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].

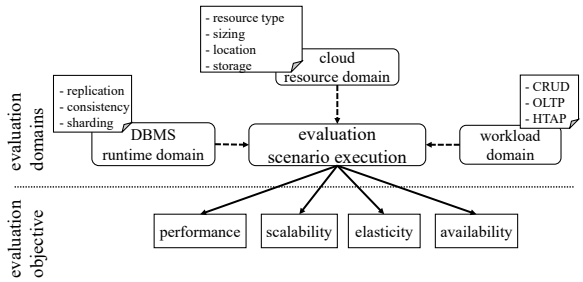

Figure 1: DBMS Evaluation Domains

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

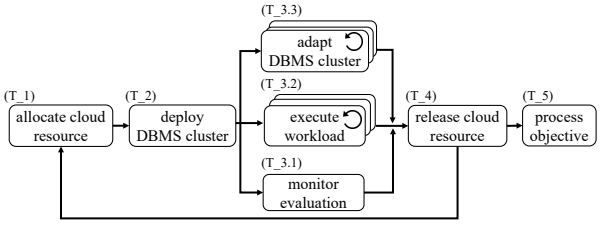

Figure 2: DBMS Evaluation Process

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 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.

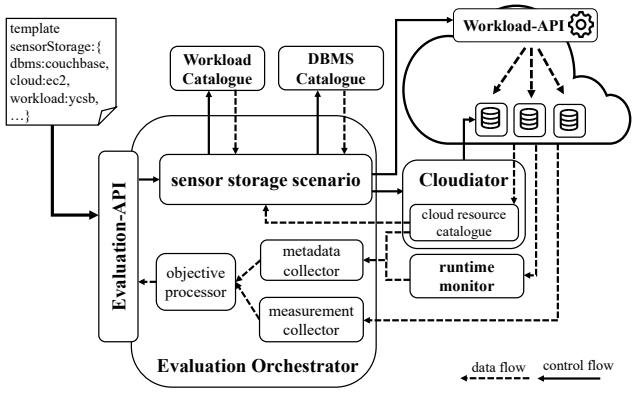

Figure 3: Mowgli architecture

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

[2]https://omi-gitlab.e-technik.uni-ulm.de/mowgli

[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

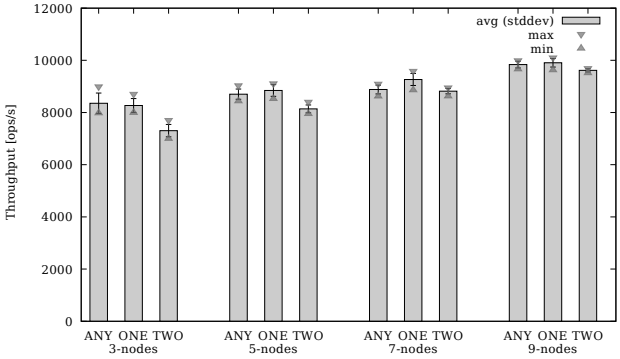

Figure 4: Q1 - Cassandra - OpenStack SSD Storage

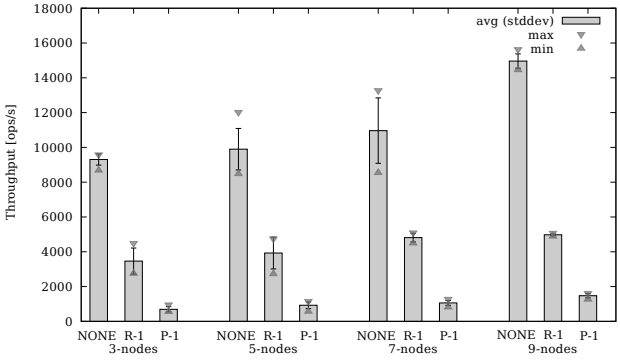

Figure 5: Q1 - Couchbase - OpenStack SSD Storage

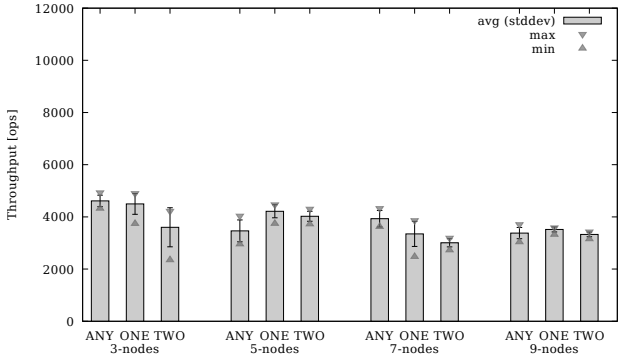

Figure 6: Q1 - Cassandra - OpenStack Remote Storage

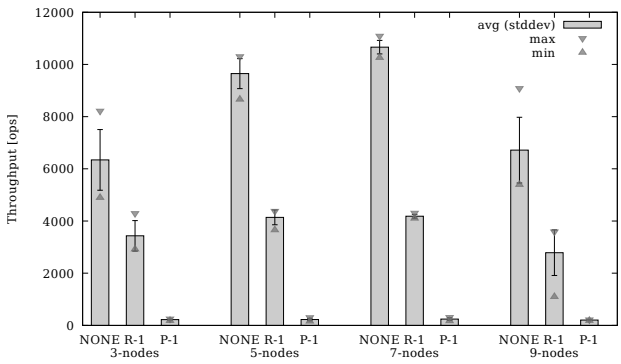

Figure 7: Q1 - Couchbase - OpenStack Remote Storage

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 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%

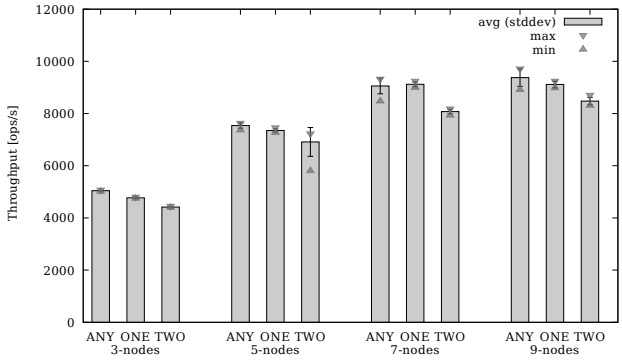

Figure 8: Q1 - Cassandra - EC2 Remote Storage

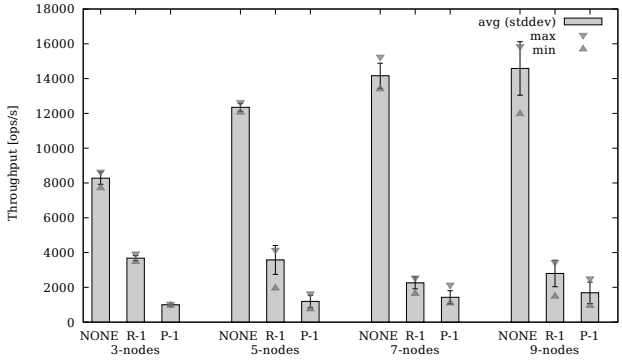

Figure 9: Q1 - Couchbase - EC2 Remote Storage

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.

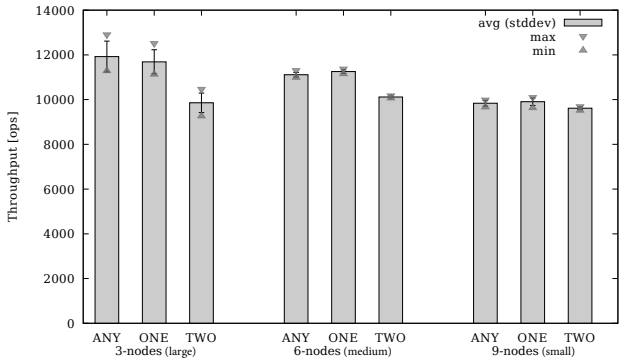

Figure 10: Q2 - Cassandra - SSD Storage

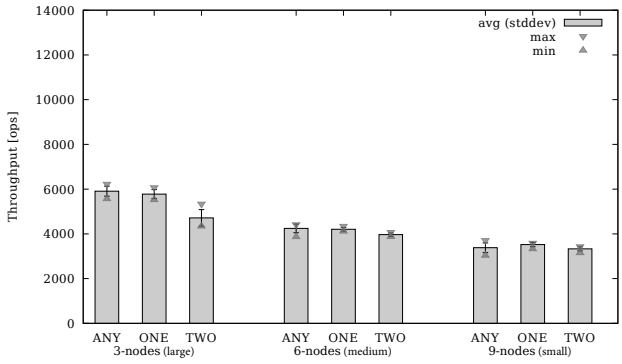

Figure 11: Q2 - Cassandra - Remote Storage

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.