Towards an Extensible Middleware for Database Benchmarking

David Bermbach[1], Jörn Kuhlenkamp[1], Akon Dey[2(✉)], Sherif Sakr[3], and Raghunath Nambiar[4]

[1] Information Systems Engineering Group, TU Berlin, Berlin, Germany
{david.bermbach,j.kuhlenkamp}@tu-berlin.de
[2] School of Information Technologies, University of Sydney, Sydney, Australia
akon.dey@sydney.edu.au
[3] King Saud Bin Abdulaziz University for Health Sciences, Riyadh, Saudi Arabia
sakrs@ksau-hs.edu.sa
[4] Cisco Systems, Inc., San Jose, USA
rnambiar@cisco.com

Abstract. Today's database benchmarks are designed to evaluate a particular type of database. Furthermore, popular benchmarks, like those from TPC, come without a ready-to-use implementation requiring database benchmark users to implement the benchmarking tool from scratch. The result of this is that there is no single framework that can be used to compare arbitrary database systems. The primary reason for this, among others, being the complexity of designing and implementing distributed benchmarking tools.

In this paper, we describe our vision of a middleware for database benchmarking which eliminates the complexity and difficulty of designing and running arbitrary benchmarks: workload specification and interface mappers for the system under test should be nothing but configuration properties of the middleware. We also sketch out an architecture for this benchmarking middleware and describe the main components and their requirements.

1 Introduction

Relational database management systems (RDBMS) have been around since the 1960 s and have long been considered to be a one-size-fits-all solution to data persistence. However, over the last few years, a plethora of new data storage solutions – typically referred to as NoSQL (Not Only SQL) systems – have been developed to step in where RDBMS have previously been unable to fulfill certain complex application requirements, e.g., elastic scalability. Today's data storage systems are primarily categorized by their supported data model and their application data access interface into *column stores* (e.g., Bigtable [14] or Cassandra[31]), *key-value stores* (e.g., Dynamo [19] or Voldemort[1]), *document stores* (e.g., MongoDB[2] or

[1] project-voldemort.com.
[2] mongodb.org.

R. Nambiar and M. Poess (Eds.): TPCTC 2014, LNCS 8904, pp. 82–96, 2015.
DOI: 10.1007/978-3-319-15350-6_6

CouchDB[3]), and *RDBMS* (e.g., MySQL[4] or PostgreSQL[5])[13,44]. In addition to these, there are other database systems targeting special use cases, e.g., caching storage for objects (e.g., Memcached[6] or Redis[7]) and graph-oriented data (e.g., Neo4j[8]), and the so-called NewSQL[9] databases (e.g., VoltDB[10] or NuoDB[11]). In essence, there are hundreds of different database systems available today and the number is increasing everyday.

Choosing a single database system from this large set of available database systems for a concrete use case is a non-trivial task [41]. From an application developer's perspective, there are certain functional requirements for a database system based on the application's needs. From the subset of database systems fulfilling the demanded functional requirements, an application developer typically wants to select the "best" database system based on non-functional quality attributes like performance, availability, data consistency, security, cost, etc. This shows the clear need for an ability to compare different database systems in terms of their non-functional quality characteristics which is usually addressed by benchmarking.

For such a benchmark to measure these quality attributes in a meaningful way, it requires running a workload that is comparable to the workload which will eventually be generated by the application. Therefore, measurement results obtained while running different workloads have little meaning. Most benchmarks available today can either be used with a subset of database systems (e.g., TPC[12] benchmarks for RDBMS) or do not use realistic, application-driven workloads (e.g., YCSB [15] or YCSB++ [36]). This prevents a fair comparison of database systems from different categories, e.g., a column store and an RDBMS.

Furthermore, such database benchmarks should also be easy to use, i.e., it should consist of both a measurement method and a ready-to-use toolkit. Again, existing benchmarks fall short by either not including a toolkit (e.g., see the TPC Express initiative [35]) or implementing the toolkit based on design decisions which limit the toolkit's applicability to only a subset of existing databases (e.g., YCSB [15]).

We argue that a middleware for the execution of arbitrary database benchmarks is missing. When designing a benchmark, the benchmark designer should concentrate on his core competences – namely, specifying a realistic, application-driven workload profile and means to analyze obtained measurements. Instead of having the middleware take care of the hassle of distributed benchmarking

[3] couchdb.apache.org.
[4] mysql.com.
[5] postgresql.org.
[6] memcached.org.
[7] redis.io.
[8] neo4j.org.
[9] NewSQL is a term used to refer to a new generation of RDBMS that attempt to provide the same scalable performance of NoSQL systems for OLTP applications while maintaining the full ACID guarantees provided by traditional RDBMS.
[10] voltdb.com.
[11] nuodb.com.
[12] tpc.org.

and managing the measurement infrastructure, today, a benchmark designer has to implement this from scratch for every single benchmark. The result of this is obvious; we end up with either application-driven benchmarks without a toolkit or benchmarking toolkits with flawed implementations and limited features. We are of the opinion that workload specifications and mappers should be treated as configuration parameters only.

In this vision paper, we present the first steps towards a middleware for the execution of arbitrary database benchmarks which:

- should offer an execution environment for diverse workloads,
- should not make any assumptions regarding the underlying architecture and implementation of the database under test, and
- should incorporate the measurement of performance as well as of more complex quality of service (QoS) levels.

For this purpose, we first identify the general requirements for benchmarks in Sect. 2 and describe why these requirements are difficult to maintain without a benchmarking middleware. Based on this, we discuss the requirements such a middleware should, hence, fulfill in Sect. 3. Finally, we sketch out the high-level architecture we envision for this middleware (Sect. 4) and discuss related work (Sect. 5) before coming to a conclusion in Sect. 6.

Please, note that some of the requirements from Sects. 2 and 3 may overlap as requirements related to the execution of benchmarks will inevitably also be requirements for a middleware solution targeting the execution of benchmarks. Still, the focus might differ depending on the section.

2 Requirements for Database Benchmarks

The process of benchmarking database systems is typically repetitive, time-consuming and tiresome [42]. To exacerbate the situation, without a good benchmark, the results of the benchmarking process can be confusing and misleading and may result in making wrong design or database system choices. The task of performing the benchmark in a new application domain or using newly developed database systems, like NoSQL systems, can be even more drawn-out due to the lack of a good benchmark. In order, to be suitable for testing the performance and usability of a wide range of database systems and to be useful in simulating a wide variety of application use cases, we define the following desirable characteristics of the benchmark.

Easy to use: In order to be suitable for a wide variety of application and cater to different types of database users, the benchmark should be easy to configure, run, use, and extend. The results of the benchmarking process must be easy to understand so that they are suitable for making objective decisions.

Distributed Application Aware: Increasingly, typical applications are deployed as a set of distributed database clients, spread across an often large geographical area that may span continents. To successfully emulate such application

scenarios, the benchmark itself must be distributed in nature to simulate these geographically distributed database clients. The framework must make it easy to define a workload simulating a distributed application; distribution and coordination of the workload should be handled efficiently and correctly. The results of the execution of the benchmark should be gathered across all the benchmark workload instances in a correct and efficient manner as though they were running on a single machine.

Negligible Impact on Results: The benchmark itself and the infrastructure needed to perform the distribution must be sufficiently light-weight so that it neither becomes a performance bottleneck nor adversely skews the recorded measurements. This means that the benchmark itself should be able to scale and that changing the implementation of the benchmark should not affect the measurement results.

Fine-Grained Measurements: Measurements obtained during the benchmark should be collected and stored in a suitably fine-grained manner so that they can be easily sliced and diced for further analysis. This does not preclude collection of aggregate metrics as long as raw, unprocessed data is captured as well. Therefore, the size of collected measurement data can become large, e.g., for high request numbers per second or long running benchmarks. Thus, in combination with the requirement for negligible impact on results, efficient data structures are needed to persist measurements.

Repeatability: The workload and operations performed during the execution of the benchmark should be repeatable. This will enable an identical workload to be run against different database systems in various configurations in order to perform comparative analysis and A-B testing. For instance, it may be needed to perform an analysis based on whether encryption is enabled or not. This requires being able to replay exactly the same sequence of operation of another benchmarking run.

Wide Applicability to Application Domains: The benchmark should closely emulate a wide variety of application use cases in the form of predefined workloads so that it can be used by application designers and architects to objectively pick a suitable database system that meets their needs. Extending the predefined workloads to incorporate application-specific operations or scenarios in order to enable an apples-to-apples comparison of the database systems under test should also be easy. There should be an ability to define workloads in the form of a mix of a variety of operations to be executed in different execution patterns to simulate a wide variety of application scenarios. This should include the ability to simulate cases of increasing and decreasing load and differing periodicities of workload intensity such as sudden spikes and troughs. Workloads should not be limited to either OLTP or OLAP applications.

Provide Suitable Abstractions: The benchmark should not make any assumptions about the specific capabilities of the database under test and be unaware of the specific database implementation. For instance, it must not know whether the database supports transactions with ACID guarantees. The interface to the

database should be abstracted to enable this behavior. However, a too high abstraction level can result in workloads that only use limited software features provided by a database system under test. Thus, the implementation of specific workload scenarios becomes more difficult. Similarly, there should not be any assumptions about other QoS guarantees provided by the underlying database. Modern database systems make explicit but frequently also implicit QoS trade-offs decisions in their implementation. The benchmark should track both sides of the trade-off, i.e., different QoS dimensions. For instance, instead of making assumptions about transactional capabilities as prescribed by ACID guarantees it should track transaction isolation violations during the execution of the workload.

Support Micro-Analysis: Many evaluations of database systems use micro-benchmarks to target specific database system features, e.g., index structures, using a subset of an application scenario. To be useful in such situations, the benchmark must allow the user to define targeted patterns of operations in the benchmark that can be used to define micro-benchmarks which analyze specific, database features in isolation. Based on the more high-level application workload, micro-benchmarks should be a direct result of a drill-down in the workload stack, i.e., the benchmark should allow to enable and disable all operations of the high-level, application-driven workload individually.

Support Different Deployment Topologies: The benchmark should be able to simulate scenarios in which database applications as well as the database itself are deployed in various deployment topologies including geo-distributed deployments. Increasingly, applications also use one or more database technologies simultaneously working in concert. Benchmarking these combined setups in parallel should be supported in order to simulate such distributed application scenarios.

In our opinion, a benchmarking middleware is a suitable architecture for building a benchmarking framework that fulfills the requirements above. Due to today's speed of innovation in database systems, such a middleware may even have become a necessity. This is largely because of the following reasons:

- A middleware-based architecture will enable reuse which mitigates the risk of having to rebuild the infrastructure needed for new benchmarking applications. This in turn reduces the risk of mistakes in the benchmarking application leading to distorted measurement results.
- The application developer or database administrator can focus on the actual application-specific workload instead of worrying about the infrastructure needed to run a benchmark (e.g., distributed execution and coordination or the mapping between operations and database interface).
- The middleware abstracts the measurement and collection of a wide variety of metrics and performance characteristics across the various QoS dimensions. For instance, it is a non-trivial task to either detect transaction isolation anomalies as a result of transactional ACID property violations or to measure degrees of database (in-)consistency.

3 Requirements for a Benchmarking Middleware

Building on the requirements for distributed database benchmarks which we identified in Sect. 2, we will now analyze middleware features necessary for creating middleware-supported benchmarks according to the mentioned benchmarking requirements. We will use this to identify requirements that the middleware for database benchmarking should fulfill.

In order to increase ease of use and allow for wide applicability to application domains, a ready-to-use toolkit which does not make any assumptions about the application domain or application scenario is required. This is closely linked to providing suitable abstractions as well as supporting distributed workloads and deployments, i.e., the benchmarking middleware itself should be completely unaware of anything happening above or below the middleware layer; it should, hence, be entirely *database- and application-agnostic* and be able to handle *distribution and coordination*. Ease of use also calls for the middleware tool to come bundled with *built-in basic workloads* that are ready to use.

Fine-grained measurements will create large amounts of data so that a benchmarking middleware must be able to handle such a data stream in terms of persistence, i.e., *storage of fine-grained results* is required. Due to the complexity of implementing such measurements, the measurement process itself should also be handled by the middleware – not only for well-explored QoS dimensions like latency or throughput but also for more complex dimensions like consistency or transaction isolation. Therefore, a benchmarking middleware should come bundled with *support for advanced QoS dimensions*. Finally, a benchmarking implementation with little or no impact on repeatable measurement results obviously requires the same of an underlying middleware solution, that is, an *efficient implementation* offering a *trace-based execution* of experiments.

Building on this connection between requirements for benchmarks and the corresponding requirements for a benchmarking middleware, we will now discuss each of the middleware requirements in more detail:

Database- and Application-Agnostic: The middleware should provide a one-size-fits-all framework that supports all kinds of workloads with a variety of database access patterns (e.g., changing workload intensity, OLTP and OLAP, transactional and non-transactional, complex queries and key-based access, etc.) without any assumptions on the application scenario or the application workload. In addition to this, the middleware needs to be independent of the underlying database and should be equipped with a flexible set of mappers that map and facilitate the execution of its workloads on the different data models supported by different database systems (e.g., relational, column-oriented, document-oriented, and key-value) with the flexibility to support the definition of new data models with associated mappers.

Support for Advanced QoS Dimensions: In addition to measuring the standard evaluation metrics for database systems like response time and throughput, the middleware should facilitate the measurement of advanced QoS metrics such as consistency [4,9–12,26,48,50], availability and elasticity [43], where applicable.

Ideally, the measurement component of the middleware should be extensible so that it provides the ability to define new metrics and to plug-in new measurement modules for these newly defined metrics. Furthermore, for cloud-based deployments, the middleware should be able to track concrete usage of cloud resources (e.g., storage, CPU, network, etc.) in order to facilitate monetary cost benchmarking which is an important aspect of cloud environments [23].

Storage of Fine-Grained Results: The middleware needs to track and store all the raw measurement data which introduces a certain degree of complexity as raw measurement results can potentially become very large. To ease further analysis and to also offer convenient access to aggregated results, the middleware should also come bundled with a data analysis module that provides the ability to gain insights from the collected large amount of measurement data by suitably aggregating, transforming, correlating, and analyzing the raw measurement data.

Efficient Implementation: The middleware itself should not have a significant negative impact on the performance of the benchmark and the measurements of the evaluation metrics, i.e., choosing a different middleware implementation should not notably affect the measurement results.

Distribution and Coordination: The middleware should manage distribution and coordination of (potentially geographically) distributed measurement clients and do so in a way transparent to the benchmark developer.

Trace-Based Execution: A repeatable benchmark execution, e.g., for A-B testing, requires a trace-driven implementation so that a priori instead of ad hoc workload generation is the logical choice. Obviously, such an operation trace should also contain information on the measurement client issuing the operation according to the desired level of distribution. This will assert that repeated executions will issue the same operations after the same test duration from the same geolocation. Depending on whether the load balancer is considered to be part of the application or the database system, this may even require – in the case of replication – to issue these same requests also to the same replicas. This is especially important when measuring consistency behavior [8].

Built-in basic workloads: The middleware should come bundled with a basic set of built-in basic workloads which are ready to use. These can then also be used as building blocks for advanced workloads more closely resembling the actual workload of a concrete application use case. This will also serve as a way to perform an apples-to-apples comparison between competing database systems for general purpose use without a concrete application in mind. The TPC suite of benchmarks and the default workloads provided by YCSB [15] have played a similar role in the past.

4 An Architecture for a Benchmarking Middleware

To fulfill the requirements identified in the last section, a benchmarking middleware needs to be extensible in several dimensions. We aim to address these with the following high-level architecture (see also Fig. 1 which gives an overview):

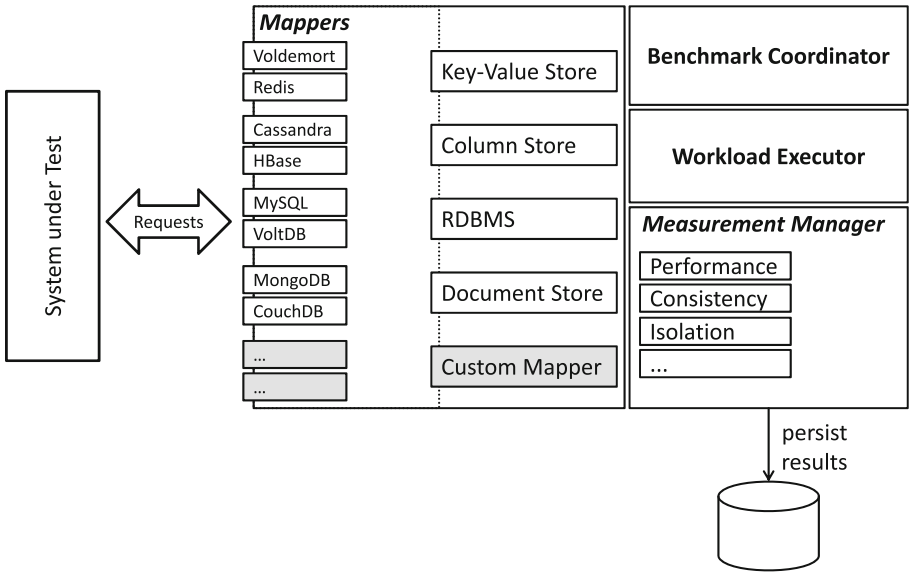

Fig. 1. High level architecture of the proposed benchmarking Middleware

Mappers are responsible for mapping queries to a particular database instance. This is implemented in two stages. In the first stage, queries are mapped to a particular type of database, e.g., a column store, but are not yet bound to a specific database system, e.g., Cassandra [31]. In the second stage, the abstract interface is mapped to a concrete database system. This way, standard mappers for different types of databases can be supported along with custom mappers. Furthermore, this also enables the ability to support additional database systems as well as entirely new types of database systems in the future.

For example, the first stage may choose to store an object as well as all objects referenced by this object through one-to-many relationships under the same key in a key-value store. It may also choose to do so in the JSON format. The second stage, in contrast, will then map the abstract CRUD interface of the key-value store to, for instance, Voldemort. This staged mapper approach keeps the middleware entirely agnostic of both the application workload and the underlying database system under test.

As another example, a query retrieving a data item based on three filter criteria might be mapped to an RDBMS with a query like `SELECT * FROM table WHERE a AND b AND c`. For a column store, in contrast a standard mapping might be to use the concatenated values of the fields referenced by a, b and c as row key and to, therefore, issue a get query for this row key.

The *Workload Executor* is responsible for executing all requests of the workload according to the workload specification. For this purpose, we propose to use a precomputed operation trace, i.e., a priori workload generation, to remove much of the necessary coordination effort. We imagine that a secondary tool will be used to create operation traces – either based on real-world traces or the

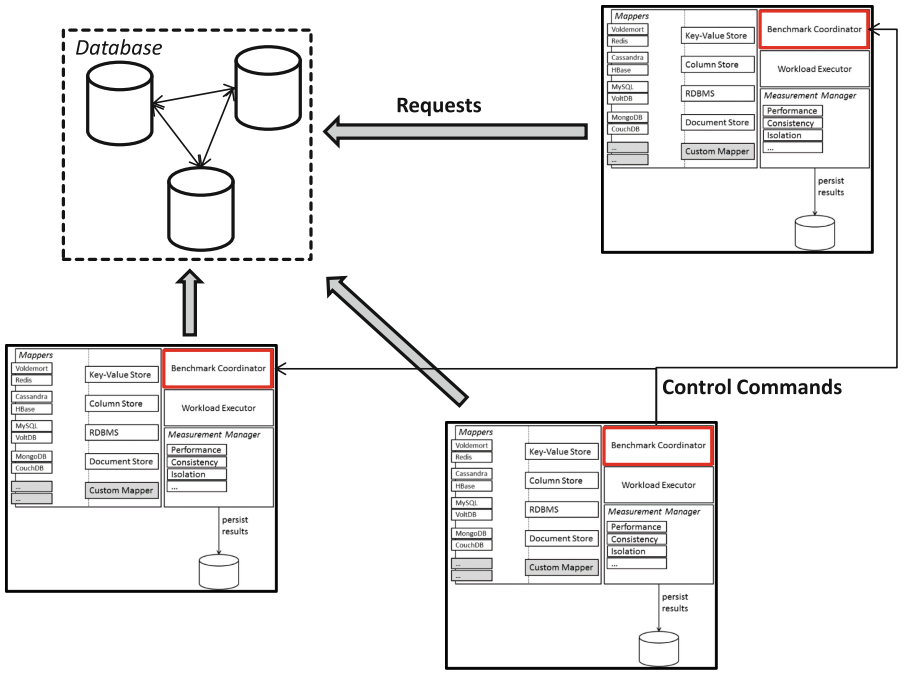

Fig. 2. Coordination between multiple benchmarking instances

corresponding workload description – in a standardized format, e.g., as tuples containing a relative timestamp, a SQL query and an origin location. Through this a priori workload generation, the Workload Executor will maintain the repeatability requirement. The requirement of having an efficient implementation is largely independent of the proposed architecture and mainly depends on the concrete implementation but an a priori workload generation through an external tool will certainly help the middleware layer to remain relatively thin by moving most coordination overhead to a time before the actual benchmark execution.

The *Measurement Manager* keeps track of all the individual measurement modules (e.g., performance or consistency behavior) and their individual measurement outputs. For this purpose, the manager provides detailed information to the modules (e.g., start and end time of requests or business transactions, operation results, etc.) and persists the metric output in a local database for post-processing. This component will include predefined measurement modules to determine latency, throughput, availability, elasticity, scalability, and consistency behavior. Therefore, the requirements of being able to store fine-grained results as well as support for advanced QoS metrics will not be violated.

As a typical benchmark run will be distributed, there will be different benchmarking instances which need to be coordinated. The *Benchmark Coordinator* is, therefore, responsible for communication across instances to assert, for instance, that all benchmarking instances have the correct information available

on operation traces, start times, etc. We propose a master-based approach as a single point of failure is, in this particular case, actually desirable, since failures will render benchmarking results unusable and fault-tolerance mechanisms would hide this from the user. The coordination of benchmarking instances is illustrated in Fig. 2. This approach requires sufficiently precise clock synchronization as provided by NTP[13] as well as the use of reliable messaging protocols for communication. Implementing the Benchmark Coordinator component will fulfill the requirement of handling coordination and distribution within the middleware layer.

5 Related Work

Related work on benchmarking middleware for distributed databases is scarce. Difallah et al. [21] propose an extensible testbed for the execution of benchmarks against relational databases. In particular, they argue for encapsulation of recurring functionality within a universal benchmarking infrastructure. Therefore, the work is closely related to our proposed benchmarking middleware. To our knowledge, this is the sole publication addressing not only a subset of our use case.

In the following, we discuss related work in three groups: (i) identified *benchmarking requirements* for database systems in different contexts, e.g., cloud environments, (ii) existing *benchmarking approaches* that address subsets of requirements outlined in Sects. 2 and 3 and (iii) specialized *foundational works* that address single functionalities of a benchmarking middleware, e.g., distributed generation of synthetic data sets.

Benchmarking Requirements: A good benchmarking middleware must identify requirements for building meaningful and useful benchmarks. Huppler [29] discusses five general characteristics of a good benchmark, i.e., *relevant*, *repeatable*, *fair*, *verifiable* and *economical*. Nambiar and Poess argue that database technologies are changing at such a rapid pace that deployment of benchmarks have became increasingly complicated. Therefore, easier means to develop and execute benchmarks are required. Smith [46] and Folkerts et al. [24] discuss requirements for benchmarks in cloud environments. Furthermore, Poess [37] and Baru et al. [6] discuss requirements for Big Data environments, respectively. Bermbach [8] discusses requirements for (consistency) metrics. Specifically, these have to be *meaningful*, *fine-grained*, have a *high resolution* and allow *reproducible* measurement results.

Benchmarking Approaches: Benchmarking approaches are *application-driven* or *system-driven*. Application-driven approaches, i.e., end-to-end benchmarks, focus on providing realistic and meaningful workloads for an application domain. System-driven approaches are often micro-benchmarks and typically build on synthetic workload generation; they focus on a broad coverage of workloads to measure isolated database features.

[13] ntp.org.

Examples of application-driven approaches grouped by application domains are: OLTP [17,18], decision support [16,25], social media [5], CEP [32], ETL [49]. Examples of system-driven approaches are the Yahoo! Cloud Serving Benchmark (YCSB) [15] and Rain [7]. YCSB provides an extensible benchmarking tool with adapters for a number of distributed database systems, e.g., HBase, Cassandra and MongoDB. Two extensions to YCSB are YCSB++ [36] and YCSB+T [20]. YCSB++ adds new features to YCSB: bulk loading for databases based on B-trees, Zookeeper-based [28] start of distributed YCSB clients and distributed monitoring based on Ganglia [33]. YCSB+T extends the original workload by providing the ability to define multi-item transactions and a data validation and anomaly detection phase that can be used to classify and quantify database anomalies introduced by the workload. Rain [7] is a workload generation toolkit that provides a load scheduling mapper that can be extended with application-specific workload generators.

Different TPC benchmarks do not provide an implementation and workload models provide limited flexibility, thus, setup and customization requires additional effort. YCSB allows the generation of synthetic workloads based on a stochastical workload model. For a large number of application workloads, an accurate emulation based on the workload model is impossible or difficult to instantiate. Furthermore, support for important features, e.g., benchmark distribution and customized collection of metrics, are limited by existing benchmarks and frameworks. Our proposed benchmarking middleware closes the gap between - application-driven and system-driven workload models and provides frequently used benchmarking functionals in a transparent way.

Foundational Work: To realize the different components of our envisioned benchmarking middleware, related work from different areas must be taken into account. To address this, we discuss related work with selected examples, which include, distributed data set generation for the Benchmark Coordinator, workload scheduling for the Workload Executor and complex QoS-metrics for the Measurement Manager. Gray et al. [27] discuss the generation of synthetic data sets. This work has been extended with a focus on distributed generation of data sets by Alexandrov et al. [1–3] and Rabl et al. [38,40]. In essence, both these approaches aim to provide a speed-up through parallelization with regard to the preparation time of database benchmarks. Schroeder et al. [45] propose additional parameters to be explicitly considered during workload generation, namely a scheduling model for requests. The model differentiates between workload generators that do or do not schedule new requests independent of received responses to the preceding request. Since, trade-offs exist between QoS metrics, it is not sufficient to characterize database systems based on a single QoS dimension, e.g., performance. Among others, the following examples of benchmarking approaches address specific system qualities: (i) *consistency* [4,9–12,26, 48,50], (ii) *dependability* [47], (iii) *scalability* [15,39], (iv) *elasticity* [22,30], and (v) *security* [34].

6 Conclusion

Historically, industry standard database benchmarks have enabled healthy competition among rival vendors that resulted in improved product offerings and was also a significant contributing factor towards the evolution of database systems themselves. While these benchmarks continue to serve both the industry and research community well, they lack some of the flexibility and extensibility required by modern cloud-based application systems in which different types of data management systems often coexist and complement each other. Further, these applications often make QoS and performance trade-offs based on a much wider set of requirements and criteria – yet, in current database benchmarks the ability to study these trade-offs is missing. In addition to this, today's database benchmarks either do not come with a read-to-use toolkit or are limited to certain kinds of database systems and based on synthetic workloads.

In this paper, we describe our vision and architecture of a middleware for benchmarking different databases and workloads. The authors plan to further extend as well as actually implement this framework to provide the necessary infrastructure for benchmarking database systems with regards to arbitrary QoS dimensions and trade-offs, to help in determining price-performance trade-offs, and to enable modern benchmarks for studying QoS behavior of multi-tenant, frequently cloud-based, federated multi-database environments.

References

1. Alexandrov, A., Brücke, C., Markl, V.: Issues in big data testing and benchmarking. In: Proceedings of the Sixth International Workshop on Testing Database Systems, DBTest 2013, pp. 1:1–1:5. ACM, New York (2013)
2. Alexandrov, A., Schiefer, B., Poelman, J., Ewen, S., Bodner, T.O., Markl, V.: Myriad: parallel data generation on shared-nothing architectures. In: Proceedings of the 1st Workshop on Architectures and Systems for Big Data, ASBD 2011, pp. 30–33. ACM, New York (2011)
3. Alexandrov, A., Tzoumas, K., Markl, V.: Myriad: scalable and expressive data generation. Proc. VLDB Endowment **5**(12), 1890–1893 (2012)
4. Anderson, E., Li, X., Shah, M.A., Tucek, J., Wylie, J.J.: What consistency does your key-value store actually provide? In: Proceedings of the 6th Workshop on Hot Topics in System Dependability (HOTDEP), HotDep 2010, pp. 1–16. USENIX Association, Berkeley (2010)
5. Armstrong, T.G., Ponnekanti, V., Borthakur, D., Callaghan, M.: LinkBench: a database benchmark based on the facebook social graph. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, pp. 1185–1196. ACM, New York (2013)
6. Baru, C., Bhandarkar, M., Nambiar, R., Poess, M., Rabl, T.: Setting the direction for big data benchmark standards. In: Nambiar, R., Poess, M. (eds.) TPCTC 2012. LNCS, vol. 7755, pp. 197–208. Springer, Heidelberg (2013)
7. Beitch, A., Liu, B., Yung, T., Griffith, R., Fox, A., Patterson, D.A.: Rain: a workload generation toolkit for cloud computing applications. Technical report, University of California at Berkeley (2010)

8. Bermbach, D.: Benchmarking eventually consistent distributed storage systems. Ph.D. thesis, Karlsruhe Institute of Technology, Germany, February 2014, to be published
9. Bermbach, D., Kuhlenkamp, J.: Consistency in distributed storage systems. In: Gramoli, V., Guerraoui, R. (eds.) NETYS 2013. LNCS, vol. 7853, pp. 175–189. Springer, Heidelberg (2013)
10. Bermbach, D., Tai, S.: Eventual consistency: how soon is eventual? An evaluation of amazon S3's consistency behavior. In: Proceedings of the 6th Workshop on Middleware for Service Oriented Computing (MW4SOC), MW4SOC 2011, pp. 1:1–1:6. ACM, New York (2011)
11. Bermbach, D., Tai, S.: Benchmarking eventual consistency: lessons learned from long-term experimental studies. In: Proceedings of the 2nd International Conference on Cloud Engineering (IC2E). IEEE (2014)
12. Bermbach, D., Zhao, L., Sakr, S.: Towards comprehensive measurement of consistency guarantees for cloud-hosted data storage services. In: Nambiar, R., Poess, M. (eds.) TPCTC 2013. LNCS, vol. 8391, pp. 32–47. Springer, Heidelberg (2014)
13. Cattell, R.: Scalable SQL and NoSQL data stores. SIGMOD Record **39**(4), 12–27 (2010)
14. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: a distributed storage system for structured data. In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI), OSDI 2006, pp. 205–218. USENIX Association, Berkeley (2006)
15. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st Symposium on Cloud Computing (SOCC), SOCC 2010, pp. 143–154. ACM, New York (2010)
16. T. P. P. Council. TPC benchmark DS: standard specification version 1.1.0. Technical report, Transaction Processing Performance Council (2012)
17. T. P. P. Council. TPC benchmark e: standard specification version 1.13.0. Technical report, Transaction Processing Performance Council (2014)
18. T. T. P. Council. TPC benchmark c: standard specification revision 5.11. Technical report, The Transaction Processing Council (2010)
19. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: Amazon's highly available key-value store. In: Proceedings of 21st Symposium on Operating Systems Principles (SOSP), SOSP 2007, pp. 205–220. ACM, New York (2007)
20. Dey, A., Fekete, A., Nambiar, R., Röhm, U.: YCSB+T: benchmarking web-scale transactional databases. In: 2014 IEEE 30th International Conference on Data Engineering Workshops (ICDEW), pp. 223–230, March 2014
21. Difallah, D., Pavlo, A.: OLTP-bench: an extensible testbed for benchmarking relational databases. Proc. VLDB Endowment **7**(4), 277–288 (2013)
22. Dory, T., Mej, B., Roy, P.V.: Measuring elasticity for cloud databases. In: Proceedings of the Second International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2011), pp. 154–160 (2011)
23. Florescu, D., Kossmann, D.: Rethinking cost and performance of database systems. SIGMOD Record **38**(1), 43–48 (2009)
24. Folkerts, E., Alexandrov, A., Sachs, K., Iosup, A., Markl, V., Tosun, C.: Benchmarking in the cloud: what it should, can, and cannot be. In: Nambiar, R., Poess, M. (eds.) TPCTC 2012. LNCS, vol. 7755, pp. 173–188. Springer, Heidelberg (2013)

25. Ghazal, A., Rabl, T., Hu, M., Raab, F., Poess, M., Crolotte, A., Jacobsen, H.-A.: BigBench: towards an industry standard benchmark for big data analytics. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, pp. 1197–1208. ACM, New York (2013)
26. Golab, W., Li, X., Shah, M.A.: Analyzing consistency properties for fun and profit. In: Proceedings of the 30th Symposium on Principles of Distributed Computing (PODC), PODC 2011, pp. 197–206. ACM, New York (2011)
27. Gray, J., Sundaresan, P., Englert, S., Baclawski, K., Weinberger, P.J.: Quickly generating billion-record synthetic databases. In: ACM SIGMOD Record, vol. 23, pp. 243–252. ACM (1994)
28. Hunt, P., Konar, M., Junqueira, F., Reed, B.: ZooKeeper: wait-free coordination for Internet-scale systems. In: USENIX ATC (2010)
29. Huppler, K.: The art of building a good benchmark. In: Nambiar, R., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 18–30. Springer, Heidelberg (2009)
30. Kuhlenkamp, J., Klems, M., Röss, O.: Benchmarking scalability and elasticity of distributed database systems. PVLDB **7**(12), 1219–1230 (2014)
31. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. SIGOPS Operating Syst. Rev. **44**(2), 35–40 (2010)
32. Li, C., Berry, R.: CEPBen: a benchmark for complex event processing systems. In: Nambiar, R., Poess, M. (eds.) TPCTC 2013. LNCS, vol. 8391, pp. 125–142. Springer, Heidelberg (2014)
33. Massie, M.L., Chun, B.N., Culler, D.E.: The ganglia distributed monitoring system: design, implementation, and experience. Parallel Comput. **30**(7), 817–840 (2004)
34. Müller, S., Bermbach, D., Tai, S., Pallas, F.:Benchmarking the performance impact of transport layer security in cloud database systems. In: Proceedings of the 2nd International Conference on Cloud Engineering (IC2E). IEEE (2014)
35. Nambiar, R., Poess, M.: Keeping the TPC relevant!. Proc. VLDB Endowment **6**(11), 1186–1187 (2013)
36. Patil, S., Polte, M., Ren, K., Tantisiriroj, W., Xiao, L., López, J., Gibson, G., Fuchs, A., Rinaldi, B.: YCSB++: benchmarking and performance debugging advanced features in scalable table stores. In: Proceedings of the 2nd Symposium on Cloud Computing (SOCC), SOCC 2011, pp. 9:1–9:14. ACM, New York (2011)
37. Poess, M.: TPC's benchmark development model: making the first industry standard benchmark on big data a success. In: Rabl, T., Poess, M., Baru, C., Jacobsen, H.-A. (eds.) WBDB 2012. LNCS, vol. 8163, pp. 1–10. Springer, Heidelberg (2014)
38. Rabl, T., Frank, M., Sergieh, H.M., Kosch, H.: A data generator for cloud-scale benchmarking. In: Nambiar, R., Poess, M. (eds.) TPCTC 2010. LNCS, vol. 6417, pp. 41–56. Springer, Heidelberg (2011)
39. Rabl, T., Gómez-Villamor, S., Sadoghi, M., Muntés-Mulero, V., Jacobsen, H.-A., Mankovskii, S.: Solving big data challenges for enterprise application performance management. Proc. VLDB Endowment **5**(12), 1724–1735 (2012)
40. Rabl, T., Jacobsen, H.-A.: Big data generation. In: Rabl, T., Poess, M., Baru, C., Jacobsen, H.-A. (eds.) WBDB 2012. LNCS, vol. 8163, pp. 20–27. Springer, Heidelberg (2014)
41. Sakr, S.: Cloud-hosted databases: technologies, challenges and opportunities. Cluster Comput. **17**(2), 487–502 (2014)
42. Sakr, S., Casati, F.: Liquid benchmarks: towards an online platform for collaborative assessment of computer science research results. In: Nambiar, R., Poess, M. (eds.) TPCTC 2010. LNCS, vol. 6417, pp. 10–24. Springer, Heidelberg (2011)
43. Sakr, S., Liu, A.: Is your cloud-hosted database truly elastic? In: SERVICES, pp. 444–447 (2013)

44. Sakr, S., Liu, A., Batista, D.M., Alomari, M.: A survey of large scale data management approaches in cloud environments. IEEE Commun. Surv. Tutorials **13**(3), 311–336 (2011)
45. Schroeder, B., Wierman, A., Harchol-Balter, M.: Open versus closed: a cautionary tale. In: Proceedings of the 3rd Conference on Networked Systems Design & Implementation, NSDI 2006, vol. 3, pp. 18–18. USENIX Association, Berkeley (2006)
46. Smith, W.D.: Characterizing cloud performance with TPC benchmarks. In: Nambiar, R., Poess, M. (eds.) TPCTC 2012. LNCS, vol. 7755, pp. 189–196. Springer, Heidelberg (2013)
47. Vieira, M., Madeira, H.: A dependability benchmark for OLTP application environments. In: Proceedings of the 29th International Conference on Very Large Data Bases, VLDB 2003, vol. 29, pp. 742–753. VLDB Endowment (2003)
48. Wada, H., Fekete, A., Zhao, L., Lee, K., Liu, A.: Data consistency properties and the trade-offs in commercial cloud storages: the consumers' perspective. In: Proceedings of the 5th Conference on Innovative Data Systems Research (CIDR), pp. 134–143, January 2011
49. Wyatt, L., Caufield, B., Pol, D.: Principles for an ETL benchmark. In: Nambiar, R., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 183–198. Springer, Heidelberg (2009)
50. Zellag, K., Kemme, B.: How consistent is your cloud application? In: Proceedings of the 3rd Symposium on Cloud Computing (SOCC), SOCC 2012, pp. 6:1–6:14. ACM, New York (2012)