compression algorithms. Consequently, comparative evaluation scenarios need to abstract the DBMS runtime domain to general runtime constraints, enabling the specification of consistent and portable DBMS runtime specifications [7, 29]. Moreover, the DBMS runtime specification needs to be extensible to allow DBMS-specific evaluations based on custom constraints [6, 7].

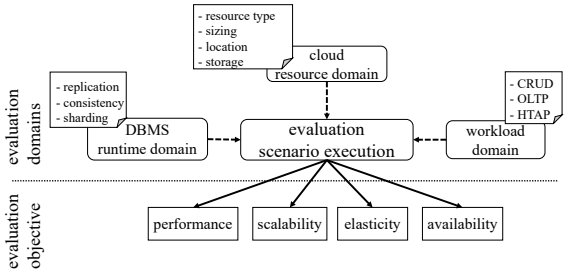


Figure 1: DBMS Evaluation Domains

2.2 Cloud Resource Domain

While cloud resources have become a common solution to operate DBMS [33], cloud resource offerings are getting more heterogenous with respect to the offered compute resource type; compute resource sizing; storage backends; control over tenant isolation and control over locations [4]. Especially for DBMS, storage backends are important as remote storage makes it easier to scale out a DBMS but network latency and bandwidth can limit the DBMS performance. Dedicated storage reduces these limitations, but a failure of a physical server decreases availability and failover mechanisms are required [1]. Hence, evaluation scenarios have to include existing cloud resource offers by abstracting provider specific details and enabling a consistent and portable cloud resource specification [29, 35].

2.3 DBMS Workload Domain

DBMS workloads emulate heterogenous application domains, from synthetic create, read, update, delete (CRUD) operations over more realistic Online Transaction Processing (OLTP) to novel Hybrid Transaction-Analytical Processing (HTAP) workloads [35]. While realistic workloads increase the significance of the results, they typically make use of DBMS-specific features, which limits their field of use [31]. In addition, each workload implementation provides its own set of workload constraints, which have direct impact on the results. Hence, workload specifications for comparative DBMS evaluations require portable workloads to compare different DBMS against the evaluation objectives [6, 29]. DBMS-specific scenarios need to support realistic workloads to enable the in-depth evaluation of DBMS-specific features [6, 31]. Respectively, the workload specification has to abstract common workload constraints to enable comparative and DBMS-specific workload specifications.

2.4 Evaluation Scenario Execution

The consistent specification of evaluation scenarios considering the introduced domains requires abstract evaluation templates that are

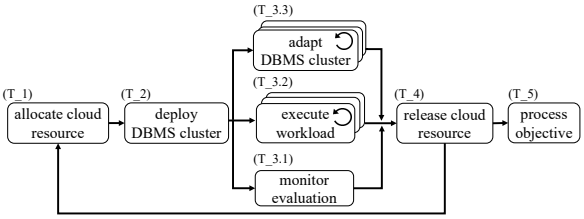


Figure 2: DBMS Evaluation Process

enriched with the concrete domain constraints. This enables the portable and reproducible execution of evaluation scenarios. Yet, the manual execution of such multi-domain evaluation scenarios is a complex and error prone process. Each domain requires detailed knowledge on its own and the entire evaluation process comprises a sequence of multiple interdependent evaluation tasks as depicted in Figure 2. Consequently, a supportive framework is required that automates the evaluation process and fulfils the established requirements of DBMS evaluation [6, 7, 21, 29, 35]:

Ease of use (R1) : The deployment and configuration of the framework needs to be simple and it needs to provide user-friendly interfaces to specify and execute the evaluation scenarios [6, 7, 21, 29]

Portability (R2): In order to execute the evaluation scenarios for different domain properties, the framework has to abstract technical implementations of each evaluation domain and map the high-level evaluation scenarios to concrete technical implementations for each evaluation domain [21, 35]

Reproducibility (R3): The framework needs to provide comparable evaluation scenario templates, which ensure the deterministic execution for concrete domain constraints [6, 29, 35]

Automation (R4): The automated execution of multi-domain evaluation scenarios requires the orchestration of evaluation tasks across all domains to ensure the reproducibility and portability [35]. In addition, complex evaluation objectives such as elasticity or availability require the DBMS cluster adaptation at evaluation runtime [36].

Significance (R5): In order to enable significant results by applying realistic domain constraints, the framework needs to support comparative and realistic workloads, commercial cloud resource offerings and relevant DBMS [6, 29, 35]

Extensibility (R6): As each evaluation domain is constantly evolving, the framework needs to provide an extensible architecture and interfaces that allow the easy integration of future domain specific constraints and evaluation objectives [7, 21, 29, 35].

3 MOWGLI

In the following, we present the multi-domain evaluation framework Mowgli[2] that builds upon existing DBMS evaluation concepts [34]. Mowgli automates the entire evaluation process shown in Figure 2 by enabling the definition and execution of portable and reproducible evaluation scenarios via a loosely coupled and

[2]https://omi-gitlab.e-technik.uni-ulm.de/mowgli

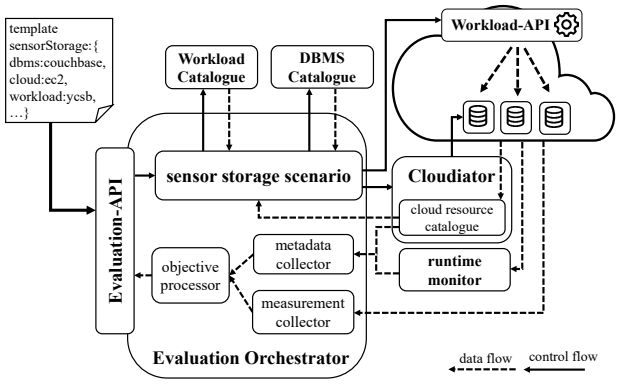


Figure 3: Mowgli architecture

extensible evaluation framework. It does so by exploiting the features of cloud orchestration tools (COTs) [5] and combines them with an extensible DBMS catalogue, an auto-generated cloud resource catalogue and a workload catalogue. The architecture of Mowgli is depicted is in Figure 3. Evaluation templates define the required input of an abstract evaluation scenario and reach the system through the *Evaluation API*. In its current state, Mowgli supports the abstract *sensor storage* evaluation scenario to address Q1 and Q2. The specification of elasticity and availability related evaluation scenarios is subject to ongoing work.

3.1 Evaluation Templates

Each evaluation template comprise three different types of sub-templates[3], which are listed in Table 1-3. The Tables list the abstract domain constraint and Mowgli's supported parameter range. Domain constrains and their parameter range can be customized due to Mowgli's extensible architecture.

The *DBMS runtime template* includes use-case specific DBMS runtime configurations in a DBMS agnostic manner, which are listed in Table 1. Hence, it describes the desired distribution requirements. It requires the mandatory configuration options cluster topology, cluster size and replication factor and offers optional DBMS-specific configurations options. The *cloud resource templates* describes the required compute and storage resources in a cloud provider agnostic way as shown in Table 2. Using this type of agnostic description of both non-functional properties of the DBMS and resources is key to making evaluations portable between DBMS and cloud providers, but also fosters reproducibility of evaluation results. Finally, the *workload template* listed in Table 3 specifies the desired load on the system by referring to known DBMS benchmarks through unified configuration properties which are extended by benchmark specific properties such as read/write consistency settings, DBMS driver settings and request distribution.

3.2 Catalogues

Catalogues enable the mapping from abstract evaluation scenario templates to executable experiments. The *DBMS catalogue* contains

[3]Exemplary input templates are publicly availablehttps://omi-gitlab.e-technik.uni-ulm.de/mowgli/getting-started/tree/icpe2019/examples

Table 1: DBMS Runtime Template

Constraint	Parameter Range
DBMS	$D_x \in \{RDBMS, NoSQL, NewSQL\}$
Cluster Topology	$CT_x \in \{data\ center, cross\ data\ center\}$
Cluster Size	$CS_x \in \{3..n\}$
Replication Factor	$RF_x \in \{n <= CS_x\}$
Custom Configuration	$CC_x \in \{key = value\}$

Table 2: Cloud Resource Template

Constraint	Parameter Range
Cloud API	$C_x \in \{OpenStack, EC2, Google\ Compute\}$
Location	$VM - L_x \in \{rack, availability\ zone, region\}$
vCores	$VM - C_x \in \{2..n\}$
RAM (in GB)	$VM - M_x \in \{2..n\}$
Storage Capacity (GB)	$VM - SC_x \in \{20..n\}$
Storage Type	$VM - ST_x \in \{HDD, SDD, Remote\}$

Table 3: Workload Template

Constraint	Parameter Range
Type	$WT_x \in \{YCSB, TPC - C\}$
Runtime (in seconds)	$WR_x \in \{60..n\}$
Client instances	$WC_x \in \{1..n\}$
Client threads	$WC_x \in \{1..n\}$
Network	$WN_x \in \{public, private\}$
Write Consistency	$WC_x \in \{low, medium, high\}$
Read Consistency	$RC_x \in \{low, medium, high\}$

mappings from DBMS templates to concrete configurations. In particular, based on the DBMS catalogues, Mowgli is able to configure and run different DBMS in the way specified in the DBMS templates. Currently, Mowgli supports Apache Cassandra[4], Couchbase[5], MongoDB[6], Riak [7] and CockroachDB[8].

Similarly, the *cloud resource catalogue* provides a mapping from cloud resource templates to actual cloud resources. As defining this mapping is cumbersome and repetitive, we use the resource discovery features of the COT Cloudiator [4, 15]. For each cloud credential stored at Cloudiator, it automatically creates the cloud-provider specific resource entries in its catalogue as well as a cloud-provider agnostic representation thereof that is referenced by Mowgli's cloud resource templates.

Finally, the *workload catalogue* captures concrete implementations of workloads. Its entries specify what kind of load to issue on the DBMS and in what order. Mowgli supports the *Yahoo Cloud Serving Benchmark (YCSB)* [11] and a DBMS-specific implementation of the TPC-C workload[9]. For our evaluation, we make use of the YCSB as it enables the emulation of a write–heavy sensor storage workload.

[4]http://cassandra.apache.org/
[5]https://www.couchbase.com/
[6]https://www.mongodb.com/
[7]http://basho.com/products/riak-kv/
[8]https://www.cockroachlabs.com/

3.3 Evaluation Process

Using the catalogues, Mowgli is able to map the concrete scenario parameters received through the evaluation-API to the abstract sensor storage scenario specification and create an executable evaluation scenario. The entire execution of a specified evaluation scenario is automated by the *evaluation orchestrator* that orchestrates the tasks depicted in Figure 2. Therefore, an evaluation scenario is internally implemented as workflow with sequential, conditional and parallel tasks. The workflow of the introduced sensor storage scenario is implemented as a subset of the introduced evaluation tasks of Figure 2 using sequential and parallel tasks. In the following, the workflow tasks executed by the *evaluation orchestrator* are presented together with the involved components of Mowgli.

- **T_1:** allocating cloud resources for each evaluation iteration via *Cloudiator* that enacts the cloud provider specific requests
- **T_2:** deploying and configuring the DBMS cluster by fetching the DBMS deployment scripts from the *DBMS catalogue* and passing them to *Cloudiator* to deploy the DBMS cluster on the allocated cloud resources
- **T_3.1** measuring system and DBMS metrics during each run via the *runtime monitor*. The runtime monitor is implemented by the time-series DBMS InfluxDB[10]
- **T_3.2** distributing the workload execution across the specified *workload-API* instances
- **T_4** releasing the cloud resource after the each evaluation iteration via *Cloudiator* and repeating task (1)-(4) according to the scenario parameters
- **T_5** collecting and processing of the evaluation results as follows: the *measurement collector* collects basic performance metrics such as throughput and latency, provided by the applied workload; a scenario-specific *objective processor* computes composed metrics such as *scalability* [11], elasticity [17], availability [36] or the *cloud resource and distribution impact* [35] by correlating the performance metrics with the applied DBMS runtime and cloud resource specifications provided by the *metadata collector*

As the implementation of the sensor storage scenario does not require the adaptation of the DBMS cluster at evaluation runtime, T_3.3 is omitted. Although, Mowgli is able to support the adaptation of the DBMS cluster at runtime by specifying adaptation tasks that use metrics of the *runtime monitor* as adaptation trigger and *Cloudiator* to adapt the DBMS cluster.

4 SENSOR STORAGE EVALUATION SCENARIOS

In order to validate the Mowgli framework, we apply the sensor storage scenario and seek help in answering questions Q1 and Q2. Consequently, we define the sensor storage template and apply

[9]https://github.com/cockroachdb/loadgen
[10]https://docs.influxdata.com/influxdb

Q1 and Q2 specific domain constraints. This section details the choice of DBMSs, their runtime configuration, the selection of cloud resources as well as the sensor workload specification.

4.1 DBMS Specification

For the validation of the DBMS runtime centric Q1, we select Apache Cassandra [27] and Couchbase as DBMSs. The cloud resource centric Q2 is validated by using Apache Cassandra. Both are popular NoSQL DBMSs[11]. They provide a flexible data model and a multi-master architecture that supports automated sharding and horizontal scalability. They only have limited support for complex queries, but for write-heavy workloads, this is negligible. Furthermore, both DBMSs have already been subject to scalability evaluations with respect to read-update workloads and achieved promising results [24, 26, 37]. Also, the availability of other evaluation scenarios allows us to cross-check the results reported by Mowgli for read-update workloads (not part of this paper but carried out with a preliminary version of Mowgli [37]). Apache Cassandra applies the column-oriented data model, while Couchbase applies the document-oriented data model [8]. Table 4 describes the relevant runtime specifications for the selected DBMSs based on the introduced runtime constraints (*cf.* Table 1). Other options are supported, but have not been used for the results presented in this paper. By default, Mowgli configures a DBMS instance to use 50% of the available memory for its operation.

Due to the architectural similarities of both DBMSs comparable cluster topologies, cluster sizes and replication factors can be defined. Yet, they differ when it comes to persistence configuration at client side. Apache Cassandra applies write ahead logging (WAL), while Couchbase does not. Instead, it caches records directly in memory and persists them to disk asynchronously. Couchbase provides the configuration option to enforce replicating a record to *n* replica nodes via *replicateTo* or persisting the record to disk of *n* replica nodes via *persistTo*. For Apache Cassandra we can configure the amount of replicas where an item has to be written to the WAL and the in-memory cache. Consequently, the write consistency configurations can not be exactly mapped for both DBMSs. For Apache Cassandra, we select the write consistency levels *ANY, ONE, TWO* [12] while for Couchbase, we select the following options: NONE (replicateTo=NONE and persistTo=NONE) confirms a write as successful after the record has been transmitted. R-ONE (replicateTo=1) ensures that the record is written to the cache of at least one replica node, and P-ONE (persistTo=1) ensures that the record is persisted to the disk of at least on replicate node.

4.2 Cloud Resource Specification

The portability of our approach is verified by using cloud resources of two different cloud providers. For answering Q1, we apply Mowgli to three different cloud resource configurations as outlined in Table 5: OS_SSD and OS_REMOTE run on a private, OpenStack-based cloud[13] (version Pike) with full and isolated access to all physical and virtual resources. All physical hosts in the OS_SSD availability zone have two dedicated SSDs in Raid-0 configuration. Physical hosts with the OS_REMOTE configuration share one storage server with RAID-6 set-up and magnetic disks. Network bandwidth between physical hosts and the remote storage is 10G. EC2_REMOTE runs Amazon EC2 VM instances in the *Frankfurt* region and the availability zone *eu-central-1*. The selected EC2 instance type is *t2.medium*[14] and each VM is provisioned with a Remote Storage GP2 SSD EBS volume. For the evaluation the comparable VM type $VM-T_{small}$ is selected, which is available in OpenStack and EC2. Each VM type is composed by the tuple vCores, RAM and storage capacity as listed in Table 2.

Table 4: DBMS Runtime Specifications

Specification	Spec_CA	Spec_CB
D_x	Apache Cassandra	Couchbase
Version	3.11.2	5.0.1 community
CT_x	data center	data center
$CS-Q1_x$	3,5,7,9	3,5,7,9
$CS-Q2_x$	3,6,9	-
RF_x	3	3
WC_{low}	ANY	NONE
WC_{medium}	ONE	R-ONE
WC_{high}	TWO	P-ONE

Table 5: Q1 - Cloud Resource Specifications

Specification	OS_SSD	OS_REMOTE	EC2_REMOTE
C_x	OpenStack	OpenStack	EC2
$VM-L_x$	Ulm	Ulm	Frankfurt
$VM-T_{small}$	2 vCores, 4GB RAM, 50GB disk		
$VM-ST_x$	SSD	Remote	Remote
$VM-OS_x$	Ubuntu Server 16.04		
$VM-NET_x$	private		

To emphasize Mowgli's capabilities of evaluating the impact of cloud resources in the context of Q2, OpenStack with the availability zones OS_SSD and OS_REMOTE is selected. Using this private cloud allows the specification of custom VM types and the in-depth analysis of the cloud resource impact. We define an exemplary maximum resource pool $CR-MAX_x$ and specify the respective VM types as listed in Table 6. These VM types are applied to $CS-Q2_{3,6,9}$ to accordingly match the maximum resource pool.

4.3 Workload Specification and Metrics

For our evaluation, we use YCSB version 0.12.0[15], which is integrated in Mowgli (*cf.* Section 3) and allows the specification of a write-heavy workload required by Q1 and Q2. Besides, relying on the widely used YCSB, allows us to validate our results against published results.

The specification is such that the workload is issued through one independent virtual machine running in the same environment

[11] https://db-engines.com/en/ranking
[12] https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlConfigConsistency.html
[13] https://www.openstack.org/
[14] https://aws.amazon.com/ec2/instance-types/
[15] https://github.com/brianfrankcooper/YCSB/releases/tag/0.12.0

Table 6: Q2 - Cloud Resource Specifications

Specification	OS_SSD_Q2	OS_REMOTE_Q2
C_x	OpenStack	OpenStack
$VM - L_x$	Ulm	Ulm
$CR - MAX_x$	18 vCores; 36GB RAM	
$VM - T_{large}$	6 vCores, 12GB RAM, 50GB disk	
$VM - T_{medium}$	3 vCores, 6GB RAM, 50GB disk	
$VM - T_{small}$	2 vCores, 4GB RAM, 50GB disk	
$VM - ST_x$	SSD	Remote
$VM - OS_x$	Ubuntu Server 16.04	
$VM - NET_x$	private	

as the DBMS instances. This virtual machine is configured with 8 vCores, 16GB memory and 20GB of remote storage disk, running Ubuntu Server 16.04. It uses a cloud-internal network to communicate with the DBMS cluster. Table 7 contains the relevant YCSB specifications. The DBMS-specific consistency settings in Table 4 are mapped to the respective YCSB binding for Apache Cassandra and Couchbase.

The YCSB provides the performance metrics latency and throughput. As Q1 and Q2 are throughput related, the latency metrics are collected but not used for processing composed metrics. For addressing Q1, the scalability metric is computed by calculating the throughput increase with respect to the cluster size. For Q2, the throughput increase with respect to the defined different VM types and cluster sizes is computed to analyse the cloud resource and distribution impact.

Table 7: Workload Specifications

Specification	YCSB_Sensor_Workload
YCSB instances	1 (per cloud)
Threads (per instance)	16
Network	private
MAX_Runtime	1,800s
Number of records	4,000,000
Record size	5KB
Operations distribution	100% write
YCSB Binding	cassandra/couchbase2

5 MOWGLI EVALUATION

This section presents the results of the sensor storage scenario evaluation. The analysis of the results first focuses on the DBMS runtime centric Q1 by analysing performance and scalability and second on the cloud resource centric Q2 by analysing the impact of different VM types and cluster sizes.

For Q1, the DBMS specifications Spec_CA and Spec_CB in combination with the cloud resource configurations OS_SSD, OS_REMOTE and EC2_REMOTE and the YCSB_Sensor_Workload results in 72 evaluation scenarios specifications. The Q2 results are based on the DBMS specifications Spec_CA and Spec_CB with the cloud resource configurations OS_SSD_MAX, OS_REMOTE_MAX and the YCSB_Sensor_Workload, resulting in 18 evaluation scenarios specifications.

As Mowgli allows to specify the repetition of each scenario execution for $n \in \mathbb{N}$ times, we configure Mowgli to execute each scenario five times to verify the automated repeatability and to strengthen the significance of the results by providing the standard deviation as well as the minimum and maximum values. The runtime of a single evaluation is limited to 30 minutes, which is allows to the DBMS to stabilize and execute internal compaction processes. Likewise, Mowgli allows to specify custom runtime settings.

From system monitoring we ensure that the following properties hold for all evaluations: (1) The workload generator is not a bottleneck as the CPU load never exceeds 60%. (2) The network between the workload generator and DBMS cluster is not becoming a bottleneck, as the consumed network bandwidth is below the evaluated maximum available bandwidth. (3) The workload generator creates sufficient load to saturate the CPU resources of at least the 3-node clusters, *i.e.* the average CPU load of each node is > 90%.

The following sections present the throughput results as the average throughput over all five executions including standard deviation as well as global minimum and global maximum over all executions.

5.1 Q1 - DBMS Performance and Scalability

In the following, the results of Q1 are analysed for the concrete evaluation domain properties: "Which DBMS $D_{CA,CB}$ achieves the highest throughput for workload W_{YCSB_Sensor} if operated with with cluster size $CS_{3,5,7,9}$, replication factor RF_3, and ensures the write-consistency of $WC_{low,medium,high}$ by running on VM type VM_{small} in cloud $C_{OS\_SSD,OS\_REMOTE,EC2\_REMOTE}$?"

We group the results by cloud type (OS_SSD, OS_REMOTE, EC2_REMOTE). For each cloud type, we discuss the performance impact of cluster size and write consistency. The scalability is analysed computing the average throughput increase from the 3-node cluster to the 9-node cluster whereby the average throughput of the 3-node cluster represents the baseline.

5.1.1 OpenStack SSD Results. The Apache Cassandra results depicted in Figure 4 show that write consistency only has a slight impact on the performance for all cluster sizes. It is surprising that *ANY* as the weakest consistency provides less throughput than *ONE* for the 5-7-9-node clusters. With respect to the scalability, the throughput scales with growing cluster sizes, *e.g.* a scale-up of 19% is achieved from a 3-9 node cluster with write consistency ONE as listed in Table 8. The highest scalability factor of 31% is achieved for the write consistency TWO.

The results for Couchbase depicted in Figure 5 show significant differences depending on the applied write consistency. While the NONE configuration (not providing any guarantees at all) for a 3-node cluster achieves only 5% less throughput compared to 9-node Apache Cassandra cluster, we see massive drops in throughput when applying R-ONE and P-ONE. For R-ONE, the throughput for the 3-node cluster drops by 62% and for the 9-node cluster by 66% compared to NONE. For P-ONE it decreases by 92% compared to NONE and by 80% compared to R-ONE for the 3-node cluster. With respect to scalability, Table 8 shows that Couchbase achieves

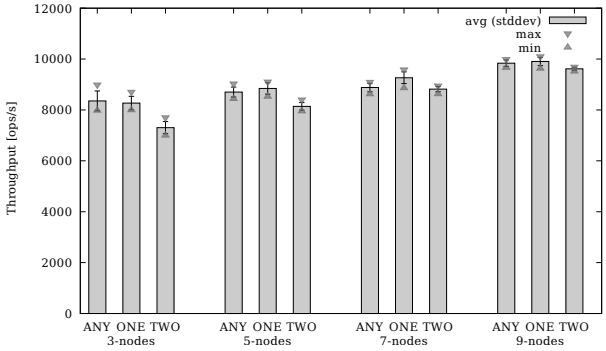


Figure 4: Q1 - Cassandra - OpenStack SSD Storage

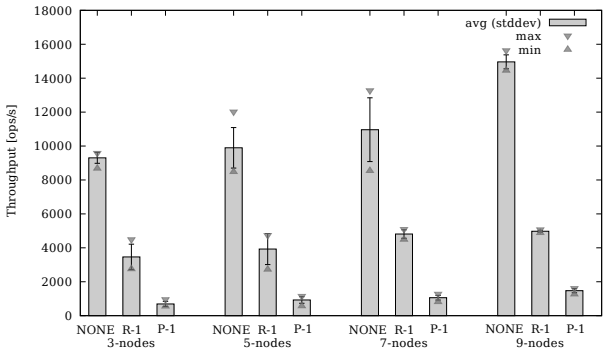


Figure 5: Q1 - Couchbase - OpenStack SSD Storage

a scale-up from 3–9 nodes for the write consistency NONE of 60%, for R-1 of 43% and P-1 of 113%.

Table 8: Scalability - OpenStack SSD Storage

Apache Cassandra				
Consistency	3-nodes	5-nodes	7-nodes	9-nodes
ANY	100%	+4%	+6%	+17%
ONE	100%	+6%	+12%	+19%
TWO	100%	+11%	+20%	+31%
Couchbase				
Consistency	3-nodes	5-nodes	7-nodes	9-nodes
NONE	100%	+6%	+17%	+60%
R-1	100%	+13%	+38%	+43%
P-1	100%	+33%	+53%	+113%

5.1.2 OpenStack Remote Storage Results. The evaluation scenarios on OpenStack with remote storage also comprise 3–9 node. Yet, the use of remote storage makes expect that the overall performance of a write-heavy workload will suffer due to concurrent use of storage.

The graphs for Apache Cassandra depicted in Figure 6 show indeed less throughput than for the SSD case. It also shows that a

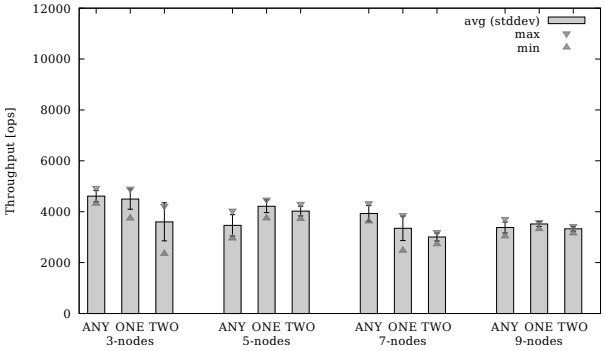


Figure 6: Q1 - Cassandra - OpenStack Remote Storage

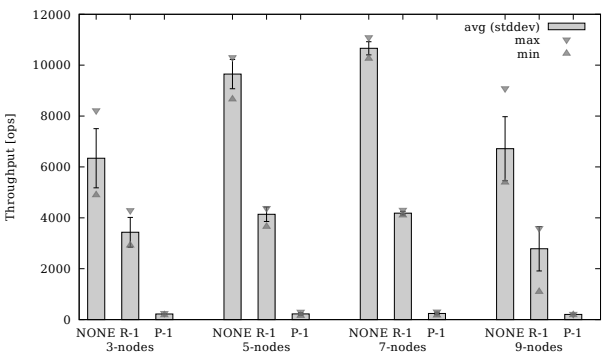


Figure 7: Q1 - Couchbase - OpenStack Remote Storage

larger cluster size does not improve the throughput because the single remote storage server represents the shared resource and results in a bottleneck (*cf.* Section 4.2). Increasing the cluster size from 3 to 9 nodes even results in a scale-up of -27% for write consistency ANY as listed in Table 9.

Also Couchbase depicted in Figure 7 achieves less throughput for all write consistency levels compared to the SSD case. Even though it uses asynchronous writes and no WAL, the write rate is limited and outstanding writes decrease throughput. While, from 3-7 nodes Couchbase still achieves a scale-up for *NONE* and *R-ONE* as the cache size and number of disk writers increases with the number of nodes as listed in Table 9, for the 9-node cluster the scalability factor is negative and the variance in the results increases. For *P-ONE*, the throughput stays on a constant level of 230 ops/s for 3-9 nodes.

5.1.3 EC2 Remote Storage Results. The EC2 results of Cassandra (*cf.* Figure 8) show the analogue performance impact of the write consistency as for the OpenStack results, *i.e. ANY* and *ONE* are in similar ranges where *TWO* results in 10% less throughput compared to ONE, independent of the cluster size. While the provisioned EC2 VMs use remote storage, the results show a clear scale-up from 3 to 9 nodes, *e.g.* 90% for *ONE* as shown in Table 10. Hence, the EC2 remote storage infrastructure does not impose the same bottleneck as OS_REMOTE. The Couchbase results, depicted in

Table 9: Scalability - OpenStack Remote Storage

Apache Cassandra				
Consistency	3-nodes	5-nodes	7-nodes	9-nodes
ANY	100%	-25%	-15%	-27%
ONE	100%	-7%	-26%	-22%
TWO	100%	+11%	-17%	-8%
Couchbase				
Consistency	3-nodes	5-nodes	7-nodes	9-nodes
NONE	100%	+52%	+68%	5%
R-1	100%	+20%	+21%	-10%
P-1	100%	+1%	+9%	-8%

Table 10: Scalability - EC2 Remote Storage

Apache Cassandra				
Consistency	3-nodes	5-nodes	7-nodes	9-nodes
ANY	100%	+49%	+79%	+85%
ONE	100%	+54%	+91%	+90%
TWO	100%	+56%	+82%	+92%
Couchbase				
Consistency	3-nodes	5-nodes	7-nodes	9-nodes
NONE	100%	+49%	+71%	+76%
R-1	100%	-3%	-39%	-24%
P-1	100%	+20%	+43%	+68%

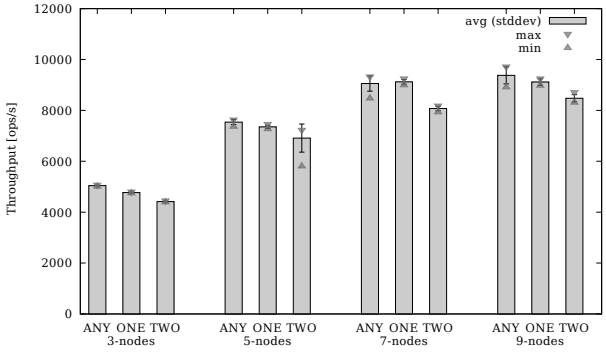


Figure 8: Q1 - Cassandra - EC2 Remote Storage

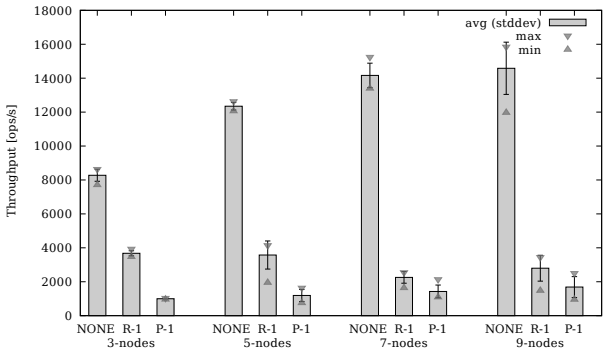


Figure 9: Q1 - Couchbase - EC2 Remote Storage

Figure 9, verify the findings that the EC2 remote storage does not impose a bottleneck. For P-ONE, we see a scale-up of 68% from 3 to 9 nodes. Further, as in the OS cases, Couchbase shows a significant drop in performance with higher consistency levels.

5.1.4 Comparative DBMS Analysis. Comparing both DBMSs, Apache Cassandra achieves better throughput if strong write consistency is required. Couchbase achieves the highest throughput in total if the weakest write consistency NONE is applied, while for R-ONE the throughput is constantly lower (OpenStack SSD, EC2) or similar (OpenStack Remote) to Apache Cassandra; P-ONE constantly achieves lower throughput than Apache Cassandra. Hence, Apache Cassandra should be preferred if write consistency is required while Couchbase should be preferred if maximum throughput is required and data inconsistency or (partial) data loss is tolerable.

With respect to scalability, both DBMSs scale with increasing cluster sizes, if there is no bottleneck on the cloud resource level. Yet, the scale-up degree depends heavily on the applied DBMS runtime configurations and cloud resource configurations.

With respect to the cloud resource configuration, Apache Cassandra achieves better throughput with SSD storage backends as WAL generates synchronous I/O for each write operation. In contrast, in the case of Couchbase, the applied storage backend affects the results of NONE and R-ONE only secondary. Consequently, only the throughput of P-ONE seems to be correlated to the storage.

5.2 Q2 - Cloud Resource Allocation and Distribution Impact

In the following, we analyse the results of Q2 for the concrete evaluation domain properties:

"Which cluster size $CS_{3,6,9}$ of DBMS D_{CA} achieves the highest throughput for workload W_{YCSB_Sensor} if operated with replication factor RF_3, ensuring write-consistency $WC_{low,medium,high}$ and running on VM types $VM_{small,medium,large}$ in cloud C_{OS_SSD,OS_REMOTE} if the maximum number of available cloud resources is $CR - MAX_{18\ vCores,36GB\ RAM}$?"

The results are grouped by OS_SSD_Q2 and OS_REMOTE_Q2 as listed in Table 6. For each cloud type, we discuss the impact of cluster size in correlation to the VM type and storage backend.

5.2.1 OpenStack SSD Results. The results depicted in Figure 10 show that the 3-node cluster on large VMs achieve the highest throughput. These results are expected as larger cluster sizes require additional network and coordination operations. Yet, the three node cluster also shows the highest throughput variance which indicates potentially suboptimal placement of the large VMs on the same physical server or interfering load of other VMs. Yet, an analysis would require the correlation of physical server monitoring with the Mowgli results, which is currently not supported and depends on provider specific monitoring information.

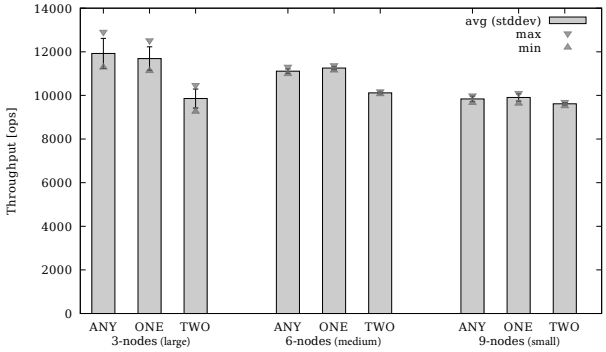


Figure 10: Q2 - Cassandra - SSD Storage

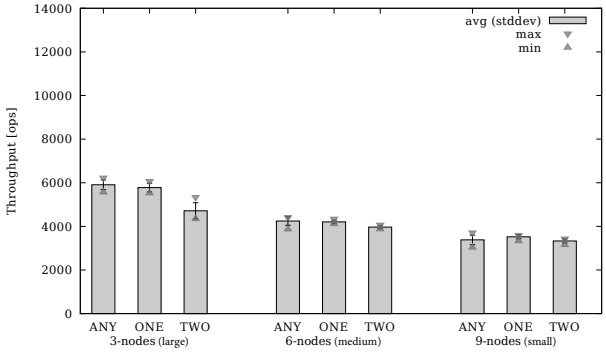


Figure 11: Q2 - Cassandra - Remote Storage

The relative throughput impact of the selected VM type and cluster size is listed in Table 11 where the average throughput of the 3-node cluster with VM type large represents the baseline. It is noteworthy that the throughput for write consistency TWO can even be increased by using more and less powerful nodes (*e.g.* 6-node cluster on VM type medium), which increases the internal writer threads. This indicates that the write performance configurations of the vanilla Apache Cassandra installation are underprovisioned for the large VM size and provide optimisation capabilities.

Table 11: SSD Storage & Distribution Impact

	Apache Cassandra		
Consistency	3-nodes large	6-nodes medium	9-nodes small
ANY	100%	-7%	-18%
ONE	100%	-4%	-16%
TWO	100%	+2%	-3%

5.2.2 OpenStack Remote Results. Figure 11 depicts the throughput of Apache Cassandra with respect to the three specified VM types and the remote-storage backend. The results clearly show that larger cluster sizes decrease the throughput as the shared usage of the remote-storage imposes a bottleneck. The relative throughput decrease is listed in Table 12 where the average throughput of the 3-node cluster represents the baseline. In addition, these results verify the negative scale-up results for Apache Cassandra (*cf.* Section 5.1.1).

Table 12: Remote Storage & Distribution Impact

	Apache Cassandra		
Consistency	3-nodes large	6-nodes medium	9-nodes small
ANY	100%	-29%	-43%
ONE	100%	-28%	-40%
TWO	100%	-16%	-30%

5.2.3 Comparative Resource Allocation Analysis. Comparing the different VM types and storage backends first shows that allocating larger VM types with small cluster sizes provides better throughput than large clusters of small VMs due to additional communication and coordination overhead. Yet, the latter can improve the availability in case of physical hardware failure if the VMs are distributed equally across the physical infrastructure. Mowgli eases the determination of the performance versus availability tradeoff. Second, as cloud resources are typically shared amongst multiple tenants, interferences can impact the performance or even limit the scalability as shown for the remote storage in the Ulm OpenStack. Mowgli enables the extensive evaluation of cloud resources for the operation of DBMS to identify potential bottlenecks and interferences.

6 DISCUSSION

Within this section, first we discuss the advantages and limitations of Mowgli in order to answer Q1, Q2 and similar questions based on the introduced requirements towards a multi-domain evaluation framework (*cf.* Section 2.4). Second, we validate the significance of our results by comparing them to related evaluation results and discuss how Mowgli can improve their portability and reproducibility.

6.1 Mowgli Feature Analysis

Ease of use (R1): While Mowgli comprises multiple loosely coupled services, its deployment of all five components is automated via Docker[16] and its configuration is based on six parameters. Mowgli provides a simple graphical as well as a REST-based interface. With the presented evaluation results we verified the usability of Mowgli for DBMS runtime and cloud resource specific evaluation scenarios. The specification of the sensor storage scenario comprises 16 template properties, separated by four DBMS properties, five cloud resource properties and seven workload properties[17]. Our experiences show that undergraduate students can be taught to use Mowgli in the order of less than a day.

Portability (R2): In the context of answering Q1 and Q2, the sensor storage scenario is applied to two DBMS, two cloud providers, three different storage backends and three VM types, which shows

[16] https://omi-gitlab.e-technik.uni-ulm.de/mowgli/docker
[17] Excluding YCSB binding specific properties for Apache Cassandra/Couchbase

that Mowgli eases the *evaluation portability* between different DBMS or cloud resources. While the DBMS catalogue abstracts the deployment of the DBMS, still thorough DBMS-specific knowledge is required to extend the DBMS catalogue. Similarly, extending cloud resource templates or adding new cloud providers requires knowledge of the Cloudiator framework.

Reproducibility (R3): Using our existing templates[3], the deterministic *reproduction* of our validation scenario on any of the supported DBMS or cloud providers is a matter of minutes.

Automation (R4): DBMS fully automates the evaluation process based on customizable evaluation tasks by orchestrating the cloud resource allocation, DBMS deployment, workload execution, system monitoring and releasing cloud resources. It automates the collection of workload-specific performance metrics as well as the system and DBMS metrics during the evaluation execution. It also provides advanced processing and visualization support for the sensor storage scenario. Custom processing and visualization task are supported by implementing new objective processors.

Significance (R5): In its current state, Mowgli supports two major cloud providers, *i.e.* EC2 and Google Compute as well as OpenStack for private clouds. With respect to the DBMS domain, five common DBMS are supported (*cf.* Section 3.2). Regarding the workload domain, the YCSB enables comparable evaluation scenarios by simple synthetic workloads. In order to enable more in-depth evaluation scenarios for DBMS-specific features, additional workloads such as TPC-C, HTAP or trace-based workloads need to be integrated into Mowgli. As a first step into this direction, a preliminary TPC-C workload implementation has been integrated.

Extensibility (R6): As outlined in Section 3, Mowgli builds upon loosely coupled components, which interact via REST-based interfaces. Hence, the framework is prepared for extending dedicated components, *e.g.* the workload-API with additional workloads or extending supported DBMS in the DBMS catalogue. In order to add a new evaluation scenario, the evaluation orchestrator needs to be extended by (1) defining a new evaluation workflow by building upon the existing tasks or by implementing new ones; (2) defining a new evaluation scenario template by building upon the existing DBMS configurations and cloud resource templates; (3) implementing the mapping from the scenario template to the scenario workflow. Consequently, thorough domain specific knowledge is required as the Mowgli components can only provide the conceptual technical abstraction but the domain specific commands are still required. The extensibility of Mowgli has been demonstrated by extending the current evaluation scenario (targeting *performance* and *scalability*) to elasticity [37] and availability [36].

6.2 Evaluation Result Verification

With the growing impact of distributed DBMSs, performance and scalability evaluations are a widely addressed research topic. Consequently, we compare existing DBMS evaluation results with our results to validate their correctness with respect to Q1 and Q2. Hereby, we only consider research publications and no white papers due to their questionable scientific neutrality. Further, we only select results for Apache Cassandra and Couchbase that evaluate the scalability of different DBMS cluster sizes and rely on the YCSB as workload. Optionally, the results are created on cloud resources. Several published results evaluate the performance of Apache Cassandra and Couchbase with the YCSB [2, 19, 23, 24, 38, 39]. Yet, only a few evaluate their scalability based on different cluster sizes [11, 30] and by using cloud resources [26, 37], which consolidates the need for Mowgli in order ease the DBMS evaluation by portable and reproducible evaluation scenarios. In the following these results are analysed and compared to our results in chronological order.

An evaluation of the early version 0.5.0 of Apache Cassandra has been conducted with the initial YCSB [11]. The evaluations are carried out on a proprietary private cloud middleware and with Cassandra cluster sizes from 2 to 12 nodes. While the results are based on read-heavy and read-update workloads, they also verify the scalability of Cassandra with growing cluster sizes.

The result of [30] execute a write-heavy YCSB workload against 2 to 14 Cassandra nodes on physical hardware. Similar to our results, Cassandra shows a throughput increase with growing cluster sizes. Yet, the results of [30] show a nearly linear scalability of Apache Cassandra which due to disabled replication and scaling YCSB client instances and threads relative to the cluster size. Hence, the Cassandra cluster is always saturated while in our scenario a constant workload is applied, which saturates only a 3-node cluster. Yet, our presented evaluation scenario can easily be adapted to scale the workload in relation to the cluster size.

The previous evaluations [30] are reproduced by [26], replacing the physical resources with cloud resources on EC2 with remote storage. Similar to [30], the results show a nearly linear scalability of Cassandra by increasing the workload relative to the cluster size, which verifies our Apache Cassandra results on EC2. In addition, [26] evaluate the performance impact based on the selected cloud storage backend configurations, which accompanies our results with respect to the SSD and remote storage results. In this context, [26] emphasize the need but also the complexity to evaluate different cloud resource configurations.

In a preliminary version of Mowgli, the scalability of Cassandra and Couchbase was evaluated by read-heavy and read-update workloads by using one VM type and one storage backend [37] on a dedicated host in the OpenStack cloud at Ulm. The results confirm the scalability of Couchbase and Cassandra with growing cluster sizes. Yet, the results are carried out without replication and the lowest consistency settings. The impact of VM resource configurations including storage backends have not been analysed.

The comparative analysis of existing evaluation results, verifies the significance of Mowgli as the scalability of Apache Cassandra and Couchbase is verified. In addition, the impact of the selected storage backend is confirmed. Yet, the analysis also shows that reproducing existing evaluations is a time consuming and error prone task, as the required domain properties might not be documented or have changed over time. This also limits the portability as existing evaluations do not provide any abstraction of the evaluation domains. Hence, porting existing evaluation results becomes a challenging task [26]. Therefore, Mowgli enables the reproducible and portable evaluation execution for multi-domain scenarios.

7 RELATED WORK

Since the era of RDBMS, their selection is guided by domain-specific *benchmarks* that have evolved together with distributed DBMSs.

The need for portable and reproducible evaluations then led to the integration of existing benchmarks into *evaluation frameworks*, which extend the sole workload generation by DBMS runtime features to cover more complex evaluation domains [7, 35].

7.1 Benchmarks

Benchmarks are applied to evaluate non-functional features of DBMSs by artificial or trace-based workloads, producing evaluation metrics [21]. While traditional DBMS benchmarks mainly target the performance, recent benchmarks also target non-functional features of distributed DBMSs, such as scalability, elasticity, consistency and availability [35]. The workload domain of DBMS benchmarks is distinguished between OLTP, Online Analytical Processing (OLAP) and the recently evolving HTAP.

Performance benchmarks of the OLTP and OLAP domains for the relational data model are provided by the transaction performance council (TPC)[18], namely TPC-C[19] and TPC-H[20]. Building upon these, HTAPBench enables HTAP workloads for the relational data model [10]. For the NoSQL data models, the YCSB [11] is widely used for performance and scalability benchmarks. Advancements of YCSB such as YCSB+T and YCSB++ focus on the performance of transactions in NoSQL models [13] and on the consistency of distributed NoSQL DBMSs [28] respectively. While the YCSB workloads target artificial CRUD operations, web-application workloads are presented by OLTP-Bench [14] and BG [3]. BenchFoundry [6] presents a trace-based workload generator for realistic workloads.

Existing benchmarks cover a variety of workload domains, which enables significant DBMSs evaluations. Hence, our framework does not focus on the definition of a new benchmark rather than on integrating existing benchmarks to enable DBMS runtime-driven and resource-driven evaluation scenarios.

7.2 Cloud-centric DBMS Evaluations

A multitude of modern DBMS evaluations has been conducted within the recent years based on existing benchmarks. Yet, only a subset of these evaluations focus on cloud-related aspects. Originally, YCSB evaluated the performance and scalability of Apache Cassandra, Apache HBase and Yahoo PNUTS in Yahoo's data center [11]; yet, only for read- and update-heavy workloads running on a static pool of physical resources. Building upon YCSB, cloud-centric evaluations have been conducted: [26] focus on the scalability and elasticity of Apache Cassandra and HBase for different cluster sizes on Amazon EC2 with different remote storage backends. Also [24] build upon fixed EC2 resources for evaluating the performance impact of different consistency configurations for Apache Cassandra, MongoDB and Riak. A private OpenStack cloud is used by [37] to evaluate the scalability and elasticity of Apache Cassandra, Couchbase and MongoDB under varying workload sizes.

While these results prove the scalability of Apache Cassandra and Couchbase with respect to read-heavy and read-update workloads, the scalability of write-heavy workloads has not been evaluated, especially with respect to different cloud resource offerings and storage backends. Yet, even as these evaluation results provide a thorough technical explanation, their reproducibility is limited and error-prone due to the complexity of the involved domains, *i.e.* cloud computing, distributed DBMSs and benchmarks. Hence, [26, 37] highlight the need for more sophisticated DBMS evaluation to ensure reproducibility, portability and significance.

7.3 Evaluation Frameworks

While the portability and reproducibility of evaluation results has been emphasized for a long time [21], its compliance becomes even more challenging with the evolving technologies. Hence, building only upon benchmarks for distributed DBMSs in the cloud is not sufficient to enable reproducible, portable and comparable results which take into account the runtime configurations [7, 35]. Therefore, evaluation frameworks need to provide additional features such as evaluation orchestration, resource abstraction and the specification of portable evaluation scenarios [7, 35].

[25] presents an evaluation framework that builds upon the YCSB and enables the evaluation of Amazon's DBaaS offerings and Apache Cassandra with a focus on scalability and the performance impact of different consistency configurations. Yet, the framework does not abstract the DBMSs deployment and the cloud resource offerings. Hence, the framework is not supporting portable evaluation scenarios and cannot be applied to different cloud providers and cloud resources. A cloud-resource centric framework is presented by [9], which provisions cloud resources, orchestrates applications, executes generic micro-benchmarks and monitors the execution performance. While this framework focuses on evaluating cloud resources based on resource-specific micro-benchmarks, DBMS evaluation and cloud resources for DBMS are not in its scope.

Hence, current evaluation frameworks either focus on the benchmark execution and DBMS runtime configuration or on the cloud resource benchmarking in general, Mowgli combines both aspects and automates the full evaluation execution. This enables the definition and execution of portable and comparable evaluation scenarios for different cloud resources and DBMSs.

8 CONCLUSION

Big Data and IoT demand for distributed and scalable database management systems (DBMS). Cloud resources provide these scalability demands on the resource level. Yet, operating a DBMS in the cloud is a challenging task, due to immense number of DBMSs and cloud resource offerings. While DBMS evaluation guides this task, current approaches do not consider DBMS runtime and cloud resource constraints, limiting evaluation portability and reproducibility. Hence, we present Mowgli that enables portable and reproducible evaluations in a consistent manner. Mowgli provides abstract evaluation scenario templates, which are mapped to DBMS runtime and cloud resource configurations and executed automatically by allocating cloud resources, deploying DBMS cluster, executing the workload and adapting the DBMS cluster.

We evaluate the usability of Mowgli by applying an IoT-driven evaluation scenario for Apache Cassandra and Couchbase that first focusses on their performance and scalability with respect to the runtime constraints cluster size and write consistency and second, focuses on the impact of the allocated cloud resources in correlation to the cluster size. Both DBMSs are evaluated on three resource

[18] http://www.tpc.org/information/benchmarks.asp
[19] http://www.tpc.org/tpcc/default.asp
[20] http://www.tpc.org/tpch/default.asp

configurations comprising Amazon EC2 and a private OpenStack. The executed 102 evaluation scenarios verify the portability and reproducibility by Mowgli and allow the correlation between DBMS runtime constraints and cloud resources. Both DBMSs show a scale-up for growing cluster sizes and the performance of Cassandra correlates with the applied storage while the performance of Couchbase heavily depends on the applied write consistency level. The significance of Mowgli is verified by evaluating its features against established requirements of DBMS evaluation and by comparing the collected results against existing evaluation results.

While Mowgli provides a powerful tool to guide the way through the DBMS jungle, a holistic DBMS recommendation system building upon machine learning and artificial intelligence is subject to ongoing work. Furthermore, availability evaluation scenarios emulating cloud resource failures and sudden workload peaks are ongoing work. With respect to the workload domain, ongoing work investigates into hybrid transaction-analytical processing workloads and their impact on the DBMS runtime and cloud resource constraints.

Acknowledgements

We thank Eddy Truyen of the KU Leuven for his constructive feedback on earlier versions of the paper. The research leading to these results has received funding from the EC's Framework Programme HORIZON 2020 under grant agreements 731664 (MELODIC) and 732667 (RECAP). We also thank the Daimler TSS for their valuable and constructive discussions and the funding for parts of the work.

REFERENCES

[1] Daniel Abadi, Rakesh Agrawal, Anastasia Ailamaki, Magdalena Balazinska, Philip A Bernstein, Michael J Carey, Surajit Chaudhuri, Jeffrey Dean, AnHai Doan, Michael J Franklin, et al. 2016. The Beckman report on database research. *Commun. ACM* 59, 2 (2016), 92–99.

[2] Veronika Abramova, Jorge Bernardino, and Pedro Furtado. 2014. Which nosql database? a performance overview. *Open Journal of Databases (OJDB)* 1, 2 (2014), 17–24.

[3] Sumita Barahmand and Shahram Ghandeharizadeh. 2013. BG: A Benchmark to Evaluate Interactive Social Networking Actions.. In *CIDR*.

[4] D. Baur, D. Seybold, F. Griesinger, H. Masata, and J. Domaschka. 2018. A Provider-Agnostic Approach to Multi-cloud Orchestration Using a Constraint Language. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 173–182. https://doi.org/10.1109/CCGRID.2018.00032

[5] Daniel Baur, Daniel Seybold, Frank Griesinger, Athanasios Tsitsipas, Christopher B Hauser, and Jörg Domaschka. 2015. Cloud Orchestration Features: Are Tools Fit for Purpose?. In *IEEE/ACM UCC*.

[6] David Bermbach, Jörn Kuhlenkamp, Akon Dey, Arunmoezhi Ramachandran, Alan Fekete, and Stefan Tai. 2017. BenchFoundry: A Benchmarking Framework for Cloud Storage Services. In *International Conference on Service-Oriented Computing*. Springer, 314–330.

[7] David Bermbach, Jörn Kuhlenkamp, Akon Dey, Sherif Sakr, and Raghunath Nambiar. 2014. Towards an extensible middleware for database benchmarking. In *TPCTC*.

[8] Rick Cattell. 2011. Scalable SQL and NoSQL data stores. *Acm Sigmod Record* 39, 4 (2011), 12–27.

[9] Ryan Chard, Kyle Chard, Bryan Ng, Kris Bubendorfer, Alex Rodriguez, Ravi Madduri, and Ian Foster. 2016. An automated tool profiling service for the cloud. In *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on*. IEEE, 223–232.

[10] Fábio Coelho, João Paulo, Ricardo Vilaça, José Pereira, and Rui Oliveira. 2017. HTAPBench: Hybrid Transactional and Analytical Processing Benchmark. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. ACM, 293–304.

[11] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *ACM SoCC*.

[12] Ali Davoudian, Liu Chen, and Mengchi Liu. 2018. A Survey on NoSQL Stores. *ACM Computing Surveys (CSUR)* 51, 2 (2018), 40.

[13] Akon Dey, Alan Fekete, Raghunath Nambiar, and Uwe Rohm. 2014. YCSB+T: Benchmarking web-scale transactional databases. In *IEEE ICDEW*.

[14] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudre-Mauroux. 2013. Oltp-bench: An extensible testbed for benchmarking relational databases. *VLDB Endowment* (2013).

[15] Jörg Domaschka, Daniel Baur, Daniel Seybold, and Frank Griesinger. 2015. Cloudiator: a cross-cloud, multi-tenant deployment and runtime engine. In *9th Symposium and Summer School on Service-Oriented Computing*.

[16] Jörg Domaschka, Christopher B Hauser, and Benjamin Erb. 2014. Reliability and availability properties of distributed database systems. In *Enterprise Distributed Object Computing Conference (EDOC), 2014 IEEE 18th International*. IEEE, 226–233.

[17] Thibault Dory, Boris Mejías, PV Roy, and Nam-Luc Tran. 2011. Measuring elasticity for cloud databases. In *Proceedings of the The Second International Conference on Cloud Computing, GRIDs, and Virtualization*.

[18] Michael Galloway, Gabriel Loewen, Jeffrey Robinson, and Susan Vrbsky. 2018. Performance of Virtual Machines using Diskfull and Diskless Compute Nodes. (2018). https://doi.org/10.1109/CLOUD.2018.00101

[19] Andrea Gandini, Marco Gribaudo, William J Knottenbelt, Rasha Osman, and Pietro Piazzolla. 2014. Performance evaluation of NoSQL databases. In *European Workshop on Performance Engineering*. Springer, 16–29.

[20] Felix Gessert, Wolfram Wingerath, Steffen Friedrich, and Norbert Ritter. 2017. NoSQL database systems: a survey and decision guidance. *Computer Science-Research and Development* 32, 3-4 (2017), 353–365.

[21] Jim Gray. 1992. *Benchmark handbook: for database and transaction processing systems*.

[22] Katarina Grolinger, Wilson A Higashino, Abhinav Tiwari, and Miriam AM Capretz. 2013. Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing: Advances, Systems and Applications* 2, 1 (2013), 22.

[23] Abdullah Talha Kabakus and Resul Kara. 2017. A performance evaluation of in-memory databases. *Journal of King Saud University-Computer and Information Sciences* 29, 4 (2017), 520–525.

[24] John Klein, Ian Gorton, Neil Ernst, Patrick Donohoe, Kim Pham, and Chrisjan Matser. 2015. Performance evaluation of NoSQL databases: a case study. In *Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems*. ACM, 5–10.

[25] Markus Klems, David Bermbach, and Rene Weinert. 2012. A runtime quality measurement framework for cloud database service systems. In *Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the*. IEEE, 38–46.

[26] Jörn Kuhlenkamp, Markus Klems, and Oliver Röss. 2014. Benchmarking scalability and elasticity of distributed database systems. *Proceedings of the VLDB Endowment* 7, 12 (2014), 1219–1230.

[27] Avinash Lakshman and Prashant Malik. 2010. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review* 44, 2 (2010), 35–40.

[28] Swapnil Patil, Milo Polte, Kai Ren, Wittawat Tantisiriroj, Lin Xiao, Julio López, Garth Gibson, Adam Fuchs, and Billie Rinaldi. 2011. YCSB++: benchmarking and performance debugging advanced features in scalable table stores. In *ACM SoCC*.

[29] Mark Raasveldt, Pedro Holanda, Tim Gubner, and Hannes Mühleisen. 2018. Fair Benchmarking Considered Difficult: Common Pitfalls In Database Performance Testing. In *Proceedings of the Workshop on Testing Database Systems*. ACM, 2.

[30] Tilmann Rabl, Sergio Gómez-Villamor, Mohammad Sadoghi, Victor Muntés-Mulero, Hans-Arno Jacobsen, and Serge Mankovskii. 2012. Solving big data challenges for enterprise application performance management. *Proceedings of the VLDB Endowment* 5, 12 (2012), 1724–1735.

[31] Vincent Reniers, Dimitri Van Landuyt, Ansar Rafique, and Wouter Joosen. 2017. On the state of nosql benchmarks. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. ACM, 107–112.

[32] Pramod J Sadalage and Martin Fowler. 2013. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education.

[33] Sherif Sakr. 2014. Cloud-hosted databases: technologies, challenges and opportunities. *Cluster Computing* 17, 2 (2014), 487–502.

[34] Daniel Seybold. 2017. Towards a framework for orchestrated distributed database evaluation in the cloud. In *Proceedings of the 18th Doctoral Symposium of the 18th International Middleware Conference*. ACM, 13–14.

[35] Daniel Seybold and Jörg Domaschka. 2017. Is Distributed Database Evaluation Cloud-Ready?. In *Advances in Databases and Information Systems*. Springer, 100–108.

[36] Daniel Seybold, Christopher B Hauser, Simon Volpert, and Jörg Domaschka. 2017. Gibbon: An Availability Evaluation Framework for Distributed Databases. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 31–49.

[37] Daniel Seybold, Nicolas Wagner, Benjamin Erb, and Jörg Domaschka. 2016. Is elasticity of scalable databases a Myth?. In *IEEE Big Data*.

[38] Enqing Tang and Yushun Fan. 2016. Performance comparison between five NoSQL databases. In *Cloud Computing and Big Data (CCBD), 2016 7th International Conference on*. IEEE, 105–109.

[39] Jan Sipke Van der Veen, Bram Van der Waaij, and Robert J Meijer. 2012. Sensor data storage performance: SQL or NoSQL, physical or virtual. In *Cloud computing (CLOUD), 2012 IEEE 5th international conference on*. IEEE, 431–438.

Chapter 13

[core6] Kaa: Evaluating Elasticity of Cloud-Hosted DBMS

This article is published as follows:

Daniel Seybold, Simon Volpert, Stefan Wesner, André Bauer, Nikolas Herbst, and Jörg Domaschka. “Kaa: Evaluating Elasticity of Cloud-Hosted DBMS” in *11th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2019, IEEE, pp. 54–61, DOI: https://doi.org/10.1109/CloudCom.2019.00020.

Kaa: Evaluating Elasticity of Cloud-hosted DBMS

Daniel Seybold, Simon Volpert, Stefan Wesner
Institute of Information Resource Management
Ulm University, Germany
firstname.lastname@uni-ulm.de

André Bauer, Nikolas Herbst
Informatik II
University of Würzburg, Germany
firstname.lastname@uni-wuerzburg.de

Jörg Domaschka
Inst. of Inf. Resource Management
Ulm University, Germany
joerg.domaschka@uni-ulm.de

***Abstract*—Auto-scaling is able to change the scale of an application at runtime. Understanding the application characteristics, scaling impact as well as the workload, an auto-scaler aligns the acquired resources to match the current workload. For distributed Database Management Systems (DBMS) forming the backend of many large-scale cloud applications, it is currently an open question to what extent they support scaling at run-time. In particular, elasticity properties of existing distributed DBMS are widely unknown and difficult to evaluate and compare. This paper presents a comprehensive methodology for the evaluation of the elasticity of distributed DBMS. On the basis of this methodology, we introduce a framework that automates the full evaluation process. We validate the framework by defining significant elasticity scenarios for a case study that comprises two DBMS for write-heavy and read-heavy workloads of different intensities. The results show that scalable distributed DBMS are not necessarily elastic and that adding more instances to a cluster at run-time may even decrease the experienced performance.**

***Index Terms*—elasticity, cloud, NoSQL, scalability, distributed DBMS**

I. Introduction

Auto-scaling exploits the cloud's on-demand resources to adapt application scale to the resource demands of the application's currently experienced workload [1]. Realising this kind of elasticity requires a scalable application architecture and the capability of the application to scale-out/-in at run-time [2]. For stateless applications, scaling at run-time is no issue and research has focussed on auto-scalers that determine the required application scale at a given time. For such scenarios, the quality of the achieved elasticity is of high importance [3].

For stateful applications such as Database Management Systems (DBMS), the NoSQL movement is promising scalability in return to giving up the ACID properties of relational DBMS, favouring distributed DBMS [4]. For this kind of DBMS, the capability to scale-out/-in at run-time is of importance: (a) to provide client-side performance guarantees for changing workload patterns; (b) with the growing amount of data generated DBMS need to grow over time as well; (c) any elastic application may reach a scale-out degree where the storage back-end becomes a bottleneck if not scaled as well; (d) when offering *database-as-a-service (DBaaS)* [5], cloud operators benefit from the ability to adapt the scale of their customers' DBMS instances according to workload; (e) better understanding of stateful applications will help to improve auto-scalers in wide-spread orchestration platforms such as Kubernetes.

While DBMS auto-scalers exist for very specific scenarios [6], [7], it is an open question to what extent existing DBMS support elasticity [8] for general workloads; similarly, the metrics needed to determine the quality of elasticity in different scenarios are widely undecided, but needed to compare DBMS [9]. Consequently, elasticity evaluations of cloud-hosted DBMS are strongly needed [4].

In accordance with established methods [10], [11], we claim that cloud-based evaluations need to increase the availability of data, the quality of data, and the flexibility and repeatability of experiments to mitigate the current situation and improve the understanding of the elasticity of distributed DBMS. Considering the enormous design space, only an automation framework is able to provide the necessary amount of data and ensure repeatability.

In consequence, this paper provides the following contributions: (i) a comprehensive methodology for the evaluation of distributed DBMS elasticity that builds on established metrics [12], [13]; (ii) an extension to our Mowgli evaluation framework [14] capable to evaluate elasticity based on above methodology. (iii) a validation of the framework by defining significant elasticity scenarios for a case study with two DBMS for write-heavy and read-heady workloads.

The remainder of this paper is structured as follows: Section II introduces the background on distributed DBMS and DBMS scalability and elasticity. Section III presents challenges with respect to DBMS elasticity evaluation, while in Section IV we discusses our Kaa evaluation framework. In Section V, we present a case study evaluating the elasticity of two DBMS under two workloads. We discuss the results in Section VI, before Section VII presents related work and Section VIII concludes.

II. Background and Terminology

This section summarises the background on distributed DBMS and presents the terminology we use in this paper, particularly with respect to scalability and elasticity.

A. Distributed Database Management Systems

The evolution of DBMS has led to an increasing heterogeneity in available DBMS. Currently they are separated into three top-level categories: *relational*, *NoSQL* and *NewSQL* [4], [15]. For being able to benefit from any type of horizontal scale-out, the DBMS needs to operate as a distributed DBMS. Distributed DBMS provide a logical DBMS instance to clients,

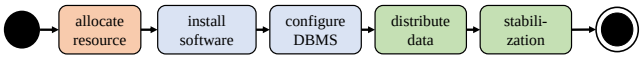


Fig. 1: Steps of an elastic scale-out

but distribute the DBMS functionality across multiple physical or virtual resource entities. These hosting DBMS nodes are connected via network and form a DBMS cluster.

Distributed DBMS exploit sharding as a basic technique: the full data set is separated such that each data-hosting DBMS node manages only a local share of the overall data set. With more nodes joining the cluster, the overall storage and memory capacity increases and the system scales horizontally. Ideally, with adding more nodes to the cluster, also the compute capacity increases so that also the throughput scales horizontally.

Due to the fact that an increased cluster size increases the probability of failures, distributed DBMS also provide means for replication. That means, a single data item is stored by multiple cluster members so that in case of single-node failures, one or more copies are still available. The consistency level of the data defines how eagerly replicas of data items are kept in sync with each other. Besides providing fault-tolerance and increasing availability, the use of replicas also allows to further increase the scaling of read operations, as multiple nodes host the item.

B. DBMS Elastic Scale-out Process

Most distributed DBMS support *elastic scale-out*, i.e. increasing the DBMS cluster at run-time without service interruption [16]. Figure 1 illustrates these steps: In the first step, a new resource is provided. In particular for cloud-hosted DBMS, this means acquiring a new virtual machine. Step two provisions the necessary software, including the installation of DBMS binaries. Step four sets the configuration files.

Afterwards, the new DBMS node joins the DBMS cluster. At that point, it is not able to handle client requests yet. This can only be done after the data set managed by the DBMS has been re-shareded and the new shards have been distributed over all cluster members. During this transition, additional compute and storage resources are used to exchange data amongst all nodes of the cluster. The step completes once the new node is able to answer client requests and hence, from a client-side perspective, appears as a regular cluster member. Internally, the DBMS may go through a stabilization phase [12] once the data shuffling has been completed.

All of the steps except for the first one differ from DBMS to DBMS. Among the remaining ones, software installation and configuration need to be performed from outside the DBMS. The distribution of data and the stabilization phase run automatically and usually cannot be influenced externally.

C. DBMS Scalability and Elasticity Metrics

Here, we define scalability and elasticity as used in this paper. By building upon established elasticity metrics [12], we define metrics related to these two aspects that build upon the traditional performance metrics storage capacity (gigabytes), throughput (requests per second) as well as latency (response time per request) [17].

Definition 1 (DBMS Horizontal Scalability)**.** *Horizontal scalability denotes the capability of a distributed application to increase its performance by increasing the cluster size.*

The remainder of this paper focusses on scalability with respect to throughput and latency. In particular, the performance improvement achieved from increasing the DBMS cluster size is defined by scale-out metrics:

Metric 1 (DBMS Scale-out)**.** *The Scale-out metric [12] correlates the performance of the DBMS with the DBMS cluster size for a given workload. Ideally, not only the cluster size, but the overall available resources need to be considered.*

Accordingly, the *scale-out factor* denotes the improvement in latency and throughput when changing the size of a DBMS cluster. A good scale-out property is indicated by a constant latency and a throughput increasing proportionally with the cluster sizes [12]. Yet, the Scale-up metric exclusively determines the performance difference between two cluster sizes. It does not capture the impact of scaling out a cluster at run-time and therefore ignores aspects related to elasticity.

Definition 2 (Elastic scale-out)**.** *An elastic scale-out is the change of a DBMS cluster size at run-time. Its impact is measured through the DBMS elasticity metrics data distribution impact, data distribution time, and provisioning time [12].*

Metric 2 (Provisioning time)**.** *This metric captures the time span required to provide a new computational resource, e.g. a virtual machine, and to install software on that resource. As such, it also reflects differences between cloud providers.*

Metric 3 (Data distribution impact)**.** *This metric captures the performance change during the data distribution phase of an elastic scale-out.*

Metric 4 (Data distribution time)**.** *This metric captures the duration of the data distribution phase.*

Conceptionally, similar metrics exist for the stabilisation phase. Yet, we are currently not aware how they can be precisely measured for any DBMS presented in this paper, so that we do not consider them in the following.

III. DBMS Elasticity Evaluation Challenges

The analysis of established DBMS benchmarks [18] shows that they only support client-side performance metrics, but lack the required support on the DBMS operator side. Due to that, we first define an elasticity evaluation process that includes the DBMS operator related tasks. Secondly, we identify requirements for enabling holistic DBMS elasticity evaluations.

A. Elasticity Evaluation Process

Figure 2 depicts our elasticity evaluation process with individual *evaluation tasks (ET)* defined in Table I. In contrast

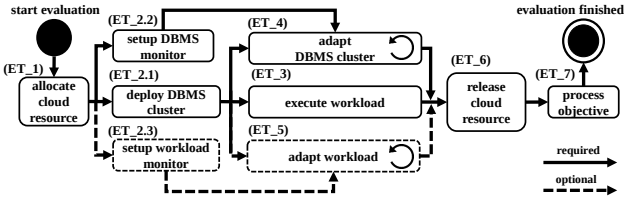


Fig. 2: Elasticity Evaluation Process

to DBMS workload tools, this process supports the operator-side by managing cloud resources and the DBMS cluster [18], [19]. In addition, the process handles workload issuing and client-side evaluation metrics.

In the first step of the evaluation process, the necessary resources are allocated (ET_1). ET_2 comprises the deployment of the workload generators and DBMS cluster on the allocated as well as the setup of the DBMS and workload monitoring systems. The workload is executed in ET_3. While the system is under test, adaptations can be applied for the DBMS cluster (ET_4) and the workload (ET_5).

Adaptations to the cluster size (ET_4) are a requirement for any type of elasticity evaluation. The DBMS adaptations in ET_4 can either be *time-based* or *metric-based*. In the first case, predefined timestamps are used as a trigger for executing the DBMS adaptation. Time-based triggers enable isolated elasticity evaluations for specific workload configurations, ensuring reproducibility. In the latter case, system and DBMS metrics are retrieved from the DBMS monitor (cf. ET_2.2) and combined into *DBMS scaling rules (DSR)*. DSR enable more realistic evaluations, but they require a thorough understanding of the DBMS utilization in order to apply reasonable metric compositions. Consequently, DSR neglect reproducibility in favour of realistic scenarios.

Adaptations to the workload (ET_5) are required to emulate fluctuating workloads. The trigger for these adaptations can be caused *time-based*, *progress-based*, and *metric-based*: Time-based triggers use predefined timestamps, progress-based triggers refer to the execution progress of ET_3, and composed metric-based triggers are termed *workload scaling rules (WSR)* and follow the same concept as DSR. If specified, DBMS and workload adaptations can be repeated multiple times within an elasticity evaluation to emulate realistic application scenarios of increasing and decreasing DBMS workload patterns [20].

TABLE I: Elasticity Evaluation Tasks

ET	Description
1	allocate new cloud resources for the evaluation
2.1	deploy and configure the DBMS cluster
2.2	setup system and DBMS monitoring
2.3	(optional) setup workload state monitoring
3	execute a baseline workload to measure the performance metrics
4	execute elastic scale-out/in based on predefined adaptation rules
5	(optional) adapt workload intensity
6	release the cloud resource
7	collect and process the elasticity results

B. Elasticity Evaluation Requirements

The evaluation process from Section III-A defines the evaluation tasks, but also increases the complexity compared to performance or even scalability evaluations, which already depend on many impact factors including DBMS, cloud resources, and workload domain [14]. For handling this complexity, a framework supporting automated elasticity evaluations needs to fulfil the following requirements:

An *elasticity evaluation scenario specification (R1)* that comprises all technical aspects to ensure the reproducibility and portability for adopters and subsequent evaluations. Therefore, it needs to contain the cloud resource, DBMS runtime and workload configurations and provide an executable scenario of the full evaluation process (*cf.* Figure 2). The elasticity evaluation scenario needs to support the definition of the adaptation tasks (ET_4, ET_5) on a fine-grained level, yielding relevant adaptations on operator side, realistic workload models, and support for fluctuating workloads and overload situations [9].

An *adaptation controller (R2)* provides an external adaptation mechanism for both the DBMS cluster, but also the adaptation of running workloads, and the scheduling of additional workloads. The latter supports the emulation of fluctuating workloads and overload situations.

A *detailed evaluation data set (R3)* is the basis for the computation of the elasticity metrics defined in Section II-C. It includes not only fine-grained performance metrics, but also an extensive set of evaluation meta-data such as cloud resource provisioning times, adaptation trigger timestamps, system metrics, and DBMS metrics.

Evaluation automation support (R4) covering all ETs ensure transparency and reproducibility that are key concepts in DBMS and cloud service evaluations [10], [11]. Moreover, portability support is required by a reasonable level of abstraction across the cloud, DBMS and workload domain [18], [21], *i.e.* enabling the execution of evaluation scenario on different cloud resources or for different DBMS.

IV. Elasticity Evaluation Framework: Kaa

In earlier work we analysed established DBMS benchmarks and evaluation frameworks with respect to their support of elasticity evaluations [18]. The results show that a variety of OLTP[1] and HTAP[2] partially fulfil R3, but none of them support R1, R2 and R4. Here, we introduce Kaa to overcome these limitations. It is based on our Mowgli scalability evaluation framework [14]. We first give an overview on Mowgli before introducing Kaa and it elasticity evaluation extensions.

A. Mowgli

Mowgli automates the performance and scalability evaluation process, *i.e.* ET_1 – ET_3 and ET_6 – ET_7 (cf. Figure 2). It enables the definition of portable and reproducible performance and scalability *evaluation templates* and their execution via a loosely coupled and extensible architecture (cf. Figure 3).

[1] online-transaction-processing (OLTP)
[2] hybrid-transaction-analytical-processing (HTAP)

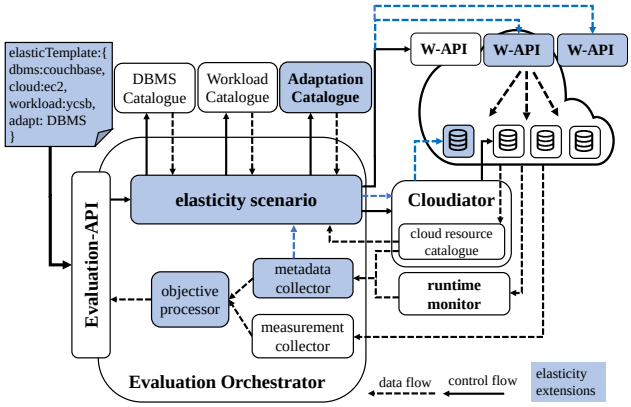


Fig. 3: Mowgli architecture with Kaa elasticity extension

Mowgli exploits cloud orchestration tools (COTs) [22] and combines them with extensible DBMS, auto-generated cloud resource and workload catalogues. *Evaluation scenario (ES)* templates define the required input structure of the *Evaluation API*. The catalogues contain the technical implementation of each ES. The *evaluation orchestrator* schedules the execution of each ES via the Cloudiator COT [22] and the *workload-APIs (W-API)*. In addition, it collects extensive system and DBMS metrics.

B. Kaa

In order to support elasticity evaluations, we enhance Mowgli through Kaa (blue boxes in Figure 3). Kaa enables the specification and automated execution of ET_4 and ET_5 to establish reproducible and portable elasticity ESs. Table II summarises the parameters and supported parameter ranges for the DBMS adaptation template. Workload adaptation parameters are listed in Table III. For illustration, we provide a set of full-fledged elasticity scenario templates, including cloud resource, DBMS and workload specifications [14][3].

TABLE II: DBMS Adaptation Template

Constraint	Parameter Range	Case Study
adaptation type	$\{scale-out, scale-in\}$	scale-out
trigger type	$\{time, DSR\}$	time
time trigger	$\{1 \dots n\}$ $seconds$	180
DSR	$(metric, threshold, duration)$	N/A

The DBMS adaptation specification comprises n DBMS adaptation templates, defining the desired adaptation types and adaptation triggers: Time-based triggers define a delay until the adaptation is executed; DSRs define advanced triggers by composing metrics, thresholds and validity durations. The evaluation orchestrator sequentially processes the adaptation steps. For DSRs, Kaa correlates the defined threshold and the validity period with the runtime metrics collected by the DBMS monitor (implemented through InfluxDB). Currently, Kaa supports triggers based on the system metrics CPU, memory, disk, and network. The framework is extensible for additional system metrics and DBMS-specific metrics.

The analogue approach is executed for the workload adaptation templates where the adaptation type *increase* either (i) increases the number of threads for of a running workload via the W-API or (ii) starts identical workloads on additional W-API instances. The actual decision is workload-dependent. For the integrated Yahoo Cloud Serving Benchmark (YCSB) [12], only the latter option is supported. Time- and metric-based triggers are processed in the similar way as for the DBMS adaptation templates. Progress-based triggers correlate the overall progress of the baseline workload (*i.e.* ET_3) with the progress threshold.

By automating each evaluation task, Kaa is not only able to provide performance metrics, but also extensive task-related meta-information such as applied configurations and runtimes that are provided in machine interpretable formats for advanced post processing.

TABLE III: Workload Adaptation Template

Constraint	Parameter Range
adaptation type	$\{increase, decrease\}$
worker threads	$\{1 \dots n\}$
trigger type	$\{time, progress, WSR\}$
time trigger	$\{1 \dots n\}$ $minutes$
progress trigger	$\{0 \dots 1.0\}$ $\%$ of $total$ $workload$ $progress$
WSR	$(metric, threshold, duration)$

V. Case Study

In order to validate Kaa against the identified evaluation challenges (*cf.* Section III), we apply a case study evaluating the elasticity of two distributed DBMS operated on cloud resources. While Kaa supports multiple cloud providers, DBMS and workloads [14], we address the following baseline hypothesis which is relevant for each DBaaS provider:

Hypothesis: *During an elastic scale-out, the DBMS continuously serves requests while the throughput decreases due to the data redistribution. With the completion of the data redistribution, the throughput starts to increase and surpasses the initial throughput.*

In the following, first we introduce methodology and the concrete ES specifications we use. Secondly, we analyse the results. Due to space limitations we only analyse the results with respect to throughput, while the complete Kaa data set also contains latency-related evaluations. The 160 data sets are archived and publicly available [23].

A. Methodology

For the evaluation, we apply a two-phase approach for each of the DBMS: In the *workload calibration* phase, we iteratively apply increasing intensive workloads to a fixed and pre-defined cluster size (ET_1 – ET_3, ET_6, and ET_7). For each workload intensity, we obtain the DBMS utilization.

[3]https://omi-gitlab.e-technik.uni-ulm.de/mowgli/getting-started

From the *(utilization, workload intensity)* tuples, we identify those workload intensities that lead to *low*, *optimal* or *overload* DBMS utilization as specified in Table VII.

In the *elastic scale-out* phase, we apply the resulting three tuples under a time-based adaptation trigger. This results in the iterative execution of ET_1 – ET_4, ET_6, and ET_7.

B. Configuration

For the case study, we rely on fixed cloud resource configurations: VM_SMALL for DBMS nodes and VM_LARGE for the W-API instance (cf. Table IV). All physical servers hosting VM_SMALL types use a two SSDs Raid-0 configuration.

TABLE IV: Cloud Resource Specifications

Specification	VM_SMALL	VM_LARGE
Cloud	private OpenStack Ulm, version Rocky	
VM vCores	2	8
VM RAM	4GB	8GB
VM Disk	70GB	20GB
OS	Ubuntu Server 16.04	
network	private, 10GbE	

We select Apache Cassandra and Couchbase as both are popular NoSQL DBMS[4] that have already been subject to scalability evaluations and achieved promising results [14], [24]. Both provide a comparable multi-master architecture that supports automated sharding and horizontal scalability. Due to their architectural similarities, a comparable cluster size and replication factor can be applied (cf. Table V). Yet, as they differ with respect to their consistency mechanisms, client-side consistency settings differ. Each DBMS node is configured to use 50% of the available memory for its operation.

TABLE V: DBMS Runtime Specifications

Specification	Apache Cassandra	Couchbase
Version	3.11.2	5.0.1 community
Cluster size	3	3
Replication	3	3
Write consistency	ONE	P=0, R=0
Read consistency	ONE	default

For the evaluation, Kaa uses one W-API that provides YCSB [12] in version 0.15.0. It is a widely adopted workload to evaluate distributed DBMS. In order to demonstrate the flexibility of our approach, we define typical DBMS workloads: the write-heavy workload emulates volume spikes while the read-heavy workload emulates data spikes [20].

C. Calibration Phase Results

The calibration phase is required to determine the DBMS utilization by correlating the workload intensities (*cf.* Table VI) with the resulting throughput of the write-heavy and read-heavy workloads. By choosing these number of worker threads, we ensure that the CPU load of the W-API does not exceed 80% to avoid a bottleneck at W-API side. Figure 4

[4]https://db-engines.com/en/ranking

TABLE VI: Workload Specifications

Specification	write-heavy	read-heavy
Metric reporting interval	10 s	
Initial Number of records	2.000.000	
Record size	5KB	
Operations	5.000.0000 writes.	5.000.0000 reads
Read distribution	N/A	zipfian
Calibration worker threads	1-2-4-8-16-32-64-128	

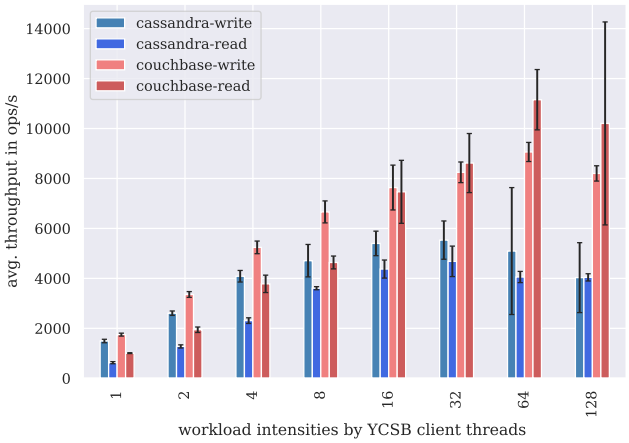


Fig. 4: Calibration Results — 3 Node Cluster

presents the calibration scenario results. The y-axis displays the average throughput and standard deviation of the five executions per calibration scenario. The y-axis shows the applied workload intensities. Based on the scale-out metric (*cf.* Section II-C) and increasing standard deviation, we obtain the six *(utilization, workload intensity)* tuples as shown in Table VII.

TABLE VII: Calibration Tuples

	Specification	Cassandra	Couchbase
low	highest average throughput with lowest standard deviation	4	4
optimal	highest average throughput in relation to the standard deviation	16	32
overload	decreased average throughput and growing standard deviation	64	128

D. Elastic Scale-out Phase Results

The elastic scale-out phase analyses the DBMS elasticity by applying one elastic scale-out adaptation as specified in the case study column of Table II. Each elasticity scenario uses an initial cluster size of three nodes (*cf.* Table V) and applies a time-based trigger after 180 seconds that starts the elastic scale-out. We specify and execute the ESs for all eight workload intensities used in the calibration phase, each executed for five times, resulting in 80 evaluation data sets[4].

We validate the introduced hypothesis in an explorative manner by analysing the throughput development in correla-

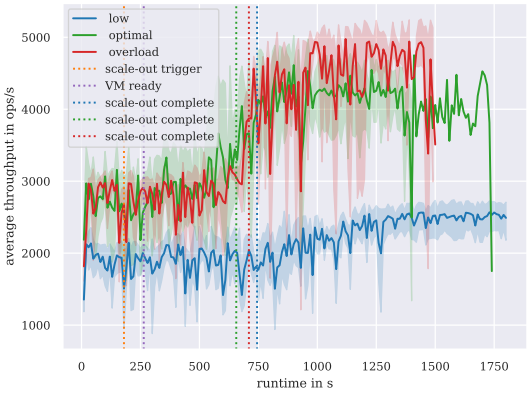


Fig. 5: Cassandra—elastic scale-out, read-heavy workload

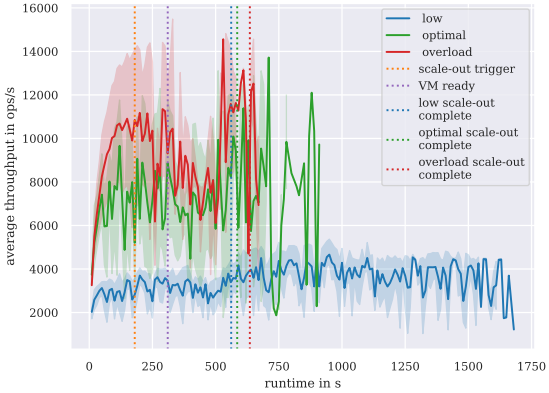


Fig. 6: Couchbase—elastic scale-out, read-heavy workload

tion to the applied evaluation task. The resulting graphs depict on the x-axis the evaluation runtime in seconds, *i.e.* ET_1 to ET_7. The y-axis depicts the average throughput including the standard deviation per timestamp over the five scenario execution. Each graph is workload-specific and groups the three calibration tuples in one graph per DBMS. In order to visualize the defined elasticity metrics (*cf.* Section II-C), each graph contains the fixed *scale-out trigger*, the average VM allocation time and the average scale-out end timestamp per utilization as vertical lines. Consequently, *provisioning duration* is visualized by the *scale-out trigger* to *VM ready* states, the *data distribution impact* is visualized by throughput development from the *VM ready* to *scale-out complete* state and likewise the *data distribution duration* for the runtime. The overall runtimes vary, as we apply the number of operations as limiting factor.

1) Read-Heavy Results: The results of the read-heavy workload are depicted in Figures 5 and 6. For Apache Cassandra, they confirm the hypothesis. There even is no significant data distribution impact during the scale-out phase. For Couchbase, the results indicate a confirmation of the hypothesis, yet with a significant data distribution impact. In comparison, Couchbase achieves a drastically higher throughput for the optimal and overload utilization (and consequently shorter runtimes) than Apache Cassandra. This is probably due to Couchbase's weaker consistency mechanisms. In conclusion, the read-heavy results confirm the initial hypothesis for both DBMS.

2) Write-Heavy Results: Figure 7 and Figure 8 depict the results for the write-heavy workload. Both graphs show a similar data distribution impact over the evaluation runtime for the applied calibration tuples: the scale-out for low utilization results in minor data distribution impact, but after the scale-out completion, no significant throughput increase is achieved. For the optimal and overload states, the data distribution impact is worse; throughput constantly decreases and the scale-out only completes after the workload has finished. These results show that an elastic scale-out imposes too much additional load due

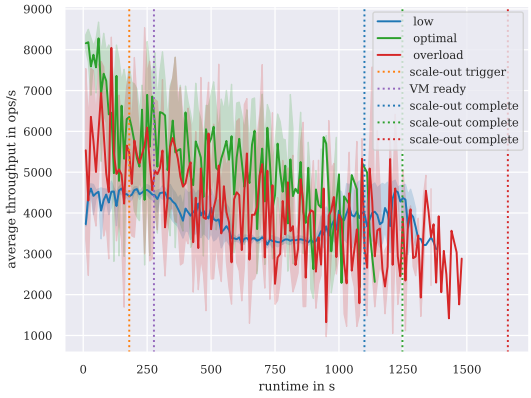


Fig. 7: Cassandra—elastic scale-out, write-heavy workload

to data redistribution and the constantly increasing number of data that needs to be redistributed and the additional replicas that need to be generated. Moreover, depending on the DBMS-specific sharding mechanism, the data redistribution might be constantly invoked due to incoming write request [4]. Consequently, the write-heavy results disprove the initial hypothesis for Cassandra and Couchbase.

E. Lessons Learned

While the applied case study only covers a small scope of Kaa's supported elasticity scenarios, it reveals a number of relevant elasticity insights *(EI)*: *(EI-1)* elastic scale-out adaptations under read-heavy workloads behave as expected, even under overload situations, *i.e.* reactive and pro-active DSRs can be applied; *(EI-2)* elastic scale-out adaptations under write-heavy workloads impose significant additional load on the DBMS, especially for optimal and overload utilization, resulting in negative outcome. Therefore, DSRs for write-heavy workloads shall be proactive. *(EI-3)* Elastic scale-out for low utilized DBMS results in low data distribution impact, but in parallel, in low throughout increase. Hence, low utilized

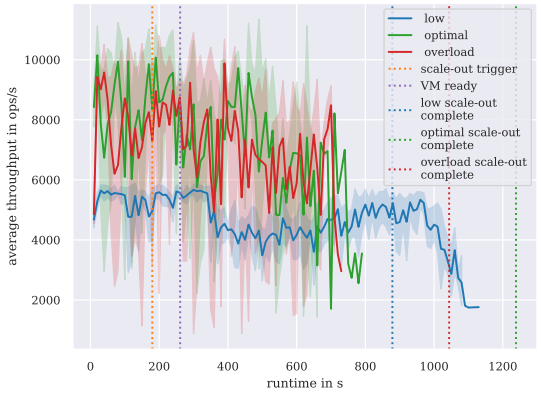


Fig. 8: Couchbase—elastic scale-out, write-heavy workload

DBMS are suitable for proactive elastic scale-outs if increasing workloads are expected. *(EI-4)* Defining DSRs requires the consideration of essential impact factors, such as the workload type and the DBMS utilization, and potential impact factors like DBMS configurations and resource provisioning time.

VI. Discussion

Here, we validate Kaa against the presented challenges on a qualitative basis before we outline its potential adoptions.

A. Validation

Elasticity evaluation scenario specification (R1): Extending Mowgli's ES with adaptations provides comprehensive elasticity scenarios, comprising cloud resources, DBMS runtime properties and composed adaptations scenarios, including DBMS and workload adaptations. Ongoing work addresses advanced adaptation specifications, including more complex DSRs [1], [25] and workload specifications [20].

Adaptation controller (R2): Kaa realises DBMS and workload adaptions at runtime by two-level orchestration: the evaluation orchestrator executes workload adaptations; a cloud orchestrator manages cloud resources and DBMS adaptations.

Evaluation data (R3): Regarding the granularity of the provided performance metrics, Kaa is dependent on the workload generator. Yet, due to its modular architecture, suitable workloads can easily be integrated via W-API. Currently, Kaa supports YCSB and two OLTP workloads. The produced data sets contain all relevant cloud resources, DBMS runtime settings, system metrics, DBMS metrics and the duration of each elastic scale-out phase in machine interpretable formats as well as visualized to support explorative analysis.

Evaluation automation support (R4): By building upon Mowgli that has been validated for enabling automation and reproducible performance and scalability results [14], Kaa extends these capabilities for elasticity evaluations. This is also verified by the applied case study that comprises the automated execution of 80 calibration and 80 elasticity scenarios. In addition, the portability of the elasticity scenario specification has been validated for two DBMS and can easily be achieved for different cloud providers as shown in [14].

B. Looking Ahead

The obtained elasticity insights lead to several remaining challenges and potential adopters: *(i)* long running elasticity ESs are required, including multiple elastic scale-out/-in and adaptations to analyse more realistic DBMS operation scenarios; *(ii)* DBMS elasticity needs to be analysed against more complex workloads of the OLTP and HTAP domain; *(iii)* an in-depth analysis of cloud providers and DBMS runtime configurations is required to derive comprehensive elasticity impact factors; *(iv)* the evaluation results of Kaa can be fed into scaling rules of DBMS auto-scalers such as Tiramloa [6] and MeT [7] or for building the foundation of DBMS adaptations by cloud auto-scalers [1].

VII. Related Work

Research on elasticity for cloud computing can be divided into describing elasticity through metrics as well as approaches to measure service elasticity. Auto-scaling is also a popular research topic in cloud computing [1], but not considered here as auto-scaling applies elasticity, but does not evaluate it.

Several metrics for elasticity have been proposed in literature: Early work focuses on mere (de)provisioning time [26], [27]. However, these metrics cover only technical properties and thus are independent of the timeliness and accuracy of demand changes. The *elastic speedup* metric proposed by SPEC OSG [27] does not capture the dynamic aspects of elasticity and is regarded as a scalability metric. Other approaches like [12], [13], [28], [29] characterise elasticity indirectly by analysing latency and throughput.

To counter the drawbacks of aforementioned metrics, elasticity metrics explicitly covering the timeliness, accuracy and stability were proposed and used in different use cases [3], [25], [30]. Yet, those metrics are mostly tied to the elasticity of stateless cloud applications and thus, are not suitable for DBMS for which elasticity is defined differently. The industry-driven SPEC Cloud™IaaS 2018[5] partially covers DBMS workloads, but still omits elasticity in the sense of automated scaling under changing load. Approaches for evaluating the elasticity in a stateless context include BUNGEE, a framework for benchmarking the elasticity of cloud platforms [30].

The need for DBMS-centric elasticity evaluations is highlighted by [4], [9] due to the growing heterogeneity of cloud resources and distributed DBMS. While there is a significant number of DBMS workloads that measure the performance metrics at client side, none of them supports the required adaptions at operator side [18]. Consequently, supportive frameworks are required to automate the evaluations and enable reproducible results [17], [18], [21]. However, DBMS frameworks that also support the operator side are limited [14], [31] and none of them supports the specification and execution of elasticity evaluations.

[5]https://www.spec.org/cloud_iaas2018/

Consequently, the number and scope of existing DBMS elasticity evaluations in the cloud is limited [8], [13], [24], [32], as supportive frameworks are missing, which also impedes their reproducibility. Our novel framework Kaa addresses these limitations by ensuring reproducible and portable DBMS elasticity evaluations in the cloud, that also enables to keep track with evolving cloud resources and DBMS.

VIII. Conclusion

Elasticity has become a key feature of cloud applications and its comparative evaluation is a central topic for cloud resources, stateless services and stateful components such as distributed DBMS. While the elasticity evaluation of the former is supported by specific evaluation frameworks, the elasticity evaluation of distributed DBMS remains a challenge, as supportive metrics exist, but evaluation frameworks are missing. This hinders reproducible and portable evaluations, which are key concepts of cloud service benchmarking. Therefore, we present the novel elasticity evaluation framework Kaa that automates the execution of DBMS elasticity evaluations and ensures reproducibility. We validate our approach by applying an elasticity evaluation case study for two DBMS under eight workload intensities and one elastic scale-out adaptation. Kaa fully automates the benchmarking of the resulting 160 evaluation scenarios and the results provide valuable insights into the elasticity and potential impact factors, such as the workload type and DBMS utilization.

Future work will comprise extensive elasticity evaluations, focusing on more complex workloads and adaptions. In addition, extensions of the metric processing capabilities will enable the analysis of established elasticity metrics under workload, cloud resource and DBMS-specific aspects.

Acknowledgements

The research leading to these results has received funding from the EC's Framework Programme HORIZON 2020 under grant agreements 731664 (MELODIC) and 732667 (RECAP).

References

[1] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling web applications in clouds: A taxonomy and survey," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 73:1–73:33, Jul. 2018.

[2] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in cloud computing: state of the art and research challenges," *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 430–447, 2017.

[3] N. Herbst, A. Bauer, S. Kounev, G. Oikonomou, E. V. Eyk, G. Kousiouris, A. Evangelinou, R. Krebs, T. Brecht, C. L. Abad *et al.*, "Quantifying cloud performance and dependability: Taxonomy, metric design, and emerging challenges," *ACM ToMPECS*, vol. 3, no. 4, p. 19, 2018.

[4] A. Davoudian, L. Chen, and M. Liu, "A survey on nosql stores," *ACM Computing Surveys (CSUR)*, vol. 51, no. 2, p. 40, 2018.

[5] S. Kächele, C. Spann, F. J. Hauck, and J. Domaschka, "Beyond iaas and paas: An extended cloud taxonomy for computation, storage and networking," in *IEEE/ACM 6th UCC*. IEEE, 2013, pp. 75–82.

[6] D. Tsoumakos, I. Konstantinou, C. Boumpouka, S. Sioutas, and N. Koziris, "Automated, elastic resource provisioning for nosql clusters using tiramola," in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE, 2013, pp. 34–41.

[7] F. Cruz, F. A. F. M. A. Maia, M. Matos, R. C. M. d. Oliveira, J. Paulo, J. Pereira, and R. M. P. Vilaça, "Met: workload aware elasticity for nosql," in *8th ACM EuroSys*. ACM, 2013, pp. 183–196.

[8] D. Seybold, N. Wagner, B. Erb, and J. Domaschka, "Is elasticity of scalable databases a myth?" in *IEEE Big Data*, 2016.

[9] S. Sakr, "Cloud-hosted databases: technologies, challenges and opportunities," *Cluster Computing*, vol. 17, no. 2, pp. 487–502, 2014.

[10] D. Bermbach, E. Wittern, and S. Tai, *Cloud service benchmarking*. Springer, 2017.

[11] A. V. Papadopoulos, L. Versluis, A. Bauer, N. Herbst, J. Von Kistowski, A. Ali-eldin, C. Abad, J. N. Amaral, P. Tůma, and A. Iosup, "Methodological principles for reproducible performance evaluation in cloud computing," *IEEE Transactions on Software Engineering*, pp. 1–1, 2019.

[12] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *ACM SoCC*, 2010.

[13] T. Dory, B. Mejías, P. Roy, and N.-L. Tran, "Measuring elasticity for cloud databases," in *Proceedings of the The Second International Conference on Cloud Computing, GRIDs, and Virtualization*, 2011.

[14] D. Seybold, M. Keppler, D. Gründler, and J. Domaschka, "Mowgli: Finding your way in the dbms jungle," in *Proceedings of the 2019 ACM/SPEC ICPE*. ACM, 2019, pp. 321–332.

[15] S. Mazumdar, D. Seybold, K. Kritikos, and Y. Verginadis, "A survey on data storage and placement methodologies for cloud-big data ecosystem," *Journal of Big Data*, vol. 6, no. 1, p. 15, 2019.

[16] D. Agrawal, A. El Abbadi, S. Das, and A. J. Elmore, "Database scalability, elasticity, and autonomy in the cloud," in *DASFAA*, 2011.

[17] J. Gray, *Benchmark handbook: for database and transaction processing systems*, 1992.

[18] D. Seybold and J. Domaschka, "Is distributed database evaluation cloud-ready?" in *ADBIS*. Springer, 2017, pp. 100–108.

[19] D. Seybold, "Towards a framework for orchestrated distributed database evaluation in the cloud," in *18th Doctoral Symposium of the 18th International Middleware Conference*. ACM, 2017, pp. 13–14.

[20] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "Characterizing, modeling, and generating workload spikes for stateful services," in *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010, pp. 241–252.

[21] L. Brent and A. Fekete, "A versatile framework for painless benchmarking of database management systems," in *Australasian Database Conference*. Springer, 2019, pp. 45–56.

[22] D. Baur, D. Seybold, F. Griesinger, H. Masata, and J. Domaschka, "A provider-agnostic approach to multi-cloud orchestration using a constraint language," in *CCGRID*, May 2018, pp. 173–182.

[23] D. Seybold and J. Domaschka, "Mowgli: Dbms elasticity evaluation data sets," Aug. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3362279

[24] J. Kuhlenkamp, M. Klems, and O. Röss, "Benchmarking scalability and elasticity of distributed database systems," *Proceedings of the VLDB Endowment*, vol. 7, no. 12, pp. 1219–1230, 2014.

[25] A. Bauer, N. Herbst, S. Spinner, A. Ali-Eldin, and S. Kounev, "Chameleon: A hybrid, proactive auto-scaling mechanism on a level-playing field," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 800–813, 2018.

[26] Z. Li, L. O'Brien, H. Zhang, and R. Cai, "On a Catalogue of Metrics for Evaluating Commercial Cloud Services," in *ACM/IEEE CCGrid 2012*, Sept 2012, pp. 164–173.

[27] D. Chandler and more, "Report on Cloud Computing to the OSG Steering Committee," Tech. Rep., Apr. 2012.

[28] C. Binnig, D. Kossmann, T. Kraska, and S. Loesing, "How is the Weather Tomorrow?: Towards a Benchmark for the Cloud," in *DBTest 2009*, ser. DBTest '09. New York, NY, USA: ACM, 2009, pp. 9:1–9:6.

[29] R. F. Almeida, F. R. Sousa, S. Lifschitz, and J. C. Machado, "On defining metrics for elasticity of cloud databases." in *SBBD*, 2013, pp. 12–1.

[30] N. R. Herbst, S. Kounev, A. Weber, and H. Groenda, "Bungee: an elasticity benchmark for self-adaptive iaas cloud environments," in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press, 2015, pp. 46–56.

[31] M. Klems, D. Bermbach, and R. Weinert, "A runtime quality measurement framework for cloud database service systems," in *Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the*. IEEE, 2012, pp. 38–46.

[32] I. Konstantinou, E. Angelou, C. Boumpouka, D. Tsoumakos, and N. Koziris, "On the elasticity of nosql databases over cloud management platforms," in *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 2011, pp. 2385–2388.

Chapter 14

[core7] King Louie: Reproducible Availability Benchmarking of Cloud-hosted DBMS

This article is published as follows:

Daniel Seybold, Stefan Wesner, and Jörg Domaschka. ”King Louie: Reproducible Availability Benchmarking of Cloud-hosted DBMS”, *35th ACM/SIGAPP Symposium on Applied Computing (SAC)*, published 2020, ACM, DOI: https://doi.org/10.1145/3341105.3373968

King Louie: Reproducible Availability Benchmarking of Cloud-hosted DBMS

Daniel Seybold
Institute of Information Resource Management, Ulm University
Ulm
daniel.seybold@uni-ulm.de

Stefan Wesner
Institute of Information Resource Management, Ulm University
Ulm
stefan.wesner@uni-ulm.de

Jörg Domaschka
Institute of Information Resource Management, Ulm University
Ulm
joerg.domaschka@uni-ulm.de

ABSTRACT

Cloud resources have become a preferred operational model distributed Database Management Systems (DBMS) by offering the elasticity and virtually unlimited scalability, but increase the risk of failures with increasing cluster sizes. While distributed DBMS provide high-availability mechanisms, it is currently an open research question to what extent they are able to provide availability and performance guarantees in case of cloud resource failures. Especially as existing DBMS benchmarks do not consider availability. We present a comprehensive methodology for evaluating the availability of distributed DBMS in case of cloud resource failures. Based on this methodology, we introduce a novel framework that automates the full evaluation process, including the failure injection, and emphasizes reproducibility. The framework is validated by 16 diverse availability evaluations. The results show that distributed DBMS are not necessary available even if sufficient replicas are available and clients can experience significant downtimes.

CCS CONCEPTS

• **Information systems** → **Database performance evaluation**; *Database utilities and tools*; • **Computer systems organization** → **Cloud computing**;

KEYWORDS

availability, NoSQL, cloud, distributed DBMS, benchmark

ACM Reference Format:
Daniel Seybold, Stefan Wesner, and Jörg Domaschka. 2020. King Louie: Reproducible Availability Benchmarking of Cloud-hosted DBMS. In *The 35th ACM/SIGAPP Symposium on Applied Computing (SAC '20), March 30-April 3, 2020, Brno, Czech Republic.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3341105.3373968

1 INTRODUCTION

The raise of cloud computing a decade ago has led to a change in the way storage backends are treated. Instead of large, single-instance servers, the NoSQL movement has established a large amount of different Database Management Systems (DBMS) that share the concept of a distributed architecture [11]. Here, multiple independent DBMS instances form a cluster and provide the user with the impression of a single logical DBMS.

A distributed architecture enables scaling the DBMS horizontally as needed by adding more resources [26]. This increases the storage capacity of a DBMS as well as its processing capabilities. Often scaling can even be realised as an elastic scale-out during run-time without users of the DBMS experiencing any downtime [20].

Yet, the risk of failures increases with the size of the systems as failures are more likely for distributed DBMS than for single-instance DBMS. On the other hand, the distributed nature enables the introduction of replication by storing individual data items more than once. This facilitates a continued operation of the logical DBMS without the user noticing. However, the use of replication introduces overhead, as the various replicas of a data item need to be kept in sync; and due to the CAP theorem [9], availability of the DBMS and consistency of replicas are not independent, but influence each other. The higher the need for availability, the less consistency guarantees can be given in the case of failures. Vice-versa, the higher the need for consistency, the less available the system will be in the case of failures. Many of existing DMBS claim high-availability. Yet, it is widely unclear where in the continuum of possible consistency-availability trade-offs they reside and what will be the impact on the performance.

An established approach to exploit the scalability and elasticity properties of distributed DBMS is to host them across IaaS cloud systems, as they provide the necessary flexibility when more (virtual) hardware is required [26]. On the downside, cloud resources are volatile and also prone to failure on various levels [1, 16].

When designing a large-scale application, the choice of a storage backend is a crucial decision. Yet, no DBMS allows the direct configuration of availability properties; instead, a user often can only select indirect parameters such as replication degree and consistency demands, whose impact on availability is only mediately visible. Hence, it is difficult to assess the impact of a configuration on availability, but also to assess the impact of a failure on the overall DBMS performance. Moreover, future DBMS will experience different demands than existing ones as future IoT environments will expose constant write-heavy workloads that are contrary to the current read-heavy workloads.

In consequence, the choice of a DBMS requires extensive evaluations which are very time consuming when done manually, due to the large number of configuration properties and different failures [7]. Furthermore, these benchmarks need to be repeated for new DBMS appearing on the market, but also when new versions

SAC '20, March 30-April 3, 2020, Brno, Czech Republic

ACM ISBN 978-1-4503-6866-7/20/03…$15.00
https://doi.org/10.1145/3341105.3373968

of existing DBMS are released. Hence, a large degree of automation is required for reproducible benchmarking [23].

To mitigate this situation, this paper provides the following contributions: (i) we define a reproducible and portable methodology for evaluating the availability of DBMS in the context of resource failures; (ii) we introduce a novel evaluation framework that implements the defined methodology; (iii) we present an Web- and IoT-based case study that validates the framework and provides novel insights into the availability of distributed DBMS.

Our considerations and also methodology focuses exclusively on the client perspective and hence, captures metrics such as performance (latency and throughput) and success rate of queries.

The remainder of this document is structured as follows: Section 2 presents background on distributed DBMS, availability, and cloud hosting. Section 3 discusses DBMS system models and failure models. Section 4 introduces the novel availability evaluation methodology. Section 5 presents the King Louie framework. Section 6 defines the case study while Section 7 discusses the respective results. Section 8 validates King Louie. Section 2 presents related work before Section 10 concludes.

2 BACKGROUND

This section summarises background on availability and distributed DBMS focusing on their high-availability mechanisms.

2.1 Availability

For computing systems, *availability* is defined as *the degree to which a system is operational and accessible when required for use* [14]. Besides *reliability* (the *measure of the continuity of correct service"* [2]), availability is the main pillar of many fault-tolerant implementations [12]. In this paper, we focus on the availability aspect from the client perspective and assume that DBMS are reliable, *i.e.* work as specified. The availability of the DBMS can be affected by two kinds of conditions: (i) A high number of requests issued concurrently by clients, overloading the DBMS so that the requests of clients cannot be handled at all or are handled with an unacceptable $latency > \Delta t$. (ii) Resource failures occur that impact network connectivity or the compute/storage resources the DBMS is hosted on [12]. In this work, we solely consider (ii).

2.2 Distributed DBMS

Over the last decade the DBMS landscape is showing an increasing heterogeneity with respect to data models *relational*, *NoSQL* and *NewSQL*, but also with respect to distributed architectures [11, 22]. Distributed DBMS provide a logical DBMS instance to clients, but distribute the DBMS functionality across multiple physical or virtual resource entities. These resources hosting the DBMS nodes are connected via network and form a DBMS cluster.

Distributed DBMS provide horizonal scalability by exploiting sharding as a basic technique: the full data set is separated such that each data-hosting DBMS node manages only a local share of the overall data set. With more nodes joining the cluster, the overall capacity increases and the system scales horizontally.

As growing cluster sizes increase the probability of failures, distributed DBMS provide diverse means for replication [12]. A key characteristic is the *replication scope* that is classified into full and partial replication [12]. Full replication defines that each DBMS node stores a full copy of the entire database. In partial replication, each data item may only be stored on a subset of all DBMS nodes, *i.e.* the replication factor is smaller than the DBMS cluster size. Partial replication closely relates to sharding as if sharding is used without replication, there is no tolerance against DBMS node failures. On the other hand, using replication without sharding means that all data is available on all DBMS nodes.

Due to given constraints of the CAP theorem [9] that a distributed storage system can only provide two out of the three properties: consistency (C), availability (A) and partition tolerance (P), distributed DBMS provide heterogenous consistency guarantees [33]. Their evolvements show that these constraints are not a binary decision [8].

3 AVAILABILITY TRADE-OFFS

The heterogeneity of distributed DBMS offerings, their extensive amount of runtime configurations and the number of cloud resource offerings create a vast design space for operating distributed DBMS. While this design space includes use case specific decision such as the functional feature-driven DBMS selection [11] and the comparative evaluation of non-functional features such as performance and scalability [17, 21]. In this process the non-functional feature analysis needs to consider availability mechanisms as well to ensure a seamless operation in case of cloud resource failures [1, 16]. This work considers distributed DBMS from a cloud architect perspective with the primary focus on the client perspective and secondary on the DBMS operator perspective. As the clients main interest relies in getting the promised DBMS performance even in case of failures, we consider the performance impact in case of failures as key aspect in the following. Therefore, we do not discuss related approaches such as analysing the availability on the DBMS level [24] or evaluating the consistency impact on client and DBMS level [5].

In the following, we discuss availability and performance related design decision for selecting and operating cloud-hosted distributed DBMS, discuss potential cloud resource failure scenarios and classify both domains according to their impact on DBMS availability.

3.1 Operating Distributed DBMS

For operating distributed DBMS, we separate the design decisions into the *DBMS level* and *client level* and discuss the resulting *availability* and *performance* impact on the client level.

On DBMS level, a variety of fault-tolerance mechanisms are adopted in distributed DBMS [12]. Yet, the exposed runtime configurations of these fault-tolerance mechanisms are always DBMS specific and require the individual consideration and evaluation. A high-level set of common runtime configurations are the *cluster size*, the *replication factor*. With an increasing *cluster size (CS)*, data is sharded across the DBMS nodes, an increase of the overall DBMS performance can be expected if the load is evenly distributed across all nodes in the cluster. In parallel, the probability of failures increases as DBMS nodes might be distributed across multiple potentially failing resources. Therefore, replication is preferably enabled to increase availability by defining a suitable *replication factor (RF)*. The scale of the RF is limited by the CS or a DBMS specific *replication factor limit (RFL)* following the constraint $1 \leq RF \leq min(CS, RFL)$.

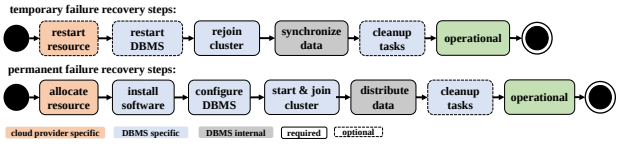


Figure 1: Failure Recovery Steps

The replication factor defines the existing number of identical data items within a DBMS cluster. Thus, it increases data availability as in case of a DBMS node failure the remaining replicas still contain copies of the data. With respect to data accessibility, replicas can take over read and write requests depending on the applied client consistency as described below. Yet, increasing the RF comes with a cost in performance as the data replication and synchronization mechanisms add additional load on the DBMS cluster.

3.2 Cloud-hosted DBMS

As cloud resources tend to fail [1, 16], their usage impacts DBMS availability as failures on different levels of the cloud resource stack can have impact on individual DBMS nodes (running on a VM), but also on larger sets of them. For instance, the failure of a hypervisor will lead to the failure of all VMs on that hypervisor [28]. Consequently, the *failure type* can be caused on the physical level as servers, network links, network device, power supplies, or cooling may fail. As a result, all DBMS nodes hosted on these resources, either directly or virtualised, become unavailable. Similarly, on the software level, management software and hypervisors can fail; algorithms, network, and devices can be buggy, misconfigured, or in the process of being restarted. Any of these failures can affect virtual machines and virtual networks, but also physical servers, physical networks, entire racks, complete availability zones, or even entire data centres. The failure of a full *availability zone* or even an entire *region* leads to the unavailability of multiple physical servers and hence unavailability of all hosted virtual machines and consequently the DBMS nodes running on these resources. These failures can be classified based on their severity *permanent* and *temporary* and occurrence expressed in the *failure rate (FR)*.

In parallel, the elasticity of cloud resources enables the implementation of manual or (semi-) automated recovery mechanism to overcome such failures. In order to recover from such failures, we define a set generic recovery steps for cloud-hosted DBMS as depicted in Figure 1, one for temporary failures and one for permanent resource failures. While the cloud provider and DBMS specific steps can be executed manually or automated by relying on cloud orchestration tools (COTs) [4], the DBMS internal steps are executed by the DBMS itself. For temporary failures, depending on the failure type, only a subset of the depicted steps might be required, *e.g.* in case of a temporary network outage, the unavailable DBMS nodes will synchronize their data with the remaining nodes as soon as their are online again.

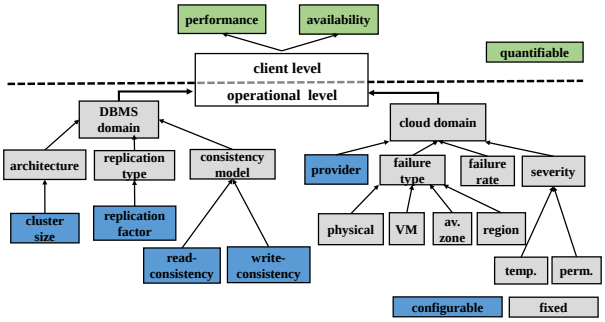


Figure 2: Availability Impact Factors

3.3 Impact Factor Classification

Operating distributed DBMS in the cloud and ensuring high-availability already becomes a challenging task due to the vast amount of distributed DBMS offers in combination with their heterogeneous runtime configuration [22]. The diverse set of potential cloud resource failures even increases this challenge. In Figure 2 we summarise availability and performance impact factors based on the *DBMS domain* and *cloud domain*. We classify the impact factors into *configurable* and *fixed* impact factors. Fixed impact factors are given by the selected DBMS such as architecture type or consistency model, or resource failure probabilities. Configurable impact factors offer a certain range of runtime configurations such as the cluster size for the DBMS or the provider for the cloud domain.

Consequently, the selection and operation process of cloud-hosted DBMS requires the consideration of a multitude of impact factors to ensure high performance, but also availability for the client level. Therefore, the preferred approach for underpinning this process is the comparative evaluation of suitable DBMS with respect to their non-functional features. But as the heterogeneous DBMS landscape raises new challenges in comparative DBMS evaluations [7], the volatile cloud context even increases these challenges [6] and raises the need for reproducible and portable evaluations [23]. Moreover, as current benchmarks only focus on performance related evaluation objectives, availability in the context of failures is not yet in the scope of DBMS benchmarks [27]. In order to address these limitations, we define a comprehensive methodology which enables comparative availability evaluation of cloud-hosted DBMS by explicitly considering the presented impact factors.

4 EVALUATION METHODOLOGY

In order to introduce the concepts and challenges of our availability evaluation methodology, we build upon the following baseline hypothesis that describes the expected client level performance and availability in case of a failure on DBMS level.

Hypothesis (H1): *If a DBMS node fails due to a cloud resource failure, the performance is temporarily decreasing as client requests to the failed node are stalled until the DBMS recognize the DBMS node failure and trigger the replicas for taking over these requests; with the replicas taking over the performance is increasing, but does not achieve the initial performance due to the reduced CS. In case the permanent recovery steps are executed, performance will decrease during the data synchronization/redistribution and increase afterwards to the initial*

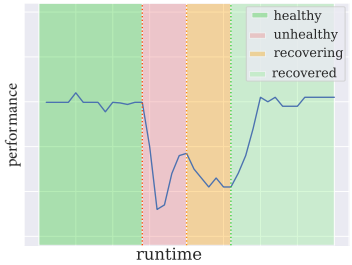


Figure 3: Expected DBMS performance in case of failure

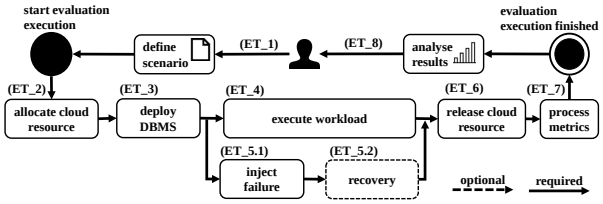


Figure 4: Availability Evaluation Process

performance. Throughout these phases, the DBMS connection remains available and all DBMS request are executed successfully.

Based on H1, the performance undergoes four phases depicted in the corresponding Figure 3: the *healthy phase* represents the operational cluster with its initial CS; in the *unhealthy phase n* DBMS nodes are unavailable due to resource failures;in the *recovery phase n* recovery steps are executed to resolve the failures before the cluster reaches the *recovered phase* with finished data synchronization/redistribution and its initial CS of operational nodes.

4.1 Metrics

As comparative availability evaluations require a set of quantifiable metrics, we build upon the established DBMS performance metrics *throughput* and *latency* [15]. These metrics are correlated with the respective phases in order to enable significant availability evaluations. In the following we only build upon the throughput metric, but the resulting metrics can similarly applied for latency. In order to correlate the throughput with the specific phase in case of a failure, we define the *phase throughput (PT)* as follows:

Phase Throughput (PT): $\frac{successful\ requests}{phase\ duration}$

As availability metrics, we consider downtime and the *request success rate (RSR)* as relevant metrics. The downtime metrics build upon [28] and represent the minimal tolerable throughput in correlation to the respective phase. The *phase downtime (PD)* and the according *phase downtime duration (PDD)* are defined as follows:

Phase Downtime (PD) : $PT < threshold$

PD Duration (PDD): $\sum_{phase\ start}^{phase\ end} PT < threshold$

The RSR builds upon the *expected service availability* [34] and spans over all all four phases:

Request Success Rate (RSR): $\frac{successfull\ requests}{total\ scheduled\ requests}$

4.2 Evaluation Process

To define a comprehensive availability evaluation methodology that provides the introduced metrics, we define the evaluation process depicted in Figure 4. It comprises eight *evaluation tasks (ETs)* with the concept of explicit cloud failure injection (ET_5.1) in combination with recovery tasks (ET_5.2) to emulate realistic cloud failure scenarios [16]. ET_1 specifies of a comprehensive *evaluation scenario*, comprising the cloud resources, the specific DBMS with the availability-related configurations, a DBMS workload and the cloud failure to be injected with an optional recovery action. The actual execution of the evaluation starts with ET_2 by allocating the cloud resources, followed by the deployment of the DBMS (ET_3). The workload execution (ET_4) and the failure injection in combination with the recovery are executed in parallel and can be executed multiple times if specified in ET_1. After ET_4 is finished, the cloud resources need to be released (ET_6). The metric processing (ET_7) completes the execution phase of the evaluation process. In the final step ET_8, the results are analysed and additional or refined evaluation scenarios are defined for further executions.

4.3 Evaluation Process Execution

The execution of this evaluation process involves detailed knowledge of the cloud provider, DBMS and workload domain. In addition, the parallel ET_4 and ET_5 require an orchestrated execution. These constraints lead to a time consuming, error prone and technically challenging execution if done manually. This limits reproducibility and portability across the domains, *e.g.* executing the evaluation scenario for different cloud providers or DBMS. Consequently, a supportive framework is required to execute the evaluation process. By building upon established benchmarking guidelines [6, 7, 23] we define the following requirements towards a supportive availability evaluation framework to ensure significance and reproducibility:

- **Evaluation Design (R1):** Reusable and comprehensive evaluation scenario specification need to be provided.
- **Abstraction (R2):** A suitable abstraction level is required for the technical domain specific properties.
- **Failure Injection (R3):** has to be supported for temporary and permanent failures of different failure types
- **Automation (R4):** is required for all ETs of the execution phase to ensure reproducibility and transparency, including the orchestration of parallel but timely dependent ETs
- **Extensibility (R5):** As each evaluation domain is constantly evolving, extensible specifications and architectures are required to support the integration of future domain specific constraints and failure models
- **Transparent Data (R6):** the resulting data needs to comprise raw metrics and extensive meta-data such as monitoring data, task related details, DBMS and cloud resource details.

5 KING LOUIE FRAMEWORK

In order to implement the defined methodology, we present the novel evaluation framework King Louie that explicitly addresses the identified requirements R1 - R6. King Louie builds upon the Mowgli framework [29] that addresses the automated performance and scalability evaluation of cloud-hosted DBMS. King Louie supports ET_1

by providing JSON-based evaluation scenario templates[1] to describe comprehensive availability evaluation scenarios. A simplified example is detailed in Listing 1. The `dbms` object exposes all relevant details (*cf.* Figure 2) to deploy a DBMS cluster on arbitrary cloud resources, a comprehensive description of the deployment model is presented in [29]. The `workload` object specifies the desired workload, including the DBMS specific read/write consistency configurations. The `failureSpec` defines the desired failure to be injected and optional recovery actions to be executed. In its current state, King Louie supports the failure types *VM*, *AVAILABILITY_ZONE* and *REGION* and the recovery *ADD-NODE* for permanent failures and *RESTART-DBMS* for temporary failures.

King Louie builds upon a loosely coupled and extensible architecture (*cf.* Figure 5). Evaluation scenarios are submitted via the *Evaluation API* and the *evaluation orchestrator* maps the specified configurations to its technical implementations. These are provided by extensible cloud, DBMS and workload *domain catalogues* (R5). Based on these technical implementations, the *evaluation orchestrator* fully automates the evaluation process (R4), *i.e.* ET_2 – ET_7. In order to abstract cloud provider APIs and ease the DBMS deployment (R2), King Louie exploits cloud orchestration tools (COTs) [4], in particular the Cloudiator framework [3]. The workload is executed by separate *workload-API (W-API)* instances that are controlled by the *evaluation orchestrator*. The failure injection is controlled by the evaluation orchestrator by exploiting the COTs control over allocated cloud resources. This enables the injection of different failure types such as the (temporary) stopping or deletion of single VMs, but also for VMs in specific availability zones or even regions. Moreover, additional failure types can be injected on running VMs the evaluation orchestrator is able to execute custom commands on the deployed DBMS nodes via the COT (R3).

By orchestrating each ET, King Louie is able to process availability and performance metrics and add extensive meta-information such as runtime configurations, logs and monitoring data that are provided in machine interpretable formats for advanced post processing (R6). Currently, King Louie supports EC2 and OpenStack based clouds, seven DBMS and two different workloads[2].

Listing 1: Availability Scenario Template

```
{"dbms": {
  "type": "CASSANDRA",
   "instances": 3,
   "replicationFactor" : 3
   "vmLocation" : "openstack",
   "vmType": "small",
   ...
  },
 "workload": {
  "type" : "YCSB",
  "config" : {
          "writeConsistency": "ONE",
          ...
  }
 }
 "failureSpec":[{
```

[1]https://omi-gitlab.e-technik.uni-ulm.de/mowgli/getting-started/tree/master/examples/availability
[2]https://omi-gitlab.e-technik.uni-ulm.de/mowgli/getting-started

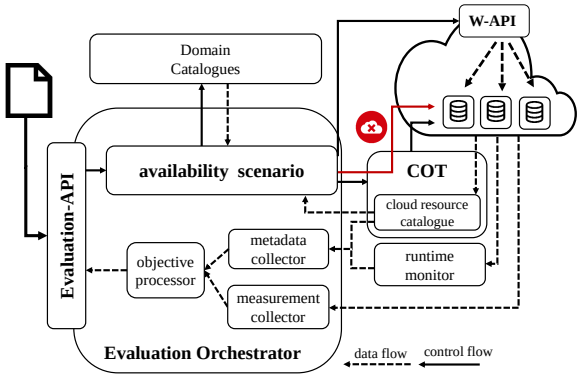


Figure 5: King Louie technical architecture

```
        "failureType": "VM",
        "severity": "permanent"
        "failureRecovery": true,
        "recoveryType": "ADD-NODE"
        }]
    }
```

6 CASE STUDY

We apply a case study to validate King Louie by analysing H1, focusing on the impact of cloud resource failures. Therefore, eight availability evaluation scenarios are specified, comprising Apache Cassandra (CA) and Couchbase (CB) in two cluster sizes (CS), a write-heavy IoT driven workload, a read-heavy Web driven workload and one failure specification. It is noteworthy that these specifications represent only a small frame of the supported evaluation scenarios supported by King Louie.

6.1 DBMS Specification

CA and CB are selected as both are popular NoSQL DBMS[3] that promise high-availability and have already achieved promising results [19, 21]. Both provide a comparable multi-master architecture with a configurable replication factor. As they differ with respect to their consistency mechanisms, client-side consistency settings can not exactly be mapped for both DBMS. CA applies write ahead logging (WAL), while CB caches records directly in memory and persists them to disk asynchronously. The case study focuses on availability and performance optimized DBMS configurations by applying a replication factor of three and weak client consistency settings as detailed in Table 1. For CA, a write and read consistency of *ONE*[4] is applied, while for CB the write consistency replicateTo=0 (R=0) and persistTo=0 (p=0) is applied, specifying the confirmation of a write as successful after the record is transmitted[5]. Conceptually, this setup overcomes two DBMS node failures. For CB, the automatic failover mechanism is explicitly enabled with the minimum configurable failover time of 30 seconds while in CA it is enabled by default with a default failover time of 10 seconds.

[3]https://db-engines.com/en/ranking
[4]https://docs.datastax.com/en/archived/cql/3.3/cql/cql_reference/cqlshConsistency.html
[5]https://docs.couchbase.com/java-sdk/2.2/durability.html#persistto-and-replicateto

Table 1: DBMS Runtime Specifications

Specification	Apache Cassandra (CA)	Couchbase (CB)
Version	3.11.2	5.0.1 community
Cluster size	3,7	3,7
Replication	3	3
Write consistency	ONE	P=0, R=0
Read consistency	ONE	default

6.2 Cloud Resource Specification

The cloud resource configurations for the DBMS nodes and W-API instance is detailed in Table 2. All physical hosts hosting VM_DBMS types use a two SSDs Raid-0 configuration. Each DBMS instance is configured to use 50% of the available memory for its operation.

Table 2: Cloud Resource Specifications

Specification	VM_DBMS	VM_WORKLOAD
Cloud	private OpenStack, version Rocky	
VM vCores	2	8
VM RAM	4GB	8GB
VM Disk	70GB	20GB
OS	Ubuntu Server 16.04	
network	private, 10GbE	

6.3 Workload Specification

The W-API of King Louie provides the Yahoo Cloud Serving Benchmark (YCSB) [10] in version 0.15.0 King Louie. The YCSB is selected as it is a widely adopted workload to evaluate distributed DBMS. Moreover, it supports a variety of distributed DBMS by implementing the client connection in a baseline approach without adding specialised code to improve availability on client side. In order to demonstrate the flexibility of our approach, we define two typical DBMS workloads, one write-heavy workload and one read-heavy workload as described in Table 3.

Table 3: Workload Specifications

Specification	write-heavy	read-heavy
Metric granularity	10 s	
Initial records	0	2.000.000
Record size	5KB	
Operations	8.000.000	
Operation type	100% write	95% read, 5% update
Read distribution	N/A	zipfian
client threads	4,16	4,16

6.4 Failure Specification

While King Louie supports an extensive number of potential failure scenarios, we define a baseline failure specification to get first insights into the availability of distributed DBMS. As presented in Table 4, the failure specification comprises a permanent cloud resource failure on VM level and the respective recovery steps for permanent failures as depicted in Figure 1. Hereby, specific delays are specified to ensure a stable workload before the respective failure injection and recovery actions are triggered.

Table 4: Failure Specification

Specification	VM_FAILURE
Failure delay	180s write-heavy, 500s read-heavy
Failure type	VM
Severity	permanent
Recovery delay	180s
Recovery type	add new DBMS node

7 AVAILABILITY EVALUATION RESULTS

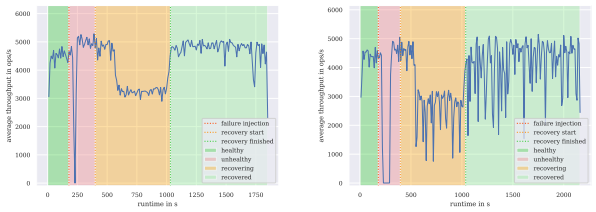
The case study results in 16 availability scenarios, *i.e.* $2\ DBMS\ types * 2\ cluster\ sizes * 2\ workload\ types * 2\ workload\ intensities * 1\ failure$. Each availability scenario is executed 10 times, resulting in 160 data sets. Each data set contains raw performance metrics, phase-specific throughput metrics for additional processing, monitoring data and auto-generated plots. Due to space constraints, we omit the presentation of the 16 threads workload intensity, but to ensure transparency, all collected evaluation data is publicly available [31].

First, for each data set, an explorative analysis based on the PT is executed by comparing the throughput graphs with H1. Based on this comparison, a set of reoccurring throughput development patterns are identified for each evaluation scenario. If different patterns are identified, the data sets are classified into the resulting pattern. For each identified pattern per availability scenario, one exemplary plot is provided in while all plots are available online[6]. In the following, we term each identified availability scenario pattern according the scheme *DBMS TYPE_CLUSTER SIZE_PATTERN ID*,

Second, for each availability scenario a descriptive analysis is carried out, resulting in measurement tables. Each Table contains the introduced phase-specific metrics (*cf.* Section 4.1) by providing the average phase throughput (APT) including the standard deviation. The downtime threshold for all evaluation scenarios is set to 10 ops/s, *i.e.* each reported throughput value below 10 ops/s is marked as downtime and the resulting downtime duration (PDD) is provided. In addition, the occurrence ratio (OR) of each pattern over the ten executions per availability scenario is provided. The RSR is provided as average over each 10 executions.

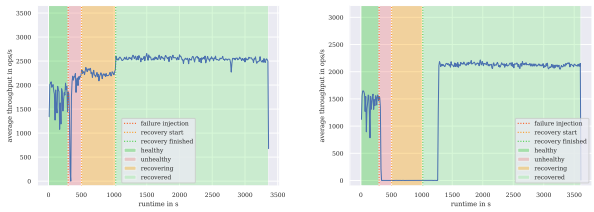
7.1 Apache Cassandra Results

7.1.1 Write-heavy Workload. The explorative analysis of the CA for the write-heavy workload results in two patterns as shown in Figure 6. Hereby, P1 and P2 reoccur for both cluster sizes. The graphs in Figure 6 clearly show that none of the identified patterns matches the expected hypothesis as for both cluster sizes a downtime occurs after the VM failure is injected. Moreover, the downtime varies from 10s to 70s as described in the corresponding measurement Table 5. Hereby, the OR of the 10s downtime, *i.e.* CA-CS3-P1 and CA-CS7-P1, is significantly higher than 70s downtime



(a) CA-CS3-P1 & CA-CS7-P1 (b) CA-CS3-P2 & CA-CS7-P2

Figure 6: Cassandra patterns for write-heavy workload



(a) CA-CS3-P1 & CA-CS7-P1 (b) CA-CS3-P2 & CA-CS7-P2

Figure 7: Cassandra patterns for read-heavy workload

patterns, *i.e.* CA-CS3-P2 and CA-CS7-P2. The fact that the throughput in the unhealthy phase remains on a similar or even slightly higher average compared to the healthy phase relates to the applied weak write consistency and replication factor of three. As in the unhealthy phase, one replica is missing, less internal replication operations are executed while there are still enough replicas available to satisfy the applied write consistency. The decreased throughput during the recovery phases relates to the redistribution of data to the newly added DBMS node. Moreover, this finding is validated by existing elasticity studies [21, 30]. The throughput variations during the recovered phase (*cf.* Table 5) are caused by internal compaction mechanisms and potential resource interferences due to the fact that the I/O bandwidth of the physical server hosting the DBMS VMs is shared with multiple tenants.

In order to analyse the unexpected throughput development, the raw YCSB results are checked for potential issues on client-side, but none are present in the result files. While P1 is partially explainable as 10s is the default request timeout before a new DBMS node takes over[6], this should only affect the client threads trying to write to the unavailable DBMS node and consequently the throughput should only decrease for their stalled write requests while the results clearly show that the overall throughput reaches the downtime threshold. Moreover, the less frequent P2 can not be correlated to Apache Cassandra configuration options.

7.1.2 Read-heavy Workload. The read-heavy result also show two throughput patterns P1 and P2 for both cluster sizes as depicted in Figure 7. Again, these patterns do not match the expected throughput development of H1, they even show an increased downtime compared to the write-heavy results. As Figure 7 depicts and the according measurement Table 5 shows, the downtime ranges from 10s to 240s, spanning over the whole unhealthy, recovery and even into the recovered phase. Again the YCSB raw results do not show any issues. While the 10s downtime is again partially explainable due to the default request timeout[7], the 240s downtime does not correlate with any CA configuration settings on client and DBMS side. A second deviation from the expected throughput development, the increased APT in the recovered phase compared to the initial phase is noteworthy. The explanation refers to the workload type as for the read-heavy workload, the longer phase duration of the recovered phase improves the caching of frequently accessed records. Compared to the write-heavy workload, the impact of additional cloud tenants is barely present as the read-heavy workload is mainly memory- and compute-bound and not I/O-bound.

7.2 Couchbase Results

7.2.1 Write-Heavy Workload. For CB three patterns are identified for the write-heavy workload, CB-CS3-P1, CB-CS3-P2 and CB-CS7-P1 as depicted in Figure 8 and the corresponding measurement Table 6. The three node patterns, CB-CS3-P1 and CB-CS3-P1, show that the VM failure causes an unexpected availability of the DBMS on the client-side, either with the beginning of the recovery phase (CB-CS3-P1) or even before the recovery phase (CB-CS3-P2). The resulting YCSB results show frequent occurrence of timeout exceptions that cause the termination of the workload even as enough replicas are available. For seven node cluster, CB-CS7-P1, the availability of the cluster is provided with the reoccurring pattern shown in Figure 8c. Hereby, a downtime of 30s is identified that correlates to the configured take-over time of CB. CB-CS7-P1 also shows a reoccurring throughput decrease in the recovered phase indicates due to ongoing CB internal management processes. The results indicate that for a write-heavy workload, a CB size of three nodes is not sufficient to overcome one node failure, even with a weak write consistency applied.

7.2.2 Read-Heavy Workload. The identified patterns of the read-heavy workload are depicted in Figure 9 and the correlating Table 6. Compared to the write-heavy workload patterns, they do not show any persistent downtime. Pattern CB-CS3-P1 shows a downtime of 30s and the less frequent CB-CS3-P2 a downtime of 175s (*cf.* Table 6). For the seven node cluster, only the pattern CB-CS7-P1 with the downtime of 30s is identified. Hereby, the 30s downtime matches again the configured take-over time, but also shows that a DBMS node failure causes a downtime for all clients,even if replicas could handle the read requests. Regarding the 175s downtime, the YCSB result files show temporary failure exceptions during the recovering phase and disappear in the recovered phase.

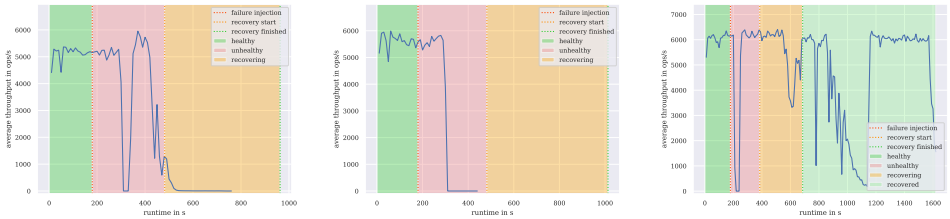
7.3 Threats to Validity

Even with the support of King Louie, the complexity of the cloud, DBMS and workload domain requires the consideration of the following aspects for adopting the results. For cloud domain, the recovery phase duration can be influenced by provider dependent VM

[6] https://docs.datastax.com/en/archived/cassandra/3.0/cassandra/operations/opsRepairNodesHintedHandoff.html

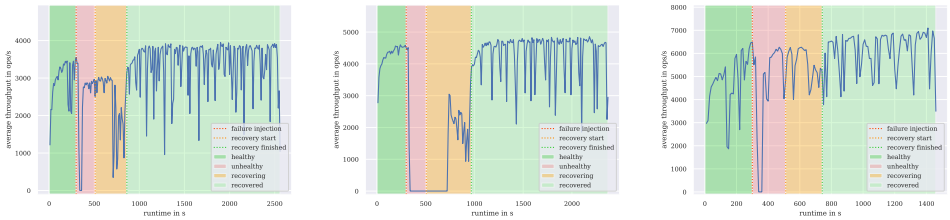
Table 5: Cassandra Phase Measurements

		write-heavy				read-heavy			
Phase	Metric	CA-CS3-P1	CA-CS3-P2	CA-CS7-P1	CA-CS7-P2	CA-CS3-P1	CA-CS3-P2	CA-CS7-P1	CA-CS7-P2
all	OR	7/10	3/10	9/10	1/10	7/10	3/10	7/10	3/10
all	RSR	99.99%	99.99%	99.99%	99.99%	99.65%	99.74%	100.00%	99.99%
initial	APT (σ)	4305 (80)	4254 (71)	4718 (64)	4779 (0)	1399 (157)	1345 (68)	2950 (89)	2816 (99)
unhealthy	APT (σ)	4384 (266)	2886 (215)	4518 (60)	3248 (0)	1728 (133)	8 (1)	3336 (41)	27 (1)
	PDD	10s	70s	10s	70s	10s	240s	10	240
recovering	APT (σ)	3361 (251)	2746 (106)	4324 (71)	4448 (0)	1967 (111)	1 (0)	3512 (42)	2 (0)
recovered	APT (σ)	4301 (490)	4071 (48)	4364 (143)	4262 (0)	2179 (152)	1855 (57)	3650 (21)	2928 (42)



(a) CB-CS3-P1 (b) CB-CS3-P2 (c) CB-CS7-P1

Figure 8: Couchbase patterns for write-heavy workload



(a) CB-CS3-P1 (b) CB-CS3-P2 (c) CB-CS7-P1

Figure 9: Couchbase patterns for read-heavy workload

Table 6: Couchbase Phase Measurements

		write-heavy			read-heavy		
Phase	Metric	CB-CS3-P1	CB-CS3-P2	CB-CS7-P1	CB-CS3-P1	CB-CS3-P2	CB-CS7-P1
all	OR	7/10	3/10	10/10	8/10	2/10	10/10
all	RSR	23.61%	18.19%	99.99%	99.99%	99.99%	99.99%
initial	APT (σ)	5172 (136)	6130 (96)	4718 (64)	3205 (692)	3691 (600)	4975 (140)
unhealthy	APT (σ)	3219 (77)	17 (24)	4925 (86)	2470 (525)	100 (22)	4643 (216)
	PDD	30s	∞	30s	30s	175s	30s
recovering	APT (σ)	82 (65)	0	5554 (57)	2581 (801)	867 (250)	5360 (403)
recovered	APT (σ)	0	0	4592 (52)	3205 (697)	3489 (843)	6213 (170)

spawning times and the DBMS performance in general can be impacted by resource interferences on different levels. As King Louie supports multiple cloud providers, these challenges can easily be addressed by additional case studies. With respect to the DBMS domain, vanilla DBMS installations are applied to ensure comparable results but there are numerous DBMS specific configurations that

can affect the availability results. Consequently, additional DBMS specific case studies need to be executed. As the DBMS catalogues of King Louie are easily extensible, DBMS specific configurations can be integrated for DBMS specific studies. For the workload domain, the usage of the YCSB enables comparability by supporting multiple DBMS, reporting DBMS performance metrics and failed requests. Yet,the implementation for the DBMS connections solely rely on the DBMS drivers and follow a basic approach without applying any dedicated availability mechanisms on client side. Consequently, the results represent a baseline and pave the path to evaluate enhanced implementations on client side as the YCSB is easily extensible and King Louie also supports custom workloads via the W-API.

Finally, the case study focuses on the client side while availability on the DBMS side is another important aspect. The 80 data sets show that even after the recovery phase, the cluster state can remain unhealthy. This is reported two times for CA and eight times for CB. These results are repeated and the unhealthy ones are excluded from the presented results. Hence, the root cause analysis of these cluster states requires further investigation on the DBMS side.

7.4 Lessons Learned

Based on the case study results, we provide a set of lessons learned on the availability of cloud-hosted DBMS in the case of failures: *(i)* as none of the results match the expected H1, there is even an increasing need for King Louie to further evaluate potential impact factors; *(ii)* the reproducibility of King Louie enables the identification of the different availability patterns in the first place; *(iii)* resource failures might not affect the availability on the DBMS side, but on the client side severe downtimes can occur or even client connections can be interrupted; *(iv)* potential throughput drops or even temporary downtimes need to be taken into account by client applications and respective mechanisms need to be in place to handle such downtimes; *(v)* as the results also show only particular explainable results, additional availability evaluations are required, considering DBMS specific configurations and metrics; *(vi)* even building upon established DBMS, guaranteeing DBMS availability and ensuring certain client performance is a challenging task that becomes even more challeing for automated operation as preferred in the cloud. Hence, comprehensive additional research with respect to DBMS behaviour and suitable COTs is required.

8 KING LOUIE VALIDATION

This section validates King Louie features against the introduced requirements and discusses potential limitations and extensions.

Evaluation design (R1) is enabled by providing comprehensive evaluation scenario templates that enable reusability and multiple customization parameters[1]. In addition, a sequence of multiple failures including optional recovery mechanism can be specified. *Abstraction (R2)* is addressed by building upon COTs for abstracting the communication with cloud provider APIs and the usage of DBMS and workload catalogues to abstract the technical details for deploying the DBMS and executing workloads. King Louie supports the specification of custom properties in the evaluation scenario, but the respective technical implementation needs to be able to process these custom properties. Its abstraction is shown by applying the availability evaluation for two DBMS of different cluster sizes and workload types. Its abstraction of different cloud providers has been validated in previous work [29]. *Failure injection (R3)* is validated by the case study by applying the VM failure type including the corresponding recovery mechanisms. While King Louie provides the framework for arbitrary failure injection, it currently supports the injection of failures on the VM and availability zone level. Additional failures can be easily added by implementing a single interface. *Automation (R4)* is demonstrated by executing 160 evaluation scenarios, generating the data sets for post-processing the availability metrics. Yet, the post-processing is not automatically executed by King Louie. *Extensibility (R5)* is addressed by building upon a loosely coupled architecture and applying the concept of domain-specific catalogues. This results in extensibility on different levels as adding or customizing DBMS deployments only requires the update of the respective scripts. Adding new failure types only requires the implementation of an interface in the evaluation orchestrator. *Transparent data (R6)* to ensure significance and reproducibility is provided by extensive data sets comprising raw performance metrics, but also monitoring data and execution logs to enable the full traceability of each evaluation execution [31].

As the availability evaluation of distributed DBMS in the volatile cloud context is a challenging task that requires the technical knowledge of multiple domains, King Louie provides a novel framework to support such evaluations by enabling reproducibility and portability. As the case study results show, there is a significant need in the thorough availability evaluation of cloud-hosted DBMS in order to provide high-available cloud services.

9 RELATED WORK

The evolving heterogeneity of distributed DBMS [11, 12, 22] has lead to numerous comparative evaluations of cloud-hosted distributed DBMS. A variety of DBMS workloads have been established for evaluating distributed DBMS in the context of Web and Big Data applications [25, 27]. Evaluations building upon these workloads focus either on the sole DBMS performance and scalability [17], the performance impact of different cloud resources [21]. With respect to availability evaluations, only the aspect of overload situations due to increasing workload intensities are evaluated [20, 30]. Yet, these studies do not consider the aspect of resource failures.

As the number of cloud resource offerings and distributed DBMS is still increasing, supportive evaluation frameworks are required to enable reproducible and portable evaluations of cloud-hosted DBMS [7, 23, 27]. Yet, none of them provides the capabilities of cloud resource failure injection and consequently none of them supports the evaluation of the introduced availability metrics.

A first approach towards analysing the availability of distributed DBMS is presented by [13] by studying existing DBMS bugs with respect to failure recovery mechanisms. Yet, the study focuses only on the DBMS layer and does not support failure injection nor the evaluation of client-side availability metrics. Building upon the "chaos monkey" concept, [32] present a framework to inject random cloud resource failures into cloud-hosted applications. While King Louie builds upon this concept, [32] does not evaluate any availability metrics nor is it intended for DBMS as failures are injected randomly which hinders reproducible and comparable evaluations. The Gremlin approach [18] presents a failure injection framework

for evaluating the availability of micro-services. A failure model describes the failures on the network level and injects them into a running system, analysing the latency impact on the client-side. Yet, Gremlin only considers statelss applications network failures, without focusing on stateful services and resource failures.

Consequently, a supportive framework for evaluating the availability of distributed DBMS under different cloud resource failures is missing. Therefore, King Louie addresses these limitations by enabling reproducible and portable availability evaluations for distributed DBMS by supporting cloud resource failure injection.

10 CONCLUSION

Cloud resources are a preferred solution to operate distributed DBMS. Yet, with increasing DBMS cluster sizes, the probability of cloud failures increases.Therefore distributed DBMS implement heterogeneous availability mechanisms and expose a diverse set of runtime configurations. Yet, in case of cloud resource failures, it remains unclear to what extent these mechanisms are able to ensure availability and guarantee stable performance. As supportive benchmarks are missing, we define a comprehensive availability evaluation methodology that ensures reproducibility and portability. Based on this methodology, we introduce the novel King Louie framework that automates the full evaluation process, including the cloud failure injection. We validate King Louie in a client-centric case study for different DBMS, cluster sizes and workload types resulting in 16 evaluation scenarios with 160 data sets. The results show novel and unexpected insights with respect to the availability and performance development in case of cloud resource failures.

Future work will focus on advanced evaluations that comprise multiple cloud failures and recovery steps. Moreover, machine learning techniques will be applied to derive efficient failure mitigation strategies. Also, King Louie will be applied to evaluate the availability from the DBMS side, including data related aspects such as accessibility and consistency.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the Federal Ministry of Education and Research of Germany under grant agreement 01IS18068 (SORRIR) and the EC's Framework Programme HORIZON 2020 under grant agreements 731664 (MELODIC) and 732667 (RECAP).

REFERENCES

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. 2010. A View of Cloud Computing. *Commun. ACM* 53, 4 (April 2010), 50–58.

[2] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing* 1, 1 (2004), 11–33.

[3] D. Baur, D. Seybold, F. Griesinger, H. Masata, and J. Domaschka. 2018. A Provider-Agnostic Approach to Multi-cloud Orchestration Using a Constraint Language. In *CCGRID*. 173–182.

[4] Daniel Baur, Daniel Seybold, Frank Griesinger, Athanasios Tsitsipas, Christopher B Hauser, and Jörg Domaschka. 2015. Cloud Orchestration Features: Are Tools Fit for Purpose?. In *IEEE/ACM UCC*.

[5] David Bermbach. 2014. *Benchmarking eventually consistent distributed storage systems*. KIT Scientific Publishing Karlsruhe.

[6] David Bermbach, Erik Wittern, and Stefan Tai. 2017. *Cloud service benchmarking*. Springer.

[7] Lexi Brent and Alan Fekete. 2019. A Versatile Framework for Painless Benchmarking of Database Management Systems. In *Australasian Database Conference*. Springer, 45–56.

[8] Eric Brewer. 2012. CAP twelve years later: How the" rules" have changed. *Computer* (2012).

[9] Eric A Brewer. 2000. Towards robust distributed systems. In *PODC*.

[10] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *ACM SoCC*.

[11] Ali Davoudian, Liu Chen, and Mengchi Liu. 2018. A Survey on NoSQL Stores. *ACM Computing Surveys (CSUR)* 51, 2 (2018), 40.

[12] Jörg Domaschka, Christopher B Hauser, and Benjamin Erb. 2014. Reliability and availability properties of distributed database systems. In *Enterprise Distributed Object Computing Conference (EDOC), 2014 IEEE 18th International*. IEEE, 226–233.

[13] Yu Gao, Wensheng Dou, Feng Qin, Chushu Gao, Dong Wang, Jun Wei, Ruirui Huang, Li Zhou, and Yongming Wu. 2018. An empirical study on crash recovery bugs in large-scale distributed systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 539–550.

[14] Anne Geraci, Freny Katki, Louise McMonegal, Bennett Meyer, John Lane, Paul Wilson, Jane Radatz, Mary Yee, Hugh Porteous, and Fredrick Springsteel. 1991. *IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries*. IEEE Press.

[15] Jim Gray. 1992. *Benchmark handbook: for database and transaction processing systems*.

[16] Haryadi S Gunawi, Mingzhe Hao, Riza O Suminto, Agung Laksono, Anang D Satria, Jeffry Adityatama, and Kurnia J Eliazar. 2016. Why does the cloud stop computing?: Lessons from hundreds of service outages. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*. ACM, 1–16.

[17] Abdeltawab Hendawi, Jayant Gupta, Liu Jiayi, Ankur Teredesai, Ramakrishnan Naveen, Shah Mohak, and Mohamed Ali. 2018. Distributed NoSQL Data Stores: Performance Analysis and a Case Study. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 1937–1944.

[18] Victor Heorhiadi, Shriram Rajagopalan, Hani Jamjoom, Michael K Reiter, and Vyas Sekar. 2016. Gremlin: Systematic resilience testing of microservices. In *2016 IEEE 36th International Conference on Distributed Computing Systems*. IEEE.

[19] Murtadha AI Hubail, Ali Alsuliman, Michael Blow, Michael Carey, Dmitry Lychagin, Ian Maxon, and Till Westmann. 2019. Couchbase Analytics: NoETL for Scalable NoSQL Data Analysis. *Proc. VLDB Endow.* 12, 12 (Aug. 2019), 2275–2286.

[20] Ioannis Konstantinou, Evangelos Angelou, Christina Boumpouka, Dimitrios Tsoumakos, and Nectarios Koziris. 2011. On the elasticity of NoSQL databases over cloud management platforms. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 2385–2388.

[21] Jörn Kuhlenkamp, Markus Klems, and Oliver Röss. 2014. Benchmarking scalability and elasticity of distributed database systems. *Proceedings of the VLDB Endowment* 7, 12 (2014), 1219–1230.

[22] Somnath Mazumdar, Daniel Seybold, Kyriakos Kritikos, and Yiannis Verginadis. 2019. A survey on data storage and placement methodologies for Cloud-Big Data ecosystem. *Journal of Big Data* 6, 1 (2019), 15.

[23] A. V. Papadopoulos, L. Versluis, A. Bauer, N. Herbst, J. Von Kistowski, A. Ali-eldin, C. Abad, J. N. Amaral, P. Tůma, and A. Iosup. 2019. Methodological Principles for Reproducible Performance Evaluation in Cloud Computing. *IEEE Transactions on Software Engineering* (2019), 1–1.

[24] Kia Rahmani, Kartik Nagar, Benjamin Delaware, and Suresh Jagannathan. 2019. CLOTHO: Directed Test Generation for Weakly Consistent Database Systems. *Proc. ACM Program. Lang.* 3, OOPSLA, Article 117 (Oct. 2019), 28 pages.

[25] Vincent Reniers, Dimitri Van Landuyt, Ansar Rafique, and Wouter Joosen. 2017. On the state of nosql benchmarks. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. ACM, 107–112.

[26] Sherif Sakr. 2014. Cloud-hosted databases: technologies, challenges and opportunities. *Cluster Computing* 17, 2 (2014), 487–502.

[27] Daniel Seybold and Jörg Domaschka. 2017. Is Distributed Database Evaluation Cloud-Ready?. In *ADBIS*. Springer, 100–108.

[28] Daniel Seybold, Christopher B Hauser, Simon Volpert, and Jörg Domaschka. 2017. Gibbon: An Availability Evaluation Framework for Distributed Databases. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 31–49.

[29] Daniel Seybold, Moritz Keppler, Daniel Gründler, and Jörg Domaschka. 2019. Mowgli: Finding your way in the DBMS jungle. In *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering (ICPE '19)*. ACM, 321–332.

[30] Daniel Seybold, Nicolas Wagner, Benjamin Erb, and Jörg Domaschka. 2016. Is elasticity of scalable databases a Myth?. In *IEEE Big Data*.

[31] Daniel Seybold, Stefan Wesner, and Jörg Domaschka. 2019. King Louie: DBMS Availability Evaluation Data Sets. https://doi.org/10.5281/zenodo.3459604

[32] Ariel Tseitlin. 2013. The Antifragile Organization. *Commun. ACM* 56, 8 (2013).

[33] Werner Vogels. 2009. Eventually Consistent. *Commun. ACM* 52, 1 (Jan. 2009), 5.

[34] Ming Zhong, Kai Shen, and Joel Seiferas. 2008. Replication Degree Customization for High Availability. In *EuroSys*.

Chapter 15

[core8] Is elasticity of scalable databases a Myth?

This article is published as follows:

Daniel Seybold, Nicolas Wagner, Benjamin Erb, and Jörg Domaschka. "Is elasticity of scalable databases a Myth?" in *4th IEEE International Conference on Big Data (Big Data)*, 2016, IEEE, pp. 2827–2836, DOI: https://doi.org/10.1109/BigData.2016.7840931.

Is Elasticity of Scalable Databases a Myth?

Daniel Seybold*, Nicolas Wagner*, Benjamin Erb† and Jörg Domaschka*
*Institute of Information Resource Management
†Institute of Distributed Systems
Ulm University, Ulm, Germany
Email: {daniel.seybold, nicolas.wagner, benjamin.erb, joerg.domaschka}@uni-ulm.de

***Abstract*—The age of cloud computing has introduced all the mechanisms needed to elastically scale distributed, cloud-enabled applications. At roughly the same time, NoSQL databases have been proclaimed as the scalable alternative to relational databases. Since then, NoSQL databases are a core component of many large-scale distributed applications.**

This paper evaluates the scalability and elasticity features of the three widely used NoSQL database systems Couchbase, Cassandra and MongoDB under various workloads and settings using throughput and latency as metrics. The numbers show that the three database systems have dramatically different baselines with respect to both metrics and also behave unexpected when scaling out. For instance, while Couchbase's throughput increases by 17% when scaled out from 1 to 4 nodes, MongoDB's throughput decreases by more than 50%. These surprising results show that not all tested NoSQL databases do scale as expected and even worse, in some cases scaling harms performances.

I. Introduction

The evolvement of cloud computing has gained tremendous focus in industry and academia, especially for web-based applications. With the typical benefits of cloud computing such as on-demand self-service, resource pooling or rapid elasticity [1] also traditional web service architectures experienced the rethinking from monolithic structures to distributed services. Whereas the distribution of the mostly stateless business logic services fits well for distribution in the cloud, the distribution of stateful database services is more challenging. Hence, in parallel to cloud computing, a distributable database class, the NoSQL databases, evolved and proclaim as alternative for traditional relational databases. NoSQL databases store data in non-relational way and promise to be scalable and to run on commodity hardware as offered by the cloud.

These developments in the database area also have influenced the service models of the initial cloud service models [1] Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) by extending them with more fined grained service models such as Database as a Service (DBaaS) [2]. Yet, the selection of the distributed database still remains as challenge.

Within the last years, a high number the existing of NoSQL databases reached a mature state and nearly all of them promise up to "unlimited" scalability by utilising cloud resources. In this context we provide an up-to-date evaluation of the common NoSQL databases, Apache Cassandra, Couchbase and MongoDB, with the focus on their scalability and elasticity capabilities for a DBaaS system.

Our contribution is twofold, *(1)* we define a thorough methodology to evaluate the scalability and elasticity of distributed databases; *(2)* we evaluate the mentioned NoSQL databases in a distributed setup with respect to their specific scalability and elasticity capabilities and discuss the results with respect to an expected behaviour.

The remainder of the paper is organised as follows. Section II describes the challenges of scalability and elasticity with respect to DBaaS. In Section III, we introduce the cloud architecture, distributed databases and distributed storage models. Section IV defines the methodology that we used for our evaluation. Section V describes the cloud environment of our evaluation. Section VI presents our evaluation results, followed by a discussion in Section VII. Section VIII discusses the related work. Finally, Section IX concludes and outlines future work.

II. Database Challenges for DBaaS

In the following we define the terms *scalability* and *elasticity* for our considerations in the context of a DBaaS, with respect to their definitions in cloud computing and for distributed databases.

A. Scalability

The term scalability is a common term in IT in general. As we focus in our work on the scalability of distributed databases in the cloud, we narrow down scalability with respect to cloud computing and database systems. In the cloud context a well-known definition is provided by Herbst et. al. [3] with *"Scalability is the ability of the system to sustain increasing workloads by making use of additional resources"*.

Database system workloads can be classified in *memory-bound* (i.e. the data fits into memory) and *storage-bound* workloads (i.e. fractions of the data is kept in memory and the rest in persistent storage) [4]. In this paper, we only focus on the scaling of memory-bound workloads. A definition of scalability for distributed systems in general and with respect to distributed databases is provided by Agrawal et. al. [5], defining the terms scale-up, scale-out and scale-in in order to manage growing workloads. Scale up or vertically scaling applies by adding more computing resources to a single node. This paper only focuses on the two horizontal scaling actions, namely scale out (i.e. adding nodes to a cluster) and scale in (i.e. removing nodes from a cluster).

For applying a distributed database in a DBaaS a high scalability factor is required to process virtually unlimited workload sizes by scaling the database cluster to a sufficient size. A high scalability factor is represented by constant latency and proportionally growing throughput with respect to the number of nodes and the workload size [6].

B. Elasticity

Scalability focuses on the general ability to process arbitrary workload sizes. Elasticity is tightly coupled to scalability and enables the overcoming of sudden workload fluctuations by scaling the cluster without any downtime. With respect to the cloud, elasticity is defined as *"Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand."* [1]

A definition with the focus on distributed databases is provided by Agrawal et. al. [5] with *"Elasticity, i.e. the ability to deal with load variations by adding more resources during high load or consolidating the tenants to fewer nodes when the load decreases, all in a live system without service disruption, is therefore critical for these systems."*

Hence, a DBaaS architecture requires a high elasticity factor, defined by scaling the cluster at run-time without downtime and improving the throughput to resolve the workload fluctuations [6].

III. Database as a Service Architecture

In this section, we introduce the technical architecture for our following evaluation. This potential architecture of a DBaaS covers the underlying cloud infrastructure, the scalability-enabled architecture of distributed databases and the corresponding storage models.

A. Cloud Infrastructure

A DBaaS system requires a high degree of flexibility on the resource level, so that it can be built upon a bare metal infrastructure or upon the IaaS cloud service model. Based on cloud features such as resource pooling or rapid elasticity [1], IaaS offers more flexibility than bare metal and is therefore the preferable way to build a DBaaS system.

IaaS provides processing, storage, networks and other fundamental computing resources to run arbitrary software [1]. The processing and storage resources are typically encapsulated in a virtual machine (VM) entity that also includes the operating system (OS). VMs run on the virtualised physical infrastructure of the IaaS provider, which is not accessible to the user. An exemplary IaaS architecture is depicted in Fig. 1.

B. Distributed Databases Architectures

Distributed databases provide a single database system to the user which is spread over multiple nodes as depicted in Fig. 1. A single database instance as part of the overall distributed database system is termed database node in the following. The overall distributed database system is termed database cluster. Our evaluation only considers distributed, shared-nothing database clusters where each database node resides on its own VM as shown in Fig. 1.

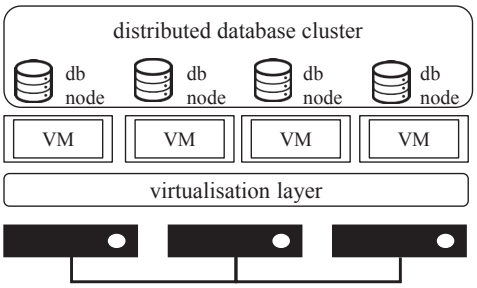


Fig. 1: System Model

The architecture of distributed databases is typically categorised into the following distribution models [7]:

1) Single Server: A single server architecture represents the simplest option without any distribution at all. The single node handles all read and write requests. We do not consider single server architectures as they do not offer horizontal scalability.

2) Master-Slave: In a master-slave distribution the data is replicated across multiple nodes. One node represents the designated master node, executing all write requests and synchronizing the slave nodes, which only execute read requests. The master-slave distribution can be applied to scale read-heavy workloads.

3) Multi-Master: The multi-master or peer-to-peer distribution do not build upon different node types, i.e. all nodes are equal. In a multi-master distribution replication and sharding [7] of the data is applied to spread write and read requests across all nodes of the cluster.

C. Storage Models

Distributed databases are commonly classified according to the following storage models.

In *relational data stores*, data is stored in tuples, forming an ordered set of attributes. Relations consist of sets of tuples while a tuple is a row, an attribute is a column and a relation forms a table. Tables are defined using a normalised data schema. SQL has established itself as a generic data definition, manipulation and query language for relational data. *Column-oriented data stores* store data by columns rather than by rows. This enables to store large amounts of data in bulk and allows for efficiently querying over very large, structured data sets. A column-oriented data model does not rely on a fixed schema. *Key/value data stores* relate to hash tables of programming languages. The storage records are tuples consisting of key/value pairs. While the key uniquely identifies an entry, the value is an arbitrary chunk of data. *Document-oriented data stores* are similar to key/value stores. However, they define a structure on the values in structured formats such as XML or JSON. Compared to key/value stores, document stores allow more complex queries, as document properties can be used for indexing and querying. Inspired by the graph theory, *graph-based data stores* rely on graph structures for data modelling. Hence, vertices and edges represent and contain data. Queries often relate to graph traversals.

IV. Methodology

Based on the presented challenges for DBaaS, this section elaborates on our methodology to evaluate the scalability and elasticity of three distributed databases. The following section describes the methodology used to select the databases, to decide on the workload issued on the databases and the way it is generated. Finally, we define two evaluation scenarios for capturing the scalability and elasticity properties of the databases.

A. Database Selection

With the emergence of cloud computing the number of available distributed database systems increased as well. However, not all storage models fit well to scalability and elasticity requirements. Relational databases offer distributed representatives such as MySQL Cluster[1] or Postgres-XL[2]. Yet, neither MYSQL Cluster[3] nor Postgres-XL[4] support elasticity, as resizing the cluster will lead to a downtime for both databases. Additionally, relational databases offer limited horizontal scaling capabilities compared to NoSQL databases [8]. Therefore, we do not consider relational databases for our evaluation. We also do not consider graph databases, as their elasticity is either very limited, e.g. neo4j[5] provides scale-out capabilities only for reads, or the elasticity properties rely on the underlying storage backends (e.g. Titan[6] with Cassandra). The specific structure of graph-based data and the corresponding queries make it also difficult to compare its performance metrics.

Hence, we set the focus of our evaluation databases to the remaining storage models: column-oriented, key-value and document-oriented. Below, we provide a brief overview of the selected databases and their selection criteria. All of them provide automated data sharding and promise high scalability and elasticity.

1) Apache Cassandra: Apache Cassandra[7] is selected as one of the most common column-oriented data stores[8]. Cassandra aims to run on top of commodity infrastructure and scale up to hundreds of nodes. Its architecture follows the multi-master paradigm and is designed to handle high write throughput without sacrificing read efficiency [9].

2) Couchbase: Couchbase is a document-oriented data store which can also be operated as a key-value data store. Its architecture follows the multi-master paradigm and takes advantage of memcached[9] for in-memory caching. We select Couchbase because of these features and also due to fact that Couchbase has gained industry attention [10], [11] for its scalability. Yet, Couchbase has undergone only limited scientific evaluation so far.

[1]https://www.mysql.com/products/cluster/
[2]http://www.postgres-xl.org/
[3]http://dev.mysql.com/doc/mysql-cluster-manager/1.4/en/
[4]http://files.postgres-xl.org/documentation/tutorial-createcluster.html
[5]https://neo4j.com/
[6]http://titan.thinkaurelius.com/
[7]http://cassandra.apache.org/
[8]http://db-engines.com/en/ranking_trend
[9]https://memcached.org/

3) MongoDB: MongoDB[10] has been selected as the most prevalent document-oriented data store[8]. MongoDB's architecture is built upon three different services: (1) `mongos` act as query router, providing an interface between clients and a sharded cluster, (2) `configs` store the metadata of the cluster, (3) `shards` contain the actual data.

B. YCSB Benchmarking Tool

In the course of the database systems evolution, the variety of database benchmarking tools grew as well. Common representatives of such database benchmarking tools include TPC Benchmarks[11], RUBiS [12] and the Yahoo Cloud Serving Benchmark (YCSB) [6].

While TPC workloads (TPC-C and TPC-E) target online transaction processing (OLTP) applications and RUBiS workloads target a typical 3-tier web application, YCSB offers simple database-centric workloads based on create, read, update and delete (CRUD) operations. Therefore, we chose the YCSB tool as it enables a comparable evaluation of the selected databases and their corresponding data models.

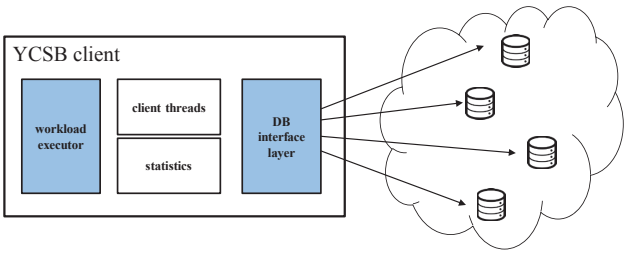


Fig. 2: YCSB client architecture

1) YCSB Architecture: YCSB offers an extensible architecture shown in Fig. 2. A YCSB client comprises the four depicted modules, *workload creator, client threads, statistics* and the *database interface layer*. The workload creator and the database interface layer offer a convenient interface to customise or extend the respective module. The design of YCSB enables the usage of an arbitrary number of YCSB clients in a distributed setup to generate the desired load.

2) YCSB Metrics: The statistics module of YCSB aggregates and reports the latency metrics (average, 95th and 99th percentile, minimum and maximum). The throughput is aggregated and reported as average and actual throughput at a specific timestamp. We focus in our evaluation on the average throughput and latency and the time series representation of the throughput to evaluate the elasticity of distributed databases, which will be described in more detail in the remainder of this section.

C. Workloads and Workload Generation

As introduced in Section IV-B YCSB issues CRUD operations by default. In addition, it provides a set of built-in distributions to perform the required random choices during

[10]https://www.mongodb.com/
[11]http://www.tpc.org/default.asp

the load generation. The supported distributions are *uniform, zipfian, latest* and *multinomial*. We choose the zipfian distribution for our workloads, as it represents a typical distribution of web workloads [13]—some records are very popular and are often accessed, while most of the records are not.

For the distribution of the operations, we select two common workloads of cloud based applications [6]: read-heavy and a read-update. The distribution of operations as well as the selected distribution of each workload is provided in Table I.

TABLE I: Composition of Chosen Workloads

Workload	Create	Read	Update	Delete	Distribution
Read-heavy	0.0%	0.95%	0.05%	0.00%	zipfian
Read-update	0.0%	0.50%	0.50%	0.00%	zipfian

D. Scalability Evaluation Scenario

The scalability evaluation bases on static cluster sizes, which allow to evaluate the scalability of the selected databases by comparing the average throughput and latency for each cluster size and determine and compare *throughput* and *latency* across the different cluster sizes.

Four static cluster sizes and a fixed number of YCSB clients are considered as depicted in Fig. 3. The cluster sizes comprise a 1-node, 2-node, 4-node and 8-node cluster of the selected databases (cf. Section IV-A). Small node sizes highlight the effects induced by changes to the cluster size. For all three databases and all four cluster sizes, we use the same, fixed number of YCSB clients to generate the workload for each database cluster.

All evaluations are performed on a previously loaded database cluster containing 1,000,000 records. All scalability evaluation runs comprise 10,000,000 operations that are equally distributed among all YCSB clients involved. We select the size of the database (i.e. 1,000,000 records) such that it entirely fits in RAM, which allows us to exclude disk I/O as bottleneck and to focus on distribution and scaling effects.

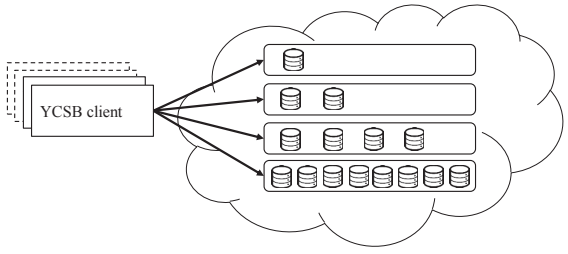


Fig. 3: Static Cluster

In order to achieve comparable results for all databases we calibrate the number of YCSB clients. For the scalability evaluation scenario, we determine the number of YCSB clients such that all database clusters are subject to comparable load. In particular, we pick the number of clients such that no database node is overloaded (>90% CPU utilisation) nor idling (<20% CPU utilisation), even in a 1-node configuration.

On the database side, no specific configurations are made. Only for MongoDB, we needed to decide on a distribution of the components not available for the two other database types (`mongos` and `configs`; cf. Section IV-A3). Therefore we monitored the resource consumption of each component and derived an appropriate distribution which is described in Section VI-A.

E. Elasticity Evaluation Scenario

The elastic evaluation scenario investigates a sudden growth in workload can be overcome by scaling the cluster under load. Starting from a 1-node cluster, the cluster is scaled-out as soon as the node running the database gets overloaded, i.e. the throughput drops significantly. We use the CPU utilisation (> 90%) and a drop in throughput as indicators for an overloaded node and extend the cluster when both conditions occur. In order to ensure overload is reached for all three databases, we vary the number of YCSB clients accordingly (cf. Fig. 4).

When an overloaded state for the 1-node cluster is reached, we manually add a new node. During that time, the load is kept at the same level. Based on the time series representation of the current throughput, we are able to evaluate the elasticity during the redistribution of the data and in the resulting 2-node cluster of each database.

As before, the 1-node cluster starts with 1,000,000 preloaded records. For operations, we need to guarantee a high load over a longer period of time, ensuring the overload state is reached, the redistribution of the data is finished and the database has time to stabilise. Therefore, we increase the total number of operations to 500,000,000, which are again distributed over all YCSB clients.

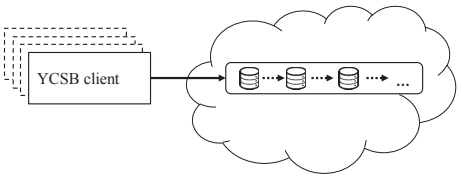


Fig. 4: Elastic Cluster

V. Evaluation Environment

The overall evaluation takes place in a private, OpenStack-based cloud[12] with full and isolated access to all physical and virtual resources. The cloud infrastructure[13] is operated with OpenStack version Kilo. In order to reduce possible side effects and to guarantee reproducible results we use the *availability zones* feature of OpenStack to dedicate physical hosts for spawning the required VMs. Hence, we dedicate one physical host to the YSCB clients and one to the database nodes. This reflects the optimal VM distribution for a distributed database in a cloud environment as one physical host is capable to run all database nodes (cf. Table II) and no

[12]https://www.openstack.org/

[13]https://www.uni-ulm.de/in/omi/institut/blog/details/article/our-openstack-physical-testbed-part-3up-to-date-with-software-and-requirements/

additional communication on a physical level is required. In the following, we describe the technical details of the cloud infrastructure, the YCSB client and database VMs.

A. Physical Infrastructure

The VMs for the YCSB clients and database nodes are placed on a different physical host. Table II shows the technical details for the physical hosts of the YCSB clients and the database nodes. The physical hosts are connected via 1Gb.

TABLE II: Physical Host Details

	YCSB Client Host	Database Host
CPU	2xIntel Xeon E5-2630v3 8-Core Haswell 2.4Ghz	2xIntel Xeon E5-2637 8-Core SandyBridge 2.6Ghz
Memory	64 GB ECC DDR4	64 GB ECC DDR4
Storage	2x1 TB HDD 7.2k rpm	NAS
Network	gigabit ethernet	gigabit ethernet
Operating System	CentOS 7, Linux 4.2.0-1.el7.elrepo.x86_64	CentOS 7, Linux 4.2.0-1.el7.elrepo.x86_64
Hypervisor	KVM, QEMU 1.5.3	KVM, QEMU 1.5.3

B. YCSB Client

The YCSB client VMs are provisioned with the details shown Table III. Observing the resource consumption of YCSB has shown that, with respect to the VM flavour, the mainly consumed resource is CPU whereas memory and disk I/O are negligible. Hence, the provisioned VM flavour is focused on CPU. For the YCSB tool we use our fork[14] of the original YCSB version 0.8 with an updated version of the Couchbase DB interface from 1.4.10 to 2.2.2. Additional details of YCSB configuration are shown in Table IV.

TABLE III: YCSB VM Details

VM Detail	Configuration
CPU	4 vCores
Memory	2 GB
Storage	10 GB
Network	gigabit ethernet
Operating System	Ubuntu Server 14.04.2 AMD64 LTS 3.13.0-59-generic x86_64
YCSB Detail	**Configuration**
Version	0.8
# of records	1.000.000
record size	1 KB
# of operations (for scalability scenario)	10.000.000
# of operations (for elasticity scenario)	500.000.000
# of threads	20

C. Database Node

Each database VM is provisioned with the details shown in Table IV. The size of the database VMs represents a typical commodity hardware configuration[15] and is selected to fit the required system requirements of all the selected databases.

The specific version of the evaluated databases is shown in the second part of Table IV. In order to guarantee a fair comparison we use each database in its vanilla version without custom optimisations. We only configure the available memory for each database to 6 GB to reserve 2 GB for the operating system. Each database is configured with the minimum replication configuration.

TABLE IV: DB VM Details

VM Detail	Configuration
CPU	4 vCores
Memory	8GB
Storage	80GB
Network	gigabit ethernet
Operating System	Ubuntu Server 14.04.2 AMD64 LTS 3.13.0-59-generic x86_64
DB Detail	**Configuration**
Apache Cassandra version	2.2.6
Couchbase version	4.0.0 community edition
MongoDB version	3.2.7

VI. Evaluation Results

In this section, we present our scalability and elasticity evaluation results. For the scalability evaluation, we show the throughput and latency development for static cluster sizes. For the elasticity evaluation, a dynamic cluster is scaled at runtime under continuous load and the throughput development is presented as time series for each database.

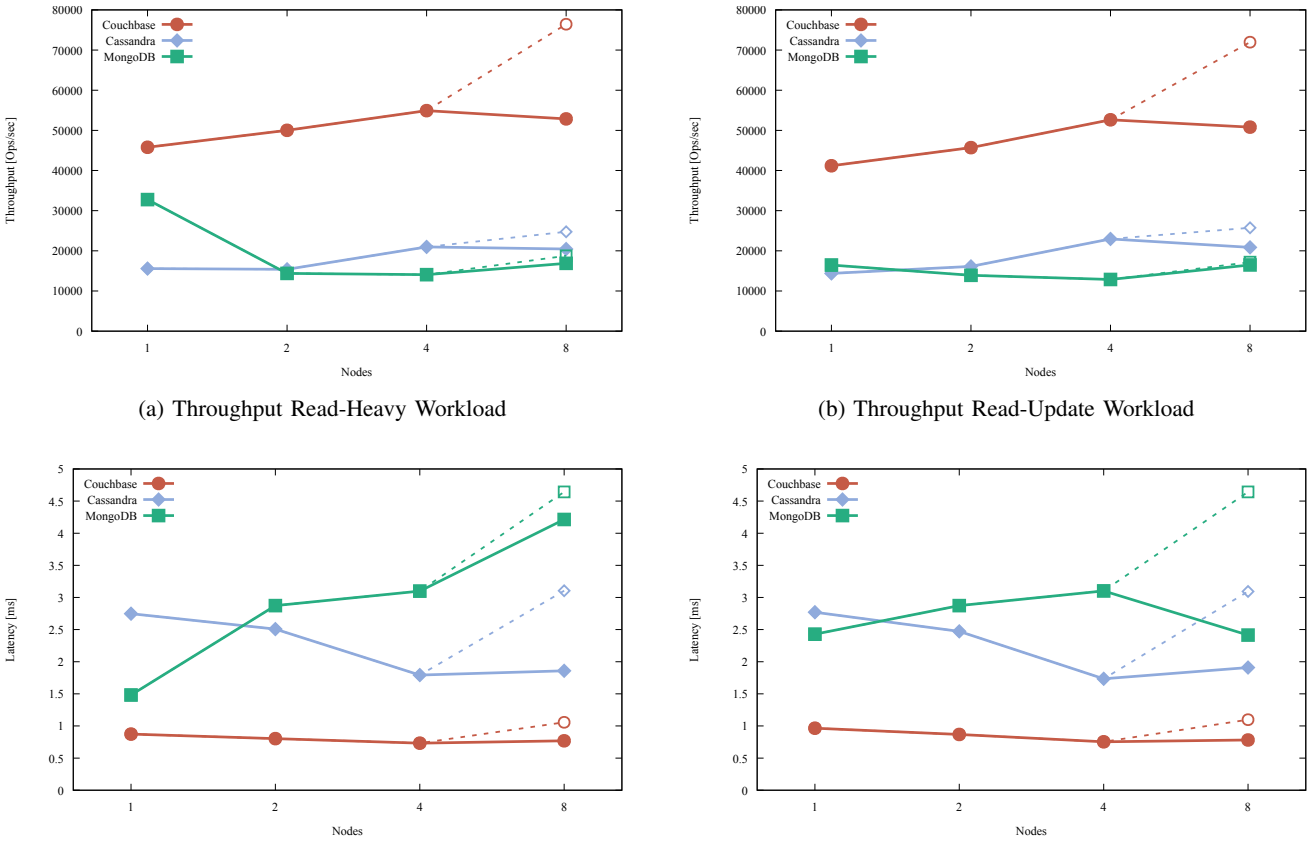
A. Calibrating the Scalability Scenario

Initial evaluation results show that three YCSB clients overload a MongoDB 1-node cluster, so that two clients is the maximum possible amount of clients. At the same time, using only one client barely loads Couchbase. Hence, we decide to use two YCSB clients for the entire scalability scenario. For any kind of 1-node and 2-node clusters, all database nodes stay within the load boundaries (cf. Section IV-D). The calibration results for 8-node cluster show that 2 YCSB clients are not sufficient to saturate Couchbase and Cassandra. Hence, all evaluations of the 8-node clusters are performed with 2 and 4 YCSB clients to ensure saturation and comparable results. Regarding the distribution of the different MongoDB nodes our calibration efforts show that the `shard` services create the main load. The `mongos` and `config` services each only generate a negligible load of $<$10% CPU utilisation. Therefore, for the 1-node cluster, we run all three MongoDB services in the same virtual machine. For larger clusters, we only deploy new shard services. This setup still allows a fair evaluation against the other databases

During the calibration, we execute the read-heavy and read-update workloads 10 times in a 2-node Couchbase cluster environment. We encounter a standard deviation of the average throughput of $<$500 operations per second. This leads us to the conclusion that our private cloud environment causes only little noise and provides stable results. In consequence, we

[14]https://github.com/seybi87/YCSB/releases/tag/scalability-evaluation

[15]https://docs.mongodb.com/manual/administration/production-notes/#hardware-considerations



(a) Throughput Read-Heavy Workload

(b) Throughput Read-Update Workload

(c) Latency Read-Heavy Workload

(d) Latency Read-Update Workload

Fig. 5: Troughput and latency results of different workloads.

consider a repetition factor of 3 for all scalability evaluations as sufficient.

In order to exclude the network as possible bottleneck for the evaluation results, we monitor the YCSB clients and database nodes using Ganglia[16]. As depicted in Fig. 6 the network load on a single YCSB client with 20 threads reaches at most ≈600 KB/sec (in/out combined). Accordingly, in worst case (cf. Table V), we can expect ≈3.0MB/s at a database node, which is far below the maximum bandwidth provided by our 1Gb network and allows us to neglect network saturation.

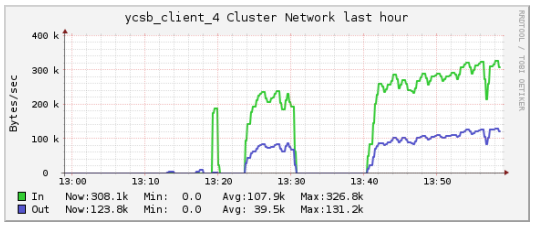


Fig. 6: YCSB Network Load

[16]http://ganglia.info/

B. Results of the Scalability Evaluation Scenario

In the following, we present the results of the scalability evaluation scenario. For both workloads we describe the measured throughput and latency per database and with growing cluster sizes. The dashed lines show the additional results by using 4 YCSB clients in an 8-node cluster (cf. Section VI-A).

1) Read-Heavy Workload: Fig. 5a depicts the throughput achieved by all databases in all cluster configurations. It is noticeable that Couchbase has the highest overall throughput for all cluster sizes. It is capable of increasing throughput by 17% when going from a 1-node to a 4-node cluster. Increasing the cluster size from 4 to 8 nodes decreases the throughput by 4% because 2 YCSB clients are not sufficient to saturate the 8-node cluster. By using 4 YCSB clients instead the throughput again increases by 29%.

Cassandra achieves the lowest throughput in a single node cluster, but outperforms MongoDB for the 2-node and 4-node cluster due to a throughput increase of 26%. Similar to Couchbase, the throughput decreases by 1% in an 8-node cluster with 2 YCSB clients but increases by 16% using 4 YCSB clients.

In contrast to its competitors, MongoDB performs better as single node than in a cluster. Its throughput decreases by 58% when moving from a 1-node to a 4-node cluster. For an 8-node

cluster throughput increases by 18% for 2 YCSB clients and by 26% for 4 YCSB clients compared to a 4-node cluster.

The latency results shown in Fig. 5c unveil a similar ranking as for the throughput. Couchbase achieves the lowest latency for all cluster sizes. Increasing its cluster to four nodes decreases latency by 17% compared to a 1-node configuration. In a 8-node cluster the latency remains constant for 2 YCSB and increases for 4 YCSB clients by 44% compared to a 4-node cluster.

Cassandra benefits the most from increasing the cluster size as a 35% latency decrease from one to 4 nodes is achieved. For eight nodes the latency increases by 3% for 2 clients and by 73% for 4 clients.

MongoDB reaches the highest latency for all cluster sizes and even with an increasing cluster size the latency increases by 100% from one to four nodes. In an 8-node cluster the latency increases by 35% for two and by 49% for four YCSB clients.

2) Read-Update Workload: For the read-update workload we measure similar trends as for the read-heavy results: Couchbase achieves the highest overall throughput and the lowest latency. Throughput is summarised in Fig. 5b, latency in Fig. 5d: Couchbase increases its throughput by 22% from a 1-node to a 4-node cluster and by 36% for an 8-node cluster with 4 YCSB clients. Cassandra again reaches the highest throughput increase of 38% from 1-4 nodes and 12% in an 8-node cluster with 4 YCSB clients. MongoDB's throughput decreases by 27% from 1-4 nodes and increases by 28% (2 YCSB clients) and by 33% (4 YCSB clients) in an 8-node cluster. Couchbase achieves a latency decrease of 20% when increasing the cluster size from one to four nodes. For 8 nodes and 2 YCSB clients the latency remains constant and for 4 YCSB clients it increases by 45%. For Cassandra we measure a 37% latency decrease from 1-4 nodes and for 8 nodes an increase of 10% for 2 YCSB clients and 78% for 4 YCSB clients. The latency of MongoDB increases by 38% from 1-4 nodes and decreases in an 8-node cluster with 2 YCSB clients by 22%. With 4 YCSB clients the latency increases by 49%.

C. Calibrating the Elasticity Scenario

As the elasticity evaluation bases on an overloaded 1-node database cluster (cf. Section IV-E) the required number of YCSB clients has to be determined for each database separately. For that purpose, for each database we proceed as follows: Starting with three clients (cf.

Section VI-A), we issue load on a 1-node cluster of that database, while monitoring CPU usage and throughput. Then, we increase the number of clients until the metrics show a constant overload situation. The resulting number of required YCSB clients is summarised in Table V.

For the distribution of MongoDB services, we use the same approach as in the scalability scenario.

The calibration results show that a benchmark run-time of 15 minutes is sufficient time for all the databases to redistribute the data and balance the load across the nodes. Therefore, each benchmark run is manually terminated after 15 minutes.

TABLE V: Number of clients used for the elasticity scenario.

	Cassandra	Couchbase	MongoDB
Clients	4	5	3

D. Results for the Elasticity Evaluation Scenario

The evaluation of the elastic cluster is performed with the read-update workload and a each database is previously loaded with 1,000,000 records.

1) Apache Cassandra: For Cassandra our experimental results show that four YCSB clients are required to create an overload situation for a single node. The four YCSB clients are started in 30s gaps to ensure a warm-up time for each YCSB client. We continuously monitor the throughput of the Cassandra instance and the CPU utilisation of the hosting VM. The throughput graph of Cassandra's throughput is depicted in Fig. 7a as time series. All four YCSB are running after 90s and produce the overload situation with a dropping throughput at 110s. In addition the CPU utilisation reaches >90% at this point. At 120s a new Cassandra node is added to the cluster and the data is getting rebalanced. As depicted in Fig. 7a the throughput stops dropping after rebalancing the data but the 2-node cluster does not reach the throughput of the one node cluster in the beginning of the benchmark.

2) Couchbase: Our experimental results have shown that five YCSB Clients are required to overload one Couchbase node. Again, the YCSB clients are started in 30s gaps and all clients are started after 120s. As depicted in Fig. 7b the throughput starts dropping at 130s and a second node is added at 150s. After rebalancing the data, the two node cluster increases the throughput significantly and compared to the initial one node cluster

3) MongoDB: For overloading a single MongoDB node, three YCSB clients are required. The three clients are started in 30s gaps and produce an overload situation for the MongoDB instance at 100s as depicted in Fig. 7c. By adding a second node at 120s the overload situation is balanced as the throughput increases but only slightly above the throughput of the initial one node cluster.

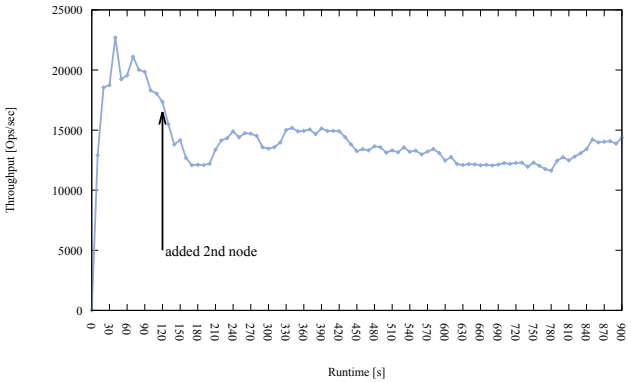
VII. Discussion

The main insight from our evaluation is that *more instances* does not necessarily mean *better performance*. Indeed, in the case of MongoDB more instances decreases throughput and increases latency.

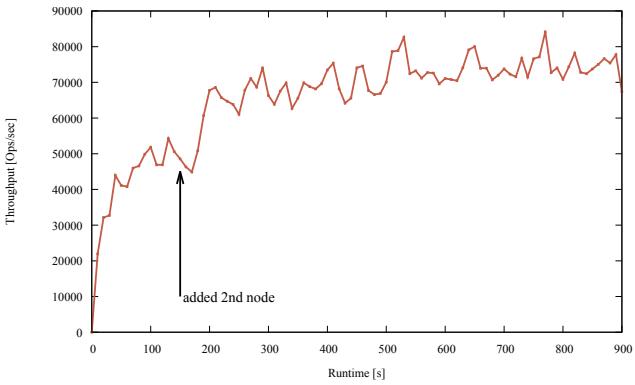
This section discusses the results from a high level view. It clarifies the impact of the evaluation methodology on the results received and draws conclusions. It also addresses impact on the design of distributed architectures.

A. Interpretation of Scalability Results

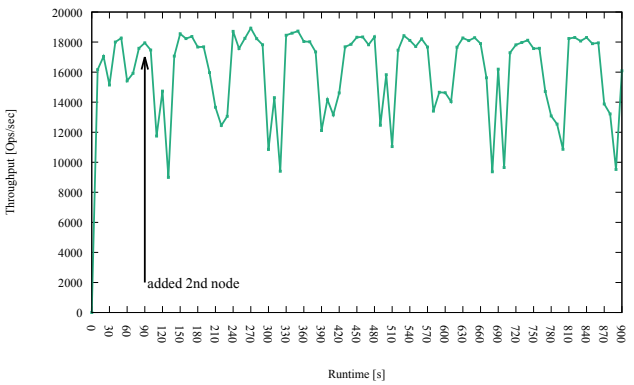
Key to the interpretation of the scalability evaluation is the fact that for all three databases, two clients were used (the four clients results in the 8-node cluster are discussed at the end). Both clients use 20 threads leading to a maximum



(a) Cassandra



(b) Couchbase



(c) MongoDB

Fig. 7: Elasticity benchmark results: adding an additional node while the database system is under load.

parallelism of 40 requests. This number was selected such that none of the databases was overloaded in a single node set-up. In consequence, for the scalability evaluation, no database instance was overloaded, and the performance metrics of the databases are directly comparable.

In an ideal case going from one to two nodes, throughput would double and going from one to four nodes it would quadruple (cf. Section II-A). The fact that this has not happened could have three causes: *(i)* the clients or the network are maximally utilised and so that clients cannot issue new queries and updates faster (we excluded this by carefully calibrating the number of clients and checking the network bandwidth). *(ii)* The database instances are overloaded (this is prevented by the set-up). *(iii)* Queuing at the databases occurs.

In case a single database instance can only serve $n < 40$ parallel requests this will cause the remaining requests to be queued. As each client thread issues only one request at a time, this load will cause the system to adjust and lead to an (on average) constant queue size. Adding a second database node to the cluster will decrease the load on the first one, as data is sharded between the two nodes. Due to using a zipfian distribution for the access pattern, an unequal distribution of load amongst nodes could be expected. As all evaluated databases apply hash-based sharding in the used default configuration, the prevention of hot spot nodes is addressed at database level. Yet, even with an uniform load distribution, throughput would only double when each database instance could process $n \geq 20$ requests at the same time. As we do not see a large increase in throughput for any of the databases (even not for 4-node clusters), we conclude that the parallelism for each database is $n < 10$.

Looking at the results for the 8-node cluster, the throughput decrease compared to a 4-node cluster might be surprising in the first place. Yet, it leads us to the assumption of a too large cluster size in respect to the issued load by two YCSB clients. Hence, an overhead of inter-cluster communication is caused by request routing as the results show no further latency decrease compared to a 4-node cluster. Increasing the load by using four YCSB clients confirms our assumption as the clusters are saturated and the throughput increases in respect to two YCSB clients. Yet, the increased load also increases the latency significantly for all three databases.

This analysis is accurate for Couchbase, which does not apply any replication strategy out of the box. The impact of their lazy (i.e. asynchronous) replication on throughput behaviour for Cassandra and MongoDB remains unclear. Also, the routing strategy of the respective database can impact the throughput (as well as the latency): while Couchbase and Cassandra route messages through the database nodes in a peer-to-peer manner, MongoDB uses a centralised routing service. Even if this service does not impose measurable load on a virtual machine, the fact that all messages from the clients have to proceed through one place, may have significant impact on the drop of throughput and increase in latency when adding more MongoDB instances. A reason may be that MongoDB relies on smart indexes for achieving high throughput; YCSB does only use simple indexing to provide comparable results.

B. Interpretation of Elasticity Results

The elasticity results raise even more questions than the results of the scalability evaluation. Only Couchbase exposes a behaviour a naive observer would expect. Adding a second node first takes some time for synchronisation, which also

causes a drop in throughput. After a while the system is fully operational again and is able to serve more requests per time unit than before.

In contrast to Couchbase, Cassandra requires more time to synchronise the new node with the existing cluster. While the drop in throughput (caused by the overload situation) stops at some point, Cassandra will not recover from the decrease, but continue performing at a lower average throughput than at peak time. This is remarkable as the 2-cluster throughput for the elasticity scenario stays below the 2-cluster throughput for the scalability. Here, the only difference is that the elasticity evaluation uses four clients instead of two for the scalability evaluation. Similar holds for MongoDB where the throughput does not change at all after scaling out. For this database, also the reason for the periodic drops in throughput raise questions. While we assume that garbage collection, internal compaction processes or similar mechanisms cause them, this need to be verified.

C. Looking Ahead: Refining the Methodology

The scalability and elasticity evaluation provided throughout this paper has led to several surprising results. We on purpose go for a purely observative approach describing *how* the systems behaves. While we had not expected to achieve near linear performance when scaling out the databases—in particular not without tweaking their configurations—it is astonishing to see that after more than four decades of distributed systems research, the use of sharding can still lead to decreasing throughput and increasing latency.

The results gained here seem to indicate that we should move to a more analytical approach in order to understand *why* the platforms behave as they do. Nevertheless, we assume that such an approach is less lasting, as it is vulnerable to version upgrades and other changes to the algorithms used by the database implementation.

Instead, we suggest to take the work here as a starting point to develop an evaluation framework that is able to repeatedly and reliably characterise the scaling and elasticity behaviour of distributed databases. In order to do that, we shall refine our methodology. This allows us to isolate different triggers on the behaviour of the database (e.g. more load vs. higher replication degree or the relation between throughput and latency). In the long run, this will have to involve the collection large amounts of monitoring data and the execution of advanced data analytics.

VIII. Related Work

DBaaS providers typically deploy a distributed database system on IaaS resources to offer a elastic database service on demand. One of the first DBaaS are offered by Amazon's DynamoDB[17] or Google's BigTable[18] which use proprietary NoSQL databases. With the expansion of NoSQL, emerging open source NoSQL databases are being considered as DBaaS storage backend, such as mlab[19] using MongoDB or OpenStack Trove[20] using Cassandra or Couchbase. Hence, performance evaluation of distributed databases is an ongoing research area.

Due to the evolution of distributed databases and cloud computing, new benchmarking tools are required with the focus on scalable and elastic databases. Released in 2010, YCSB [6] became the de facto standard benchmarking tool for distributed databases. The original version was developed by Yahoo to evaluate their column-oriented data store PNUTS [14] against Cassandra and HBase and the relational MySQL Cluster. The evaluation shows the scalability and elasticity results where Cassandra and PNUTS achieve the best results. The evaluation does not consider key-value and document-oriented data stores or overloading the database explicitly.

Rabl et. al. [4] evaluate six distributed databases (key-value, column-oriented and relational) in the light of storing monitoring data for application performance management (APM) systems. They use YCSB as benchmarking tool, but apply custom workloads reflecting typical APM workloads. Their evaluation focuses on two kind of workloads, memory-bound, i.e. records fitting completely in RAM and disk-bound, i.e. records do not completely fit into RAM. The cloud context is not considered as all evaluations are run on physical infrastructure. The results show for both workload types that Cassandra achieves the best scalability results while elasticity is not evaluated.

Gandini et. al. [15] evaluate the performance of three NoSQL databases (Cassandra, MongoDB and HBase) in the cloud. They are using Amazon EC2[21] as evaluation environment. The evaluation focuses on scalability in relation to the number of cores in a single-server setup and to the number of nodes in a distributed setup. In addition, they analyse how the replication degree influences performance. Their results show that Cassandra as well as MongoDB increase their throughput for static cluster sizes, whereas an increasing replication degree will decrease the throughput for both databases. However, the benchmarking methodology misses technical details like used database versions and YCSB version, which harms the comparison with our results.

Klein et al. [16] evaluate Cassandra, MongoDB and Riak in the light of a health care use case with a customised version of YCSB. The evaluation focuses on the performance and scalability with respect to differently strict consistency levels. Their results attest Cassandra the highest scalability, whereas similar to our results MongoDB does not scale.

Konstantinou et al. [17] analyse the distributed databases Cassandra, Hbase and Riak in the context of their TIRAMOLA framework for elastic NoSQL cluster provisioning. Their evaluation focuses on the cost in terms of time for scaling a database cluster and the optimal thresholds for executing a scaling action. Scalability and elasticity of distributed

[17] https://aws.amazon.com/dynamodb/?nc1=h_ls
[18] https://cloud.google.com/bigtable/
[19] https://mlab.com/
[20] https://wiki.openstack.org/wiki/Trove
[21] http://aws.amazon.com/ec2

databases in respect to context of cloud computing is considered by [6] however without performing the evaluation in a fully transparent cloud environment. The elasticity aspect is considered by scaling a cluster at run-time, but without creating an explicit overload situation in the database.

IX. Conclusion and Future Work

Cloud computing provides "unlimited" resources, which can be allocated on demand. Based on these prerequisites, distributed databases gained significant focus in recent years. Whereas traditional relational databases offer distributed derivates, a new class of distributed databases emerged, NoSQL databases. By promising horizontal scalability and elasticity, NoSQL databases are commonly used for the cloud service model Database as a Service (DBaaS).

In this paper, we present thoroughly evaluate the scalability and elasticity of the common NoSQL databases Cassandra, Couchbase and MongoDB with respect to DBaaS environment. Therefore, we define two evaluation scenarios, one for scalability and one for elasticity. The scalability evaluation comprises different static cluster sizes to determine the scalability with growing cluster sizes. The elasticity is evaluated by overloading a database and scaling-out the database at run-time to resolve the overload situation.

The scalability evaluation results show significant differences between the evaluated databases. Whereas Couchbase achieves the highest throughput and lowest latency results, Cassandra benefits the most from larger cluster sizes in contrast to MongoDB where a distributed setup even harms the performance. The elasticity results state that only Couchbase is able to resolve an overload situation at run-time in contrast to Cassandra and MongoDB.

The discussion concludes with the main outcome of the evaluation results: more instances does not necessarily mean better performance. An interpretation of the results is provided in greater depth by discussing possible bottlenecks on database and evaluation environment side. Whereas only Couchbase provides expected results for scalability and elasticity, the results of Cassandra and MongoDB raise additional questions for deeper evaluations. Therefore, the future work will focus on a deeper analysis of the gathered results by analysing the database distribution by including extensive resource monitoring and advanced data mining techniques. In addition, the evaluation of scalability and elasticity in larger scale clusters will also be targeted as well as the evaluation of distributed databases running on microservices platforms.

Acknowledgment

The research leading to these results has received funding from the EC's Framework Programme FP7/2007-2013 under grant agreement number 317715 (PaaSage) and the EC's Framework Programme HORIZON 2020 (ICT-07-2014) under grant agreement number 644690 (CloudSocket). In addition, we thank Alexander Rasputin for assistance in performing the evaluation.

References

[1] P. Mell and T. Grance, "The nist definition of cloud computing," 2011.

[2] S. Kächele, C. Spann, F. J. Hauck, and J. Domaschka, "Beyond iaas and paas: An extended cloud taxonomy for computation, storage and networking," in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*. IEEE Computer Society, 2013, pp. 75–82.

[3] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in cloud computing: What it is, and what it is not," in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, 2013, pp. 23–27.

[4] T. Rabl, S. Gómez-Villamor, M. Sadoghi, V. Muntés-Mulero, H.-A. Jacobsen, and S. Mankovskii, "Solving big data challenges for enterprise application performance management," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1724–1735, 2012.

[5] D. Agrawal, A. El Abbadi, S. Das, and A. J. Elmore, "Database scalability, elasticity, and autonomy in the cloud," in *International Conference on Database Systems for Advanced Applications*. Springer, 2011, pp. 2–15.

[6] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010, pp. 143–154.

[7] P. J. Sadalage and M. Fowler, *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2012.

[8] R. Cattell, "Scalable sql and nosql data stores," *Acm Sigmod Record*, vol. 39, no. 4, pp. 12–27, 2011.

[9] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[10] D. Nelubin and B. Engber, "Nosql failover characteristics: Aerospike, cassandra, couchbase, mongodb," *Thumbtack Technology, Inc., White Paper*, 2013.

[11] ——, "Ultra-high performance nosql benchmarking: Analyzing durability and performance tradeoffs," *Thumbtack Technology, Inc., White Paper*, 2013.

[12] E. Cecchet, A. Chanda, S. Elnikety, J. Marguerite, and W. Zwaenepoel, "Performance comparison of middleware architectures for generating dynamic web content," in *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*. Springer-Verlag New York, Inc., 2003, pp. 242–261.

[13] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1. IEEE, 1999, pp. 126–134.

[14] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "Pnuts: Yahoo!'s hosted data serving platform," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1277–1288, 2008.

[15] A. Gandini, M. Gribaudo, W. J. Knottenbelt, R. Osman, and P. Piazzolla, "Performance evaluation of nosql databases," in *European Workshop on Performance Engineering*. Springer, 2014, pp. 16–29.

[16] J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham, and C. Matser, "Performance evaluation of nosql databases: a case study," in *Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems*. ACM, 2015, pp. 5–10.

[17] I. Konstantinou, E. Angelou, C. Boumpouka, D. Tsoumakos, and N. Koziris, "On the elasticity of nosql databases over cloud management platforms," in *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 2011, pp. 2385–2388.

Chapter 16

[core9] Towards a Framework for Orchestrated Distributed Database Evaluation in the Cloud

This article is published as follows:

Daniel Seybold. "Towards a Framework for Orchestrated Distributed Database Evaluation in the Cloud", *18th Doctoral Symposium of the 18th International Middleware Conference (MIDDLEWARE)*, published 2017, ACM, DOI: https://doi.org/10.1145/3152688.3152693

Towards a Framework for Orchestrated Distributed Database Evaluation in the Cloud

Daniel Seybold
Institute of Information Resource Management
Ulm University
Ulm, Germany
daniel.seybold@uni-ulm.de

Abstract

The selection and operation of a distributed database management system (DDBMS) in the cloud is a challenging task as supportive evaluation frameworks miss orchestrated evaluation scenarios, hindering comparable and reproducible evaluations for heterogeneous cloud resources. We propose a novel evaluation approach that supports orchestrated evaluation scenarios for scalability, elasticity and availability by exploiting cloud resources. We highlight the challenges in evaluating DDBMSs in the cloud and introduce a cloud-centric framework for orchestrated DDBMS evaluation, enabling reproducible evaluations and significant rating indices.

CCS Concepts • **Information systems → Database performance evaluation**;

Keywords benchmarking, distributed database, cloud, NoSQL, NewSQL, orchestration

1 Introduction

In the last decade, database management systems (DBMSs) have evolved by focusing as well on distributed database management systems (DDBMSs) as evolving application domains such as the Web or Big Data impose new challenges to Online Transaction Processing (OLTP) [13]. Thus, a plethora of new DDBMSs have appeared on the DBMS landscape, which can be classified into NoSQL and NewSQL. These DDBMSs are built on a shared-nothing architecture and promise to cater for non-functional requirements such as scalability, elasticity, availability by running on commodity hardware or even on cloud resources. As cloud computing offers on-demand resource provisioning, the cloud seems to be the preferable solution to operate DDBMSs.

Yet, with the vast number of available DDBMSs and the heterogeneous cloud resource offerings, the selection of a DDBMS and its operation in the cloud becomes a challenging task. Assuming a DDBMS is required to continuously store social media data based on actual events. The data is read from a varying amount of users and periodically queried by an analytics engine. Hence, the DDBMS needs to *scale horizontally* in case of growing workloads and provide *elasticity* to handle sudden workload peaks, created by social media events and the resulting user requests. Cloud resources are used to run the DDBMS, ensuring the dynamic resource allocation on demand. Yet, as cloud resources can fail, the DDBMS needs to provide *availability* in case of cloud resource failures. Evaluating these requirements of existing DDBMS is a common approach to guide the DDBMS selection process. Yet, current evaluation frameworks (EFs) do not explicitly consider the usage of heterogeneous cloud resources and lack the support for orchestrated evaluation scenarios [11] with respect to scalability, elasticity and availability.

Middleware '17, Las Vegas, NV, USA

DOI: 10.1145/3152688.3152693

2 Problem Statement

Our research targets the enhancement of the DDBMS selection by providing a cloud-centric EF for the orchestrated scalability, elasticity and availability evaluation. In the scope of our research, we highlight the following key challenges:

Cloud Resource Characteristics needs to be considered by the EF as cloud resources tend to become more heterogeneous, from virtual machines to container technologies, various storage technologies and resource locality options. Thus, our EF will be aware of these characteristics and enables the access to cloud resources in a unified way, easing comprehensive, significant and reproducible results.

Orchestrated Evaluation Scenarios enable the evaluation of elasticity and availability in a comparable and reproducible way, by adapting the DDBMS topology by managing cloud resources. Consequently, predefined evaluation scenarios can be applied to generic DDBMS and cloud resource templates (CRTs) that will be executed by the EF.

Workload Domains such as OLTP or Hybrid Transaction and Analytical Processing (HTAP), are required for realistic evaluation scenarios. Thus, our EF will support domain-specific workload creation, based on synthetic and trace-based workloads.

Rating Indices on a DDBMS basis need to be computed based on the raw evaluation results to ease the DDBMS selection. While for established features such as performance, rating indices are available, comparable rating indices as elasticity and availability still need to be defined.

3 Related Work

One of the main drivers of non-functional DBMS feature evaluation is the Transaction Processing Performance Council (TPC)[1], providing EFs for the OLTP domain. Yet, TPC rather focuses on the performance of relational DBMS than on DDBMSs features such as elasticity, availability or the usage of cloud resources. DDBMSs centric EFs such as the Yahoo Cloud Serving Benchmark (YCSB) [5] and its extensions YCSB++ [10] and YCSB+T [6] support the evaluation of scalability and consistency based on synthetic workloads. Advanced workload domains are addressed by BG [2], LinkBench [1]

[1] http://www.tpc.org/default.asp

and OLTP-Bench [7], yet without the explicit consideration of cloud resources. Evaluating the scalability and elasticity of DDBMSs by using cloud resources is presented by [9]. Yet, the evaluation relies on synthetic workloads and does not consider multi-cloud scenarios. A first attempt to evaluate availability is presented by UPB [8] by measuring the performance impact in case of node failures, while advanced scenarios including cloud resource failures, DDBMS failover and recovery capabilities are not considered.

While existing EFs rely on static evaluation scenarios, which either focus on advanced workloads for evaluating performance or using synthetic workloads without considering heterogeneous cloud resource configurations, we propose a novel EF enabling the orchestrated evaluation of scalability, elasticity and availability for DDBMSs based on heterogeneous cloud resources and multiple workload types.

4 Approach

Our initial DDBMSs evaluation addresses scalability and elasticity of DDBMSs in the cloud [12]. Our results show significant differences with respect to elasticity and the need for orchestrated DDBMS evaluation in order to provide adaptive and reproducible evaluation scenarios. Consequently, we analyze existing EFs with the focus on their evaluation scenarios and their consideration of cloud resources [11]. As existing EFs do not yet support orchestrated evaluation scenarios, elasticity and availability evaluation lacks dedicated support. In addition, the impact of heterogeneous cloud resource is not addressed by existing EFs.

Hence, we propose a novel EF, enabling orchestrated evaluation scenarios with the focus on scalability, elasticity and availability of DDBMSs in the cloud. Its architecture is depicted in Figure 1. The *evaluation API* enables the specification of *evaluation scenarios* for scalability, elasticity and availability. Each evaluation scenario comprises the properties workload (synthetic/OLTP/HTAP); a CRT defining providers, locations and resource dimensions; and a DDBMS template provided by the *DDBMS/CRT repository*. The DDBMS template exposes a unified set of non-functional configuration options to ensure comparable evaluation of different DDBMSs. Each evaluation scenario can specify adaptation actions for *scalability*, *elasticity* and *availability* for evaluating their correlation with cost-, performance- or locality-optimized CRTs. The execution of the specified evaluation scenario is enabled by the *orchestrator* component, which is realized by a cloud orchestration tool [3]. The orchestrator unifies the cloud resources access, provisions the required cloud resources and orchestrates the DDBMS, the workload and the DDBMS adaptations at run-time. During the evaluation, system and DDBMS specific monitoring data is collected and stored by the *evaluation monitor*. Based on this monitoring data, the orchestrator is able to adapt the DDBMS automatically, according to the specified adaption actions of the evaluation scenario. The evaluation-specific metrics are collected by the *measurement collector* and retrieved by the *rating index processor* to compute significant rating indices.

5 Evaluation

The evaluation will follow a two-dimensional approach. The first dimension will evaluate the framework's features against distinguished guidelines of DBMS evaluation [4]. The second dimension comprises industry-driven evaluation scenarios for >= 5 DDBMSs

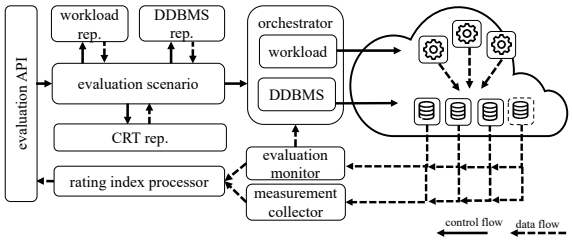


Figure 1. High-level evaluation framework architecture

based on cost-, locality and performance-optimized CRTs. The resulting scalability, elasticity and availability insights are analyzed towards their significance in the DDBMS selection process.

6 Conclusion

The vast DDBMS landscape requires new evaluation framework concepts, considering heterogeneous cloud resources and the orchestration of the evaluation, including the workload generation and the DDBMS deployment and adaptation. Thus, we propose a novel evaluation framework for the orchestrated DDBMS evaluation of scalability, elasticity and availability by explicitly addressing the usage of cloud resources.

7 Acknowledgements

This work is done under the supervision of Prof. Dr.-Ing. Stefan Wesner. The author would like to thank Dr. Jörg Domaschka for his support. The research leading to these results has received funding from the EC's Framework Programme HORIZON 2020 under grant agreement number 644690 (CloudSocket) and 731664 (MELODIC).

References

[1] Timothy G Armstrong, Vamsi Ponnekanti, Dhruba Borthakur, and Mark Callaghan. 2013. LinkBench: a database benchmark based on the Facebook social graph. In *ACM SIGMOD*.
[2] Sumita Barahmand and Shahram Ghandeharizadeh. 2013. BG: A Benchmark to Evaluate Interactive Social Networking Actions.. In *CIDR*.
[3] Daniel Baur, Daniel Seybold, Frank Griesinger, Athanasios Tsitsipas, Christopher B Hauser, and Jörg Domaschka. 2015. Cloud Orchestration Features: Are Tools Fit for Purpose?. In *IEEE/ACM UCC*.
[4] David Bermbach, Jörn Kuhlenkamp, Akon Dey, Sherif Sakr, and Raghunath Nambiar. 2014. Towards an extensible middleware for database benchmarking. In *TPCTC*.
[5] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *ACM SoCC*.
[6] Akon Dey, Alan Fekete, Raghunath Nambiar, and Uwe Rohm. 2014. YCSB+T: Benchmarking web-scale transactional databases. In *IEEE ICDEW*.
[7] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudre-Mauroux. 2013. Oltp-bench: An extensible testbed for benchmarking relational databases. *VLDB Endowment* (2013).
[8] Alessandro Gustavo Fior, Jorge Augusto Meira, Eduardo Cunha de Almeida, Ricardo Gonçalves Coelho, Marcos Didonet Del Fabro, and Yves Le Traon. 2013. Under pressure benchmark for ddbms availability. *JIDM* (2013).
[9] Jörn Kuhlenkamp, Markus Klems, and Oliver Röss. 2014. Benchmarking scalability and elasticity of distributed database systems. *VLDB Endowment* (2014).
[10] Swapnil Patil, Milo Polte, Kai Ren, Wittawat Tantisiriroj, Lin Xiao, Julio López, Garth Gibson, Adam Fuchs, and Billie Rinaldi. 2011. YCSB++: benchmarking and performance debugging advanced features in scalable table stores. In *ACM SoCC*.
[11] Daniel Seybold and Jörg Domaschka. 2017. Is Distributed Database Evaluation Cloud-Ready?. In *ADBIS*.
[12] Daniel Seybold, Nicolas Wagner, Benjamin Erb, and Jörg Domaschka. 2016. Is elasticity of scalable databases a Myth?. In *IEEE Big Data*.
[13] Michael Stonebraker. 2012. Newsql: An alternative to nosql and old sql for new oltp apps. *Communications of the ACM. Retrieved* (2012).

Chapter 17

[core10] The Impact of the Storage Tier: A Baseline Performance Analysis of Containerized DBMS

This article is published as follows:

Material from: Daniel Seybold, Christopher B. Hauser, Georg Eisenhart, Simon Volpert, and Jörg Domaschka. "The Impact of the Storage Tier: A Baseline Performance Analysis of Containerized DBMS", *24th International European Conference on Parallel and Distributed Computing (Euro-Par): Parallel Processing Workshops*, published 2018, Springer International Publishing, DOI: https://doi.org/10.1007/978-3-030-10549-5_8

The Impact of the Storage Tier: A Baseline Performance Analysis of Containerized DBMS

Daniel Seybold(✉), Christopher B. Hauser, Georg Eisenhart, Simon Volpert, and Jörg Domaschka

Institute of Information Resource Management, Ulm University, Ulm, Germany
{daniel.seybold,christopher.hauser,georg.eisenhart, simon.volpert,joerg.domaschka}@uni-ulm.de

Abstract. Containers emerged as cloud resource offerings. While the advantages of containers, such as easing the application deployment, orchestration and adaptation, work well for stateless applications, the feasibility of containerization of stateful applications, such as database management system (DBMS), still remains unclear due to potential performance overhead. The myriad of container operation models and storage backends even raises the complexity of operating a containerized DBMS. Here, we present an extensible evaluation methodology to identify performance overhead of a containerized DBMS by combining three operational models and two storage backends. For each combination a memory-bound and disk-bound workload is applied. The results show a clear performance overhead for containerized DBMS on top of virtual machines (VMs) compared to physical resources. Further, a containerized DBMS on top of VMs with different storage backends results in a tolerable performance overhead. Building upon these baseline results, we derive a set of open evaluation challenges for containerized DBMSs.

Keywords: Container · YCSB · Benchmarking · DBMS · MongoDB

1 Introduction

The raise of containers, containerization, and container orchestration [3] has a great influence on the structure of distributed applications, and greatly eased the operation of such systems by finally leveraging the realisation of continuous deployment. Support for containers is offered beside traditional virtual machine offerings by Amazon Elastic Container Service[1] and OpenStack Magnum[2].

Much of the success of containers is a consequence of the fact that they enable a quick installation of pre-packaged software components, which is a prerequisite for handing overload (scale out), bug fixing (software upgrade), and

[1] https://aws.amazon.com/ecs/.
[2] https://wiki.openstack.org/wiki/Magnum.

G. Mencagli et al. (Eds.): Euro-Par 2018 Workshops, LNCS 11339, pp. 93–105, 2019.
https://doi.org/10.1007/978-3-030-10549-5_8

replacing failed components (fault tolerance). All of these concepts work fine for mostly stateless components such as load balancers, web and application servers, message queues, and also caches. Yet, despite recent attention in the field [2,14], it is currently unclear to what extent containerization is suited for and beneficial to the operation of stateful applications. Database management systems (DBMS) are an important representative of this type of applications and a crucial part of Big Data and IoT applications.

While the containerization of DBMS particularly eases the usage of features of modern DBMS such as horizontal scalability or high availability, at least two challenges remain: *(a)* The general runtime overhead of containerized DBMS is unknown. *(b)* The container eco-system offers a myriad of different storage backends and their impacts on performance are also unclear.

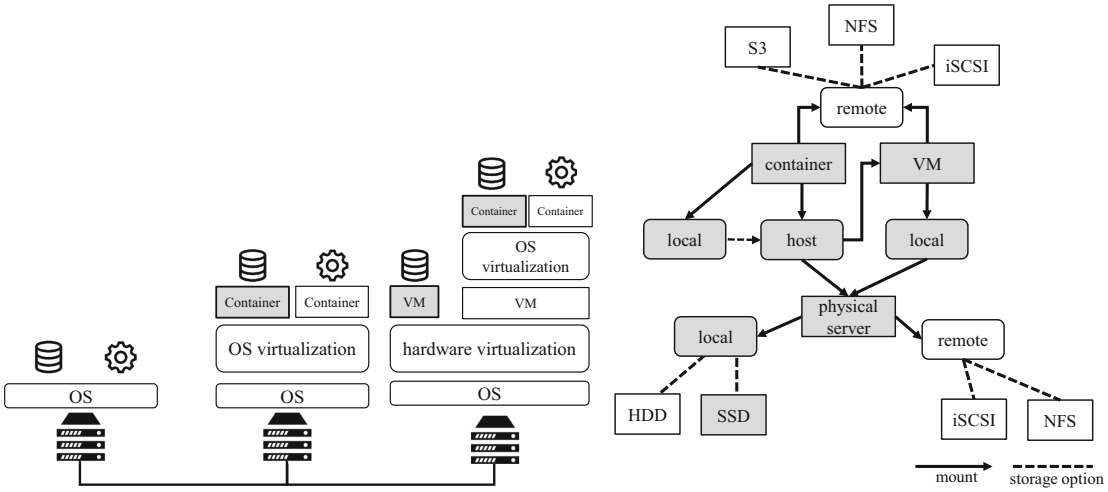
Only with an answer to these baseline questions, it is beneficial to think about more sophisticated questions such as placement of state and data migration. This paper is an initial step to identify further research and engineering challenges with respect to containerized DBMS. Our contributions are as follows: *(i)* We introduce three different operational models for DBMS ranging from bare metal to containers in virtual machines. *(ii)* We analyse the landscape of storage backends for containers and their pros and cons. *(iii)* For three operational models and two storage backends we evaluate the performance for the well known MongoDB[3] DBMS under various workloads. In contrast to related work, our main focus is not on a performance comparison between containerized and virtualised execution. *(iv)* Based on the outcome of the evaluation, we propose open challenges for modelling and evaluating DBMS performance.

The remainder of this document is structured as follows: Sect. 2 discusses the containerization of stateful applications. Section 3 defines the evaluation methodology, while Section 4 presents the evaluation environment. Section 5 discusses the results and derives open evaluation challenges for containerized DBMS. Section 6 presents related work, and Sect. 7 concludes.

2 Challenges for Containerization of Stateful Applications

Containerization in the context of cloud computing is besides *hardware virtualization* for virtual machines (VMs) so called *operating-system (OS) virtualization* for containers. In hardware virtualization a hypervisor manages the resource allocation and operating state of virtual machines. OS-Virtualization uses operating system features to create lightweight isolated environments, known as containers. Container engines allocate resources and access to *e.g.* networking and storage, the popular Docker[4] engine. Orchestrators manage VMs or containers across hypervisors or container engines [1,3,15]. These virtualization approaches provide the different operational models depicted in Fig. 1(a) where each operation model combines the benefits and drawbacks of the respective virtualization approaches: Hardware virtualization securely isolates with fixed

[3] https://www.mongodb.com/de.
[4] https://www.docker.com.



(a) Operational Models (b) Storage Backends

Fig. 1. Containerization of stateful applications

hardware-oriented offers; OS virtualization provides less strong isolation with soft hardware limits [12].

In multi-tier applications, stateful components require to store data temporarily or even durably, *i.e.* by instantiating (distributed) DBMS or stateful caches. Figure 1(b) lists potential storage backends for VMs and containers, which leads to challenging decisions when deploying stateful containers, especially in large-scale set-ups. Challenges in this field include *(i)* performance aspects such as throughput and latency, *(ii)* support for scalability, *i.e.* considering parallel read/write access, and *(iii)* failure strategies and recovery mechanisms. Since VMs and containers isolate customers to share infrastructure, performance interferences may occur whenever resources (*e.g.* storage) are utilised, which are not directly under control of the container engine and the underlying kernel.

The focus of our work evaluates the performance aspect of containerized DBMS with respect to different storage backends for containerized DBMS on physical hardware over DBMS in VMs to containerized DBMS on top of VMs. The performance and runtime overhead of these approaches are evaluated in the following.

3 Evaluation Methodology

In this section, we define an extensible evaluation methodology for the identification of potential performance overhead of common operation models for containerized DBMS. In the following, the methodology is defined on a conceptual level, while Sect. 4 describes the technical implementation.

3.1 System Architecture

In order to provide a concise analysis of the potential performance overhead of containerized DBMS in the cloud, we define an extensible system architecture, which comprises three common operational models for DBMS, highlighted in the grey boxes in Fig. 1(a). Each operational model is defined by its virtualisation, *i.e.* OS virtualisation for container, hypervisor for VMs or container on top of VMs. Further, we apply two storage backends for the containerized DBMS, namely *local* for using the container filesystem and *host* for using the hosting resource filesystem as depicted in Fig. 1(b). For the VM-based DBMS, we apply the local filesystem provided by the hypervisor. The resulting resource configurations are depicted in Table 1. While *remote* storage is also a common storage configurations for containerized DBMS, it is omitted in this work to reduce the interference factor of the network and will be targeted in future evaluations. In addition, we do not use any container-specific network virtualisation as the focus relies on compute, memory and storage.

Table 1. Operational models and storage backends

ID	Operational model [*physical (P), container (C), VM*]	Storage backend [*local (L), host (H), remote (R)*]
P-C-L	Physical + container	Local
P-C-H	Physical + container	Host
VM-L	VM	local
VM-C-L	VM + container	Local
VM-C-H	VM + container	Host

3.2 Workload and DBMS

In favour of emulating container-centric workloads, we define a *write-heavy (w-h)* workload, emulating the storage of sensor data and a *read-heavy (r-h)* workload, emulating a social media application with mostly reads and barely update operations. Both workloads are defined in a *memory-bound* version, *i.e.* the whole data set fits into memory and a *disk-bound* version, *i.e.* the data is larger than the available memory. As workload generator, we select the Yahoo Cloud Serving Benchmark (YCSB) [4], which is widely used in performance studies on NoSQL DBMSs. YCSB offers web-based workloads based on create, read, update and delete (CRUD) operations, enabling the emulation of container-centric workloads [10].

As exemplary containerized DBMS, we select document-oriented MongoDB for our evaluation as it is a NoSQL DBMS[5]. MongoDB emphasizes its operation on virtualised resources[6]. Records are stored as *documents* within *collections*.

[5] https://db-engines.com/de/ranking.
[6] https://www.mongodb.com/containers-and-orchestration-explained.

While MongoDB supports a distributed architecture, we select a single node setup for our evaluation to reduce potential interference factors such as network jitter or MongoDB specific data distribution algorithms. Yet, our methodology can easily be extended for a distributed setup and also MongoDB can be exchanged with any desired DBMS.

3.3 Metrics

For each evaluation scenario the following metrics are collected to analyse the results: *throughput* in operations per seconds and *latency* per operation type in μs. Each evaluation scenario is repeated ten times to ensure significant results and for the all metrics the minimum, maximum, average and standard deviation are provided. In addition, system metrics (CPU, RAM, I/O, network) are monitored during each evaluation scenario for MongoDB and the YCSB to provide reliable results by ensuring that none of the system resources creates a bottleneck.

3.4 Evaluation Execution

Our methodology comprises the memory-bound (mb) and disk-bound (db) evaluation scenarios. Each scenario starts with the w-h workload, followed by the r-h workload. Each workload is executed against the resource combinations of operational models and storage backends presented in Table 1. Hence, the execution (E) of the memory-bound and disk-bound scenarios can be expressed as *scenario:E(mb(wh,r-h))*, *e.g. P-C-L:E(mb(wh,r-h))* and *P-C-L:E(db(w-h,r-h))*.

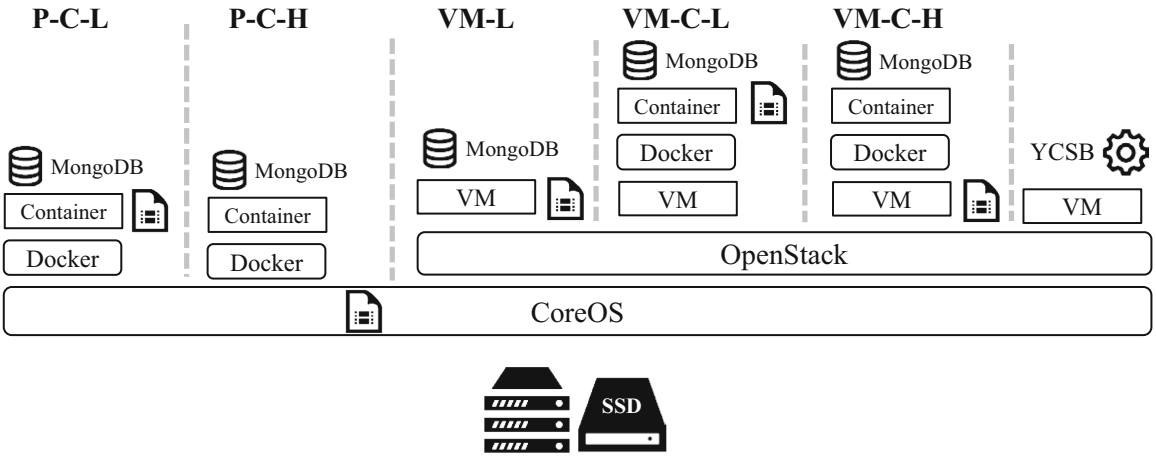


Fig. 2. Evaluation environment

4 Evaluation Environment

Based on the introduced evaluation methodology in Sect. 3, the following presents its implementation for a private, OpenStack-based cloud[7] (version Pike) with full and isolated access to all physical and virtual resources. In order to reduce potential resource interference and to guarantee reproducible results, we use the availability zones feature of OpenStack to dedicate one physical host for spawning the required VMs and containers. The resulting evaluation environment for the specified evaluation scenarios is depicted in Fig. 2. In the following, the implementation details for the resources, MongoDB and YCSB are presented.

Table 2. Evaluation scenario resources

Resource	Virtualisation	OS	Cores	RAM	FS	Storage
Physical host	-	CoreOS 1632	16[a]	64 GB	Ext4	512 GB[b]
MongoDB container	Docker 18.04	Ubuntu 16.04	4	4 GB	overlay2	40 GB
MongoDB VM	KVM, QEMU 1.5.3	Ubuntu 16.04	4	4 GB	Ext4	40 GB
YCSB VM	KVM, QEMU 1.5.3	Ubuntu 16.04	4	2 GB	Ext4	10 GB

[a] 2x Intel Xeon E5-2630 v3 8-Core Haswell 2.4 Ghz
[b] 2x 256 GB SSD of type SAMSUNG MZ7WD240HAFV-00003

4.1 Resources

As depicted in Fig. 2, all containers and VMs are located on the same physical host, which has enough resources for running the YCSB VM and the DBMSs without resource interference (*i.e.* no overbooking). Further, this set-up only uses the host-internal network interfaces and avoids the overhead of the OpenStack network service. Accordingly, all containers are configured to use the host network interface via `--network host`. The available resources of the respective physical host, container and VMs are described in Table 2. In order to ensure comparable results, the container resources on the physical host (*i.e.* P-C-L and P-C-H) are limited to 4 cores and 4 GB RAM. The containers on CoreOS use the kernel version 4.14.19-coreos while the VM and container inside the VMs use the kernel version 4.4.0-127-generic.

4.2 MongoDB and YCSB

The evaluation scenarios are based on a vanilla deployment of MongoDB and the YCSB to ensure a baseline performance evaluation of MongoDB containerization. The relevant configurations for MongoDB and the YCSB are listed in Table 3. Further, the YCSB operation distribution for the *w-h* workload are 100% write operations and for the *r-h* workload 95% read operations and 5% update operations. Table 3 also highlights overall collection size of each workload

[7] https://www.openstack.org/.

Table 3. YCSB VM details

MongoDB configuration	Value	YCSB configuration	Value
Version	3.6.3 (CE)	Version	0.12[a]
Services	1 × mongod	Record size	1 KB
Storage engine	WiredTiger	# of records (memory-bound)	2.000.000
Replication	off	# of records (disk-bound)	10.000.000
		# of operations	10.000.000
		# of threads	20
		Distribution	Zipfian

[a] https://github.com/brianfrankcooper/YCSB/releases/tag/0.12.0

version as the number of records for the *memory-bound* version results in a 2 GB MongoDB collection, while the *disk-bound* version results in a 10 GB MongoDB collection. The MongDB binding of YCSB is configured with the write concern option[8] `w = 1` and `j = false`, *i.e.* write operations are acknowledged by MongoDB after they are put into memory.

4.3 Portability and Reproducibility

The execution of each scenario is fully automated by utilizing ready to deploy artifacts, which are together with the results publicly available[9]; their release as open research data is currently under way. For the Docker images we make use of the Docker native capabilities of building images based on Dockerfiles. The VM images are generated by Packer[10]. Packer processes a Packerfile, which is similar to a Dockerfile, but uses a multitude of different virtualization providers to generate and store the image. In our case we are using OpenStack Glance[11]. This approach enables fellow researchers to reproduce, validate and extend our scenarios by changing the cloud provider or benchmark a different DBMS.

5 Results and Discussion

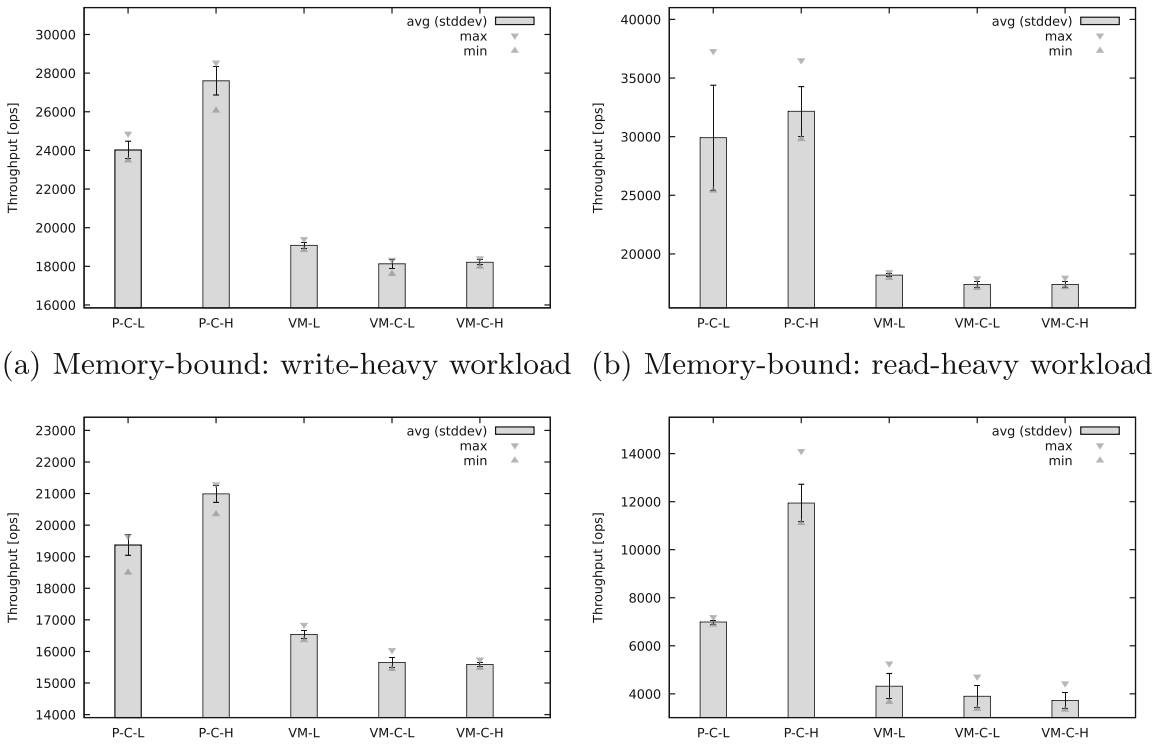
In the following, we present and discuss the results of the memory- and disk-bound evaluation scenarios (*cf.* 1) based on defined metrics in Sect. 3.3.

5.1 Evaluation Results

The throughput results are depicted in Fig. 3 and latency results in the Fig. 4. Each plot represents the results of the respective scenario, *i.e.* memory-bound

[8] https://docs.mongodb.com/manual/reference/write-concern/.
[9] https://github.com/omi-uulm/Containerized-DBMS-Evaluation.
[10] https://packer.io.
[11] https://docs.openstack.org/glance/pike/.

or disk-bound and the respective workload, *i.e.* w-h or r-h. For the latency plots of the r-h workloads, the first bar of each operational model always represents the read latency while the second bar represents the update latency. As remark, the results reflect the best case operational models as the DBMS and the YCSB are operated on the same, isolated physical host (*cf.* Sect. 4.1).



(a) Memory-bound: write-heavy workload (b) Memory-bound: read-heavy workload

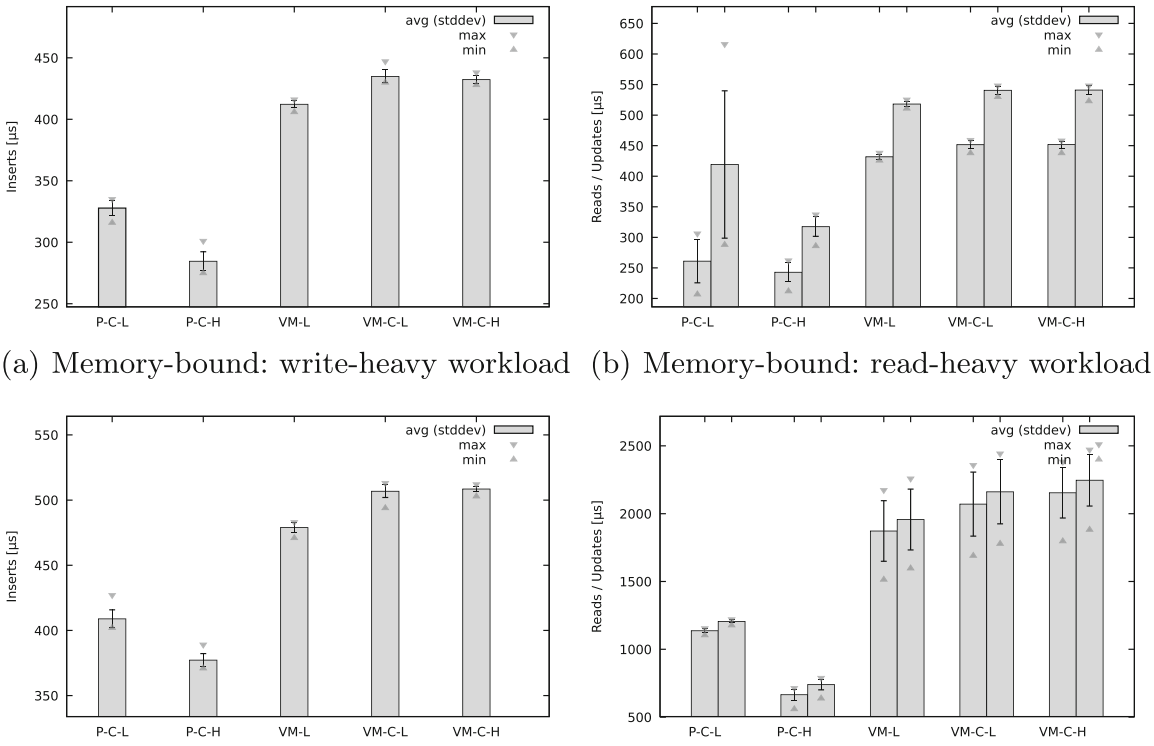
(c) Disk-bound: write-heavy workload (d) Disk-bound: read-heavy workload

Fig. 3. Throughput results

The results shows a significant throughput and latency overhead for operating a DBMS on top of VMs instead of using physical hardware. These results confirm previous performance studies of memory-bound workloads for former Docker versions [5]. A novel insight is shown by the results for the DBMS operated in a container on VM the (VM-C-L,VM-C-H) as the performance only decreases slightly compared to DBMS directly operated on VMs (VM-L), *e.g.* VM-C-H achieves 6% less throughput than VM-L for the w-h workload and 13% for r-h of the disk-bound scenario. Hence, if VMs are the only available resource, operating the DBMS in container on top of the VMs can be beneficial to exploit container orchestrators or the soft resource limits to operate additional containerized applications next to the DBMS on the same VM [12].

The second insight of the results is the performance overhead of the internal Overlay2 filesystem of Docker. The container on physical hardware with the Overlay2 filesystem (P-C-L) shows significantly less throughput and higher latencies compared to the container using the host filesystem (P-C-H). This finding most clearly applies for the r-h workload of the disk-bound scenario

(*cf.* Figs. 3(d) and 4(d)). The Overlay2 overhead is also present on container running on VMs (VM-C-H) but to a lower extent.



(a) Memory-bound: write-heavy workload (b) Memory-bound: read-heavy workload

(c) Disk-bound: write-heavy workload (d) Disk-bound: read-heavy workload

Fig. 4. Latency results

5.2 Open Evaluation Challenges

The results of our baseline evaluation, show that containers are suitable to operate DBMS, even on top of VMs. Yet, the operational models reveal significant performance deviations, also dependent on the memory- or disk-bound scenarios. Hence, the selection of the operational model in conjunction with the storage backend is a crucial decision for the DBMS operator, which has to be driven by the available operational models, the targeted performance and the demand of optional orchestration features.

Based on our methodology and the presented baseline results, we derive a set of open evaluation challenges, which have to be addressed to drive the selection process of the operational model for containerized DBMS: *(i)* The performance of the presented operational models needs to be evaluated based on public cloud offerings by considering additional hypervisors and containers with respect to memory- and disk-bound DBMS workloads. *(ii)* The presented storage backends require a dedicated evaluation with respect to different local and remote container storage drivers. This also comprises local and remote block storage of the host resource. *(iii)* As the DBMS performance deviation of the operational models VM-L, VM-C-L, and VM-C-H are in a tolerable margin, the advantages

of VM-C-L and VM-C-H have to be analysed with respect to orchestration and the co-location with suitable applications. *(iv)* The presented methodology needs to be extended for additional DBMS to evaluate their containerization feasibility and container orchestration features with respect to the scalability and elasticity of distributed DBMS [11]. Further, container orchestration features for high availability and migration of containerized DBMS need to be evaluated in the context of the presented operational models and storage backends.

6 Related Work

With the increasing usage of containers besides VMs in cloud offerings, different comparative analysis of their performance overhead and resource isolation capabilities have been conducted. Moreover, the containerization of DBMS moved into the focus, especially in combination with container orchestrators.

6.1 Performance Overhead and Resource Isolation

The performance overhead of VMs in contrast to Docker containers running on physical hardware is evaluated by [5]. SysBench[12] is used to compare the throughput of MySQL running on VMs against containerized MySQL. The results show that VMs cause a higher performance overhead as Docker containers for disk-intensive workloads. Further, the usage of the Docker AUFS storage driver causes a higher performance overhead as the usage of Docker volumes.

A related performance comparison of KVM VMs, Docker and LXC containers and a lightweight VM approach based on OSv[13] is provided by [7]. The evaluation is based on different resource-specific micro-benchmarks and the results accord to [5] for the lower performance of VMs for disk-intensive workloads.

An analysis and evaluation of the Docker storage drivers with respect to filesystem performance is presented by [13]. The results demonstrate that the choice of the storage driver can influence the filesystem performance significantly where the Btrfs storage driver achieves the best performance but less stability as the other storage drivers.

The comparative analysis of the resource isolation capabilities of VMs and containers is provided by [8,12,16]. While [12] apply resource-specific micro-benchmarks, [8,16] use DBMS and respective DBMS workloads to evaluate the resource capabilities. All of these evaluation indicate a stronger resource isolation of VMs, especially for disk-bound workloads.

While existing performance evaluations focus either on micro-benchmarks or apply DBMS only for the evaluation of the resource isolation, our evaluation provides an evaluation across multiple operational model and storage backends. In addition, the operation of container on top of VMs is emphasized by [12] but so far no performance evaluation has considered this operational model.

[12] https://github.com/akopytov/sysbench.
[13] http://osv.io/.

6.2 Containerization of DBMS

The containerization of DBMS in combination with container orchestrators provides multiple adaptation actions to automate the operation of NoSQL DBMSs [2]. Hereby, the orchestration features of Kubernetes are enhanced with distributed DBMS-specific adaptation rules for proactive and low-cost adaptations to avoid the transfer of data between nodes. While container orchestrators typically manage the disk storage internally, modifying the persistent storage within container orchestrators is difficult and hinders their adoption for DBMS [6]. Therefore, [6] present a persistent storage abstraction layer for container orchestrators, which eases the usage of containerized DBMS across different container orchestrators. Yet, the usage of container orchestrators and their internal handling of persistent storage can introduce additional performance overhead for DBMS. Hence, [14] analyse the performance overhead of using remote storage for containerized DBMS within Kubernetes.

While the usage of containerized DBMS with container orchestrators eases the automation of DBMS operation, there is potential performance overhead added by the different handling of the persistent storage of the container orchestrator. While, [14] provide a fist step into analysing this overhead for remote storage, we provide a baseline evaluation for container local and host storage backends. In addition, we apply a memory- and disk-bound workload to identify the suitability of container for the respective DBMS workloads.

7 Conclusion and Future Work

The evolvement of container leads to a variety of new operational models for distributed applications in the cloud. While containers work fine for stateless applications, stateful applications such as database management systems (DBMS) are receiving increasing attention recently. As DBMS add the persistence aspect to the operational model, storage backends for containers are evolving. Yet, the performance impact of these new operational and storage backends remains unclear for containerized DBMS. Hence, we analyse current operational and storage backends in the context of containerized DBMS. and derive a baseline evaluation methodology for a comparative evaluation of operational models and storage backends. Hereby, we define a memory- and a disk-bound scenario, which is applied on three operational models (container on physical hardware, virtual machines (VMs) and container on VMs) in combination with two storage backends (container filesystem and host filesystem), resulting in 20 evaluation configurations. The evaluation is executed in a private OpenStack with a containerized MongoDB. The results show a significant performance overhead of container running on VMs in contrast to container running on physical hardware. Yet, running container on VMs with different storage backends only causes a tolerable performance impact in contrast to running the DBMS directly on the VM. Further, the usage of the container internal filesystem causes a significant performance overhead compared to using the host filesystem.

Based on these baseline results, we conduct that container are suitable to operate DBMS but additional evaluations are required to get a clear understanding of potential performance bottlenecks. Therefore, we derive a set of open evaluation challenges, which will be addressed in future work: *(i)* evaluating additional operational models implementations; *(ii)* evaluating additional storage backends; *(iii)* consolidation of containerized DBMS and processing applications on top of VMs and *(iv)* the feasibility of DBMS containerization and orchestration for different DBMS. These challenges will be addressed within [9].

Acknowledgements. The research leading to these results has received funding from the EC's Framework Programme HORIZON 2020 under grant agreement number 731664 (MELODIC) and 732667 (RECAP).

References

1. Baur, D., Seybold, D., Griesinger, F., Tsitsipas, A., Hauser, C.B., Domaschka, J.: Cloud orchestration features: are tools fit for purpose? In: UCC, pp. 95–101. IEEE (2015)
2. Bekas, E., Magoutis, K.: Cross-layer management of a containerized NoSQL data store. In: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 1213–1221. IEEE (2017)
3. Burns, B., Grant, B., Oppenheimer, D., Brewer, E., Wilkes, J.: Borg, omega, and kubernetes. Queue **14**(1), 10 (2016)
4. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking cloud serving systems with YCSB. In: ACM Symposium on Cloud computing, pp. 143–154. ACM (2010)
5. Felter, W., Ferreira, A., Rajamony, R., Rubio, J.: An updated performance comparison of virtual machines and linux containers. In: ISPASS, pp. 171–172. IEEE (2015)
6. Mohamed, M., Warke, A., Hildebrand, D., Engel, R., Ludwig, H., Mandagere, N.: Ubiquity: extensible persistence as a service for heterogeneous container-based frameworks. In: Panetto, H., et al. (eds.) OTM 2017. Lecture Notes in Computer Science, pp. 716–731. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69462-7_45
7. Morabito, R., Kjällman, J., Komu, M.: Hypervisors vs. lightweight virtualization: a performance comparison. In: IC2E, pp. 386–393. IEEE (2015)
8. Rehman, K.T., Folkerts, E.: Performance of containerized database management systems. In: DBTEST, p. 6. ACM (2018)
9. Seybold, D.: Towards a framework for orchestrated distributed database evaluation in the cloud. In: Proceedings of the 18th Doctoral Symposium of the 18th International Middleware Conference, pp. 13–14. ACM (2017)
10. Seybold, D., Domaschka, J.: Is distributed database evaluation cloud-ready? In: Kirikova, M., et al. (eds.) ADBIS 2017. CCIS, vol. 767, pp. 100–108. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67162-8_12
11. Seybold, D., Wagner, N., Erb, B., Domaschka, J.: Is elasticity of scalable databases a myth? In: IEEE Big Data, pp. 2827–2836. IEEE (2016)
12. Sharma, P., Chaufournier, L., Shenoy, P., Tay, Y.: Containers and virtual machines at scale: a comparative study. In: Proceedings of the 17th International Middleware Conference, p. 1. ACM (2016)

13. Tarasov, V., et al.: In search of the ideal storage configuration for docker containers. In: FAS* W, pp. 199–206. IEEE (2017)
14. Truyen, E., Reniers, V., Van, D., Landuyt, B.L., Joosen, W., Bruzek, M.: Evaluation of container orchestration systems with respect to auto-recovery of databases (2017)
15. Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., Wilkes, J.: Large-scale cluster management at google with borg. In: Proceedings of the Tenth European Conference on Computer Systems, p. 18. ACM (2015)
16. Xavier, M.G., De Oliveira, I.C., Rossi, F.D., Dos Passos, R.D., Matteussi, K.J., De Rose, C.A.: A performance isolation analysis of disk-intensive workloads on container-based clouds. In: PDP, pp. 253–260. IEEE (2015)

Chapter 18

[core11] Towards Understanding the Performance of Distributed DatabaseManagement Systems in Volatile Environments

This article is published as follows:

Jörg Domaschka and Daniel Seybold. ”Towards Understanding the Performance of Distributed Database-Management Systems in Volatile Environments”, *Symposium on Software Performance*, published 2019, Gesellschaft für Informatik, URL: https://pi.informatik.uni-siegen.de/stt/39_4/01_Fachgruppenberichte/SSP2019/SSP2019_Domaschka.pdf

Towards Understanding the Performance of Distributed Database Management Systems in Volatile Environments

Jörg Domaschka
joerg.domaschka@uni-ulm.de
Ulm University, Ulm, Germany

Daniel Seybold
daniel.seybold@uni-ulm.de
Ulm University, Ulm, Germany

Abstract

Cloud computing provides scalability and elasticity mechanisms on resource level and has become the preferred operational model for many applications. These, in turn, are based on distributed architectures trusting that this leads to scalability and elasticity and hence, good performance.

Many applications rely on one or multiple database management systems (DBMS) as storage backends in order to manage their persistent state. Hence, the selection of a DBMS for a specific use case is crucial for performance and other non-functional properties. Yet, the choice is cumbersome due to the large number of available systems and the many impact factors ranging from the size of virtual resources, the type of the DBMS, and its architecture and scaling factor.

In this paper, we summarise our experiences with performance evaluation for cloud-hosted DBMS in order to find well-suited configurations for specific use cases. We demonstrate that the overall performance of a distributed DBMS depends on three major domains (workload, cloud environment, and DBMS) with various parameters for each dimension.

1 Introduction

Database Management Systems (DBMS) are a major building block of today's applications. They sit at the heart of any Web-business application, of many types of IoT applications, including geographically spread-out installations such as Smart Cities and Industry 4.0 sites; they are the backbone of serverless application.

Currently there are more than 200 distributed DBMS available as commercial and open source products that all claim to be reliable, scalable, and elastic while providing superior performance [7]. Yet, analyses of existing systems show that they differ a lot and suggest that the DBMSs need to be carefully selected based on the use case characteristics [3]. Only then can the DBMS deliver sufficient throughput and latency while at the same time satisfying potential non-functional requirements such as availability.

Cloud computing, virtualisation, and containerization offer appealing approaches to host DBMS in particular as they offer a natural way to quickly provision compute, storage and networking resources, and hence, realise the technical underpinning for dynamic scaling and adaptations. On the downside, relying on such resources introduces volatility in service quality as many critical aspects of the infrastructure are no longer in the hands of the DBMS operator. Further, there is an overwhelming selection of different cloud offerings with different impact on the performance of DBMS's service quality.

Thus, the decision for a cloud-hosted DBMS requires an understanding of the DBMS domain—which DBMS provides which features and what feature has which impact on performance; the cloud domain—which cloud configuration has which impact on performance; and finally the workload domain—what is the read/write/update ratio facing the DBMS and what consistency and reliability demands exist.

In this paper, we summarise the key insights gained over a series of papers [4, 6, 8] investigating the impact of the overall problem over the DBMS and the cloud domain. Due to limitations of space, we leave out the workload domain, as this is not something that can be influenced by an operator. Understanding the workload is hence a prerequisite for the selection process and benchmarking done in that process [5].

2 DBMS Impact Factors

The performance of non-distributed applications is mostly determined by the capabilities of the particular hardware (CPU, memory, I/O) running the application. For distributed applications the communication network plays a role as well as communication protocols causing e.g. queuing, drops, latency, and waiting time. In contrast to stateless applications, stateful applications may need to coordinate the state of their component instances causing traffic and load that are only mediately rooted in external workload, but are a consequence of internal processes such as garbage collection and consistency protocols.

In distributed DBMS (DDBMS), multiple instances of a DBMS provide the client with the impression of a logical DBMS. DDBMS provide two basic mechanisms to ensure scalability, availability, and reliability: sharding and replication. Hence, when designing a cloud-hosted DDBMS, the operator needs to size the individual instances (memory, #cores, type

and amount of storage), and decide on the cluster size (cf. the number of shards), the replication degree (cf. reliability and availability), and read and write consistency (cf. programmability and availability) [3, 8]. These decisions result in the following impact factors for operating DDBMS.

2.1 Sharding and Scale

The use of sharding distributes the data set of the logical DDBMS over the available DBMS instances. Consequently, sharding increases the overall capacity of a DDBMS when more instances are added. Sharding increases the overall throughput in cases where the workload is uniformly distributed over the shards.

2.2 Consistency and Replication

A consistency model defines in which order operations issued from multiple users to different data tuples may interleave. Hence, the consistency model defines which changes to a tuple are visible to a user.

Replication creates multiple copies of a single data tuple. This protects the tuple against data loss in case the DBMS instance hosting the shard containing that tuple fails. Hence, replication increases reliability of both the DDBMS and individual tuples. Depending on consistency requirements, the use of replication may either increase or decrease throughput. In case of weak consistency, operations may be targeted to different replicas of a tuple. For strong consistency more than one replica needs to be read/written.

The use of replicated tuples introduces the need to keep the tuples in sync with each other. According to the CAP theorem, availability and consistency are mutually dependant in any distributed and stateful system [1]. This leaves a continuum of possible trade-offs which has caused an increasing heterogeneity in the DDBMS landscape [7]. In consequence, many DDBMS have introduced specific configuration options to optimize for specific use cases. Often, this makes consistency and availability guarantees incomparable and evaluations even more challenging.

2.3 Resource Sizing

When operating a DDBMS, there is a trade-off between using multiple compute resources (e.g. virtual machines) or fewer larger ones. While the overall performance increases the fewer and the larger the individual resources are, the more vulnerable the system gets for failures. Also, replacing failed instances takes longer when larger portions of the overall state fail and need to be synced. Besides, these general considerations, the number of compute cores influences the parallelism in processing client requests, while the amount of memory influences the ratio of the data set that can reside in memory.

For storage, considerations are even more challenging due to different types of available cloud offerings [2]. In IaaS clouds usually three options exist: (i) use a virtual machine with large storage, (ii) attach volumes to the virtual machine as block devices, (iii) include (mount) a remote file system into the virtual machine. Users of cloud services may further build custom storage hierarchies from these basic services. For instance, they may run their own volume-based distributed file system within their virtual machines and mount this into other virtual machines.

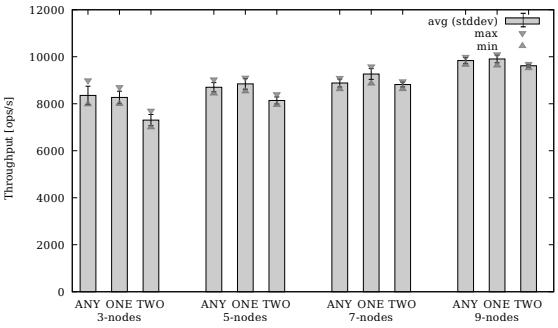


Figure 1: Impact of DBMS Domain: Cassandra [8]

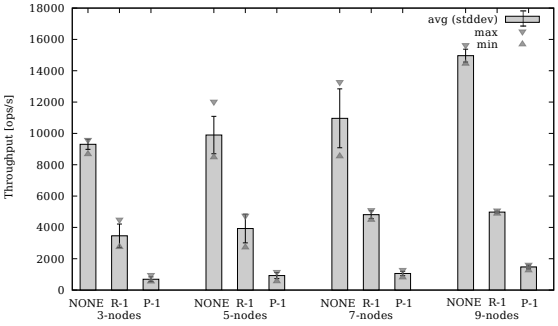


Figure 2: Impact of DBMS Domain: Couchbase [8]

2.4 Examples

Figures 1 and 2 show the average throughput over five experiments (including the standard deviation) of three different write consistency settings for Apache Cassandra (CA) and Couchbase (CB). While their consistency mechanisms can not be mapped 1:1, the applied write consistency increases from the first to the third setting. Moreover, each plot comprises the throughput of different cluster sizes, showing the scalability of the respective DBMS under a fixed workload. The results clearly show that a stronger write consistency has a significant throughput impact for while for CA the impact is neglectable. More details and results are available in [8].

3 Cloud Impact Factors

The performance of an application running on a cloud infrastructure heavily depends on many cloud provider design decisions, but also on operator design decision as discussed in the following [6, 8].

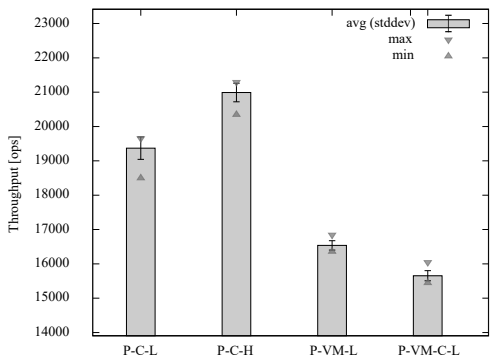


Figure 3: Impact of Cloud Resource Types [6]

3.1 Infrastructure

The maximum overall performance of a cloud-hosted application depends on the physical hardware capabilities of the cloud infrastructure including the type of physical CPUs and storage as well as networking. Further, it depends on configuration choices the cloud provider has made such as the network topology; the physical set-up of storage: SSDs vs. HDDs, hierarchical RAID systems, distributed file systems, hyperconverged infrastructure; etc.

Even though these aspects are mostly differentiating factors between cloud providers, customers can influence these decisions to some extent: for instance, they can select specific virtual machine types, regions, and availability zones that are known to be backed by a certain type of physical hardware. Yet, even if they are known to the clients, their impact on performance is hard to estimate and hence mostly unclear.

3.2 Resource Types

The resource type captures the abstraction a cloud provider offers to its customers. It defines whether the resource provided is a bare metal server, a virtual machine, a (Docker) container, or at an even higher layer. This decision has a significant impact on performance. The type of resource is often an immediate consequence of choosing a particular cloud provider.

3.3 Examples

Figure 3 shows the performance of heterogenous cloud resources based on the avg. write throughput (with stddev.) over ten experiments of a single MongoDB (MDB) instance deployed on the cloud resource types: *(P-C-L)* physical server (P) with a MDB container (C) using the local Overlay2 container filesystem (L); *P-C-H* using the host filesystem (H) of the physical server; *P-VM-L* running MDB in a VM using the VM filesystem; *P-VM-C-L* running MDB in a container on top of a VM using the local Overlay2 container filesystem. The results point out the impact of using Overlay2 with containers or VMs to operate DBMS. More results and conclusions are available in [6].

4 Conclusion

This paper summarises the design space when running distributed database management systems (DDBMS) in (volatile) cloud environments. Using examples, we illustrated that the overall performance (measured by throughput) of DDBMS are dependent on the services offered by cloud providers and the storage types used.Even minor changes in the set-up of experiments may have larger impact on performance. Besides these external factors, there are huge differences between the DDBMS themselves and the choice of a DDBMS needs to be be well considered and measured against the workload and application requirements.

Ongoing and future work extends the results shown here with the consideration of non-functional requirements. Here, we focus on the evaluation of both scalability/elasticity and availability guarantees of different DDBMS under advanced workloads. Finally, we are currently investigating the impact of overbooking and noisy neighbours on the performance of DDBMS.

References

[1] E. Brewer. “CAP twelve years later: How the ”rules” have changed”. In: *Computer* 2 (2012), pp. 23–29.

[2] S. Kächele et al. “Beyond IaaS and PaaS: An Extended Cloud Taxonomy for Computation, Storage and Networking”. In: *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*. Dec. 2013, pp. 75–82.

[3] J. Domaschka, C. B. Hauser, and B. Erb. “Reliability and Availability Properties of Distributed Database Systems”. In: *2014 IEEE 18th International Enterprise Distributed Object Computing Conference*. Sept. 2014, pp. 226–233.

[4] D. Seybold et al. “Is elasticity of scalable databases a myth?” In: *2016 IEEE International Conference on Big Data (Big Data)*. IEEE. 2016, pp. 2827–2836.

[5] D. Seybold and J. Domaschka. “Is Distributed Database Evaluation Cloud-Ready?” In: *European Conference on Advances in Databases and Information Systems*. Springer. 2017, pp. 100–108.

[6] D. Seybold et al. “The Impact of the Storage Tier: A Baseline Performance Analysis of Containerized DBMS”. In: *European Conference on Parallel Processing*. Springer. 2018, pp. 93–105.

[7] S. Mazumdar et al. “A survey on data storage and placement methodologies for Cloud-Big Data ecosystem”. In: *Journal of Big Data* 6.1 (2019), p. 15.

[8] D. Seybold et al. “Mowgli: Finding your way in the DBMS jungle”. In: *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*. ACM. 2019, pp. 321–332.