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,

2330-2186/19/$31.00 ©2019 IEEE
DOI 10.1109/CloudCom.2019.00020

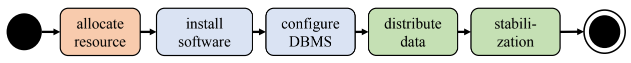

Fig. 1: Steps of an elastic scale-out

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

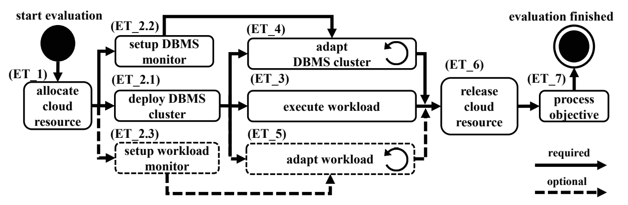

Fig. 2: Elasticity Evaluation Process

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)

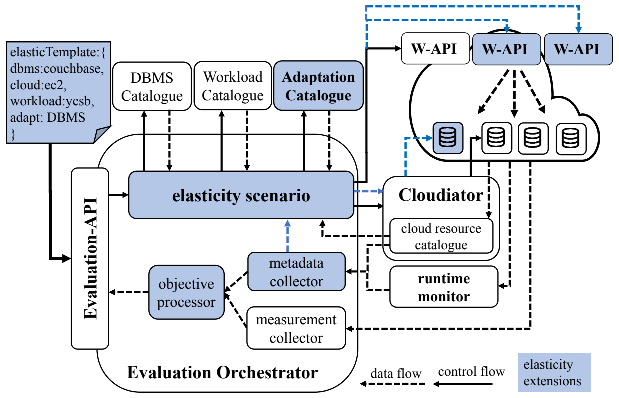

Fig. 3: Mowgli architecture with Kaa elasticity extension

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	$\{1 \ldots n\}$ $seconds$	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.

[3]https://omi-gitlab.e-technik.uni-ulm.de/mowgli/getting-started

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	$\{1 \ldots n\}$
trigger type	$\{time, progress, WSR\}$
time trigger	$\{1 \ldots n\}$ $minutes$
progress trigger	$\{0 \ldots 1.0\}$ $\%$ 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.

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 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.

[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	

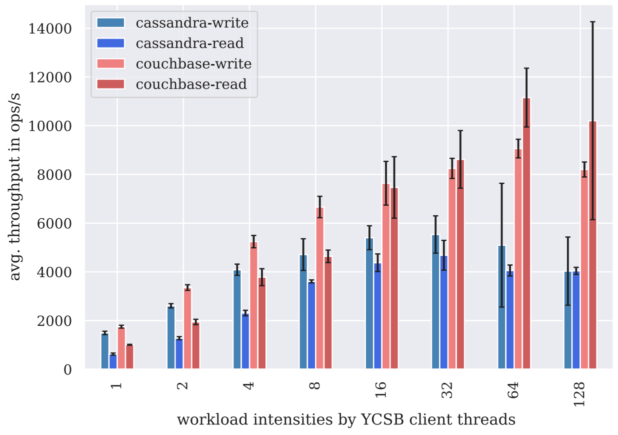

Fig. 4: Calibration Results — 3 Node Cluster

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-

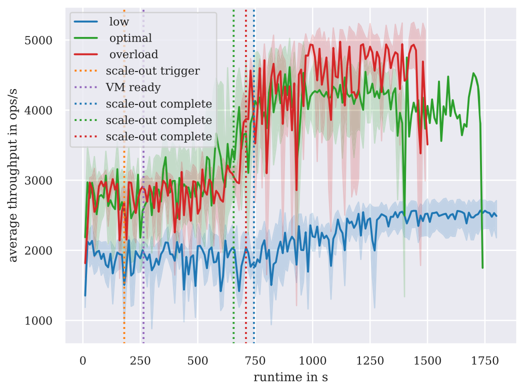

Fig. 5: Cassandra—elastic scale-out, read-heavy workload

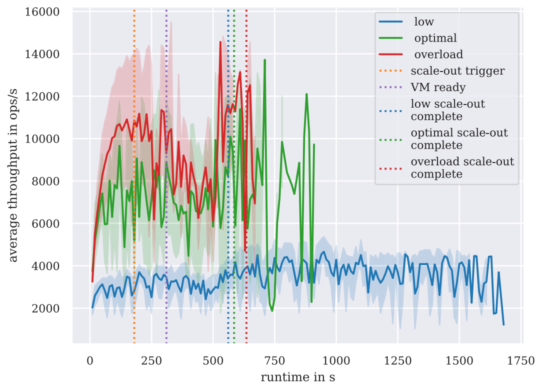

Fig. 6: Couchbase—elastic scale-out, read-heavy workload

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

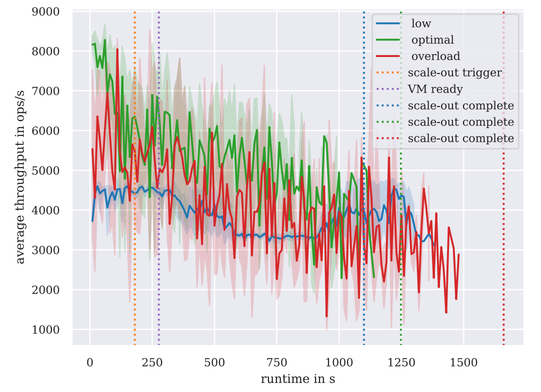

Fig. 7: Cassandra—elastic scale-out, write-heavy workload

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

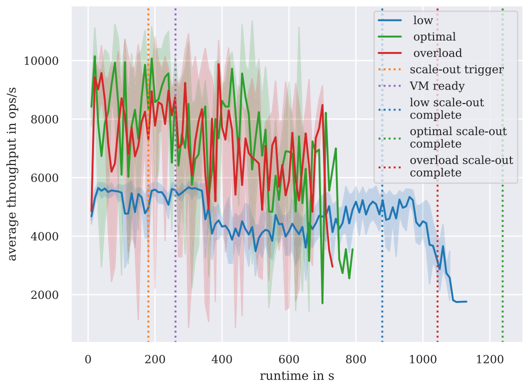

Fig. 8: Couchbase—elastic scale-out, write-heavy workload

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.