A Versatile Framework for Painless Benchmarking of Database Management Systems

Lexi Brent and Alan Fekete(✉)

The University of Sydney, Sydney, NSW 2006, Australia
{lexi.brent,alan.fekete}@sydney.edu.au

Abstract. Benchmarking is a crucial aspect of evaluating database management systems. Researchers, developers, and users utilise industry-standard benchmarks to assist with their research, development, or purchase decisions, respectively. Despite this ubiquity, benchmarking is usually a difficult process involving laborious tasks such as writing and debugging custom testbed scripts, or extracting and transforming output into useful formats. To date, there are only a limited number of comprehensive benchmarking frameworks designed to tackle these usability and efficiency challenges directly.

In this paper we propose a new versatile benchmarking framework. Our design, not yet implemented, is based on exploration of the benchmarking practices of individuals in the database community. Through user interviews, we identify major pain points these people encountered during benchmarking, and map these onto a pipeline of processes representative of a typical benchmarking workflow. We explain how our proposed framework would target each process in this pipeline, potentiating significant overall usability and efficiency improvements. We also contrast the responses of engineers working in industry with those of researchers, and examine how database benchmarking requirements differ between these two groups. The framework we propose is based around traditional synthetic workloads, would be simple to configure, highly extensible, could support any benchmark, and write output to any well-defined data format. It would collect and track all generated events, data, and relationships from the benchmark and underlying systems, and offer simple reproducibility. Complex scenarios such as distributed-client and multi-tenant benchmarks would be simplified by the framework's workload partitioning, client coordination, and output collation capabilities.

Keywords: Benchmark · TPCC · YCSB · DBMS · Performance evaluation

1 Introduction

Benchmarking database management systems (DBMSes) is critical for evaluating their correctness, performance, and efficacy. Organizations often employ industry-standard benchmarks before making purchase decisions; researchers use benchmarks to evaluate novel technology; and developers frequently run benchmarks during development to identify bugs or bottlenecks in their systems.

Due to the large variety of different DBMSes, benchmarks, database paradigms, and data formats available today, benchmarking has become an unnecessarily complex undertaking typically involving laborious manual processes and repetitive tasks [4,5,14]. This heterogeneity led to the development of new tools assisting with aspects of benchmarking, such as data and workload generation [1,2,12,13,16], precise control over request rate and transaction mixture [18], collection of statistics and environment information, and workload- or target-specific testbeds or frameworks [8,9,19]. With the rise of cloud-based distributed DBMSes, new benchmarking frameworks such as [15] have emerged targeting properties such as horizontal scaling, elasticity, and availability, with support for automated provisioning of cloud resources.

Previous research has largely focused on distinct benchmarking sub-processes or specific scenarios, rather than taking a holistic approach. For instance, OLTP-Bench [8] provided extensible support for running industry-standard benchmarks targeting relational DBMSes with a focus on fine-grained control over request rates, transaction mixtures, and access distributions; YCSB [6] provided a benchmark for large-scale distributed cloud database systems; YCSB+T [7] extended YCSB with support for transactional workloads; the TPC [17] benchmarks focused on performance evaluation of relational DBMSes; MTCB [20] provided a benchmark for multi-tenant OLTP systems; UDBMS [11] implemented a data model for benchmarking multi-model database systems; and MUDD [16], PSDG [10], PDGF [12], and NoWog [1] provided automated test data generation.

Only a few studies [3–5] have been concerned with building a comprehensive, extensible framework focusing on usability and the whole benchmarking process. Most notably, BenchFoundry [3] implemented support for deterministic trace-based workload generation within an extensible distributed benchmarking framework capable of supporting several SQL and NoSQL systems. Deterministic, trace-based workload generation makes it difficult to control the statistical distribution of inputs to match real-world situations. Implementations of traditional benchmarks within BenchFoundry may produce results inconsistent with synthetic workloads based on random sampling from a statistical distribution [5]. Additionally, BenchFoundry does not collect detailed environment metadata from benchmark clients and servers. Such metadata is often critical for assessing the validity of performance benchmarks, which need to be conducted under tightly controlled conditions and should not be impacted by resource or benchmarking bottlenecks [5].

In practice, benchmarking platforms are typically based on a series of shell scripts and configuration files that handle everything from collecting environment information to transforming benchmark output into a useful data format, in addition to executing the benchmark. Depending on the experiment, there may be several different versions of each script or configuration file typically distinguished by "meaningful" filenames. Little imagination is required to realize this often becomes chaotic. These scripts are often developed from scratch and specialised to specific systems or benchmarks, and therefore not easily adaptable. As we show in Sect. 2, these custom scripts are a major time sink and source of bugs in benchmarking workflows.

In this paper, we propose and envision a new benchmarking framework towards solving these issues and improving practices. Unlike most previous work, our proposal has a focus on usability. Uniquely, it is based on interviews revealing the real benchmarking practices of several academics and industry professionals. The result would be a highly general, extensible, and versatile framework incorporating the whole benchmarking process; from DBMS and benchmark configuration to output processing and statistical analysis, with a focus on usability and meeting the benchmarking needs of both industry and academia. While our framework proposal incorporates some ideas previously presented in the OLTP-Bench [8] and BenchFoundry [3] papers, our focus is fundamentally at a higher level. Our vision aims to remove difficulties reported by some highly experienced people; it uses traditional synthetic workloads based on sampling from a statistical distribution, while still focusing heavily on experimental repeatability.

The main contributions of this paper are:

1. An interview-based analysis of pain points in current practices and identification of similarities and differences between academia and industry.
2. The design of a new benchmarking framework addressing the pain points we identified.

The remainder of this paper is structured as follows. In Sect. 2 we describe our exploration of the benchmarking practices of academics and industry professionals. Based on this, we identify a set of major pain points in benchmarking processes and contrast the responses of industry professionals with those of academics. We show that benchmarking may be encapsulated by a *pipeline* of key processes, and we map the major pain points onto this pipeline. In Sect. 3, we describe and envision our proposed framework alongside example use cases. Section 4 describes possible avenues for future work.

2 Current Benchmarking Practices

The design and functionality of our new benchmarking framework envisioned in Sect. 3 is heavily informed by awareness of current benchmarking practices. We interviewed five people in order to gain a deeper understanding of the spectrum of practices currently employed within the community. These interviews focused on identifying pain points, time sinks, and potential improvements within respondents' existing processes.

2.1 Interview Process

Interview questions covered three broad areas: (i) **systems** including DBMSes under test, benchmarking infrastructure, and benchmarking tools; (ii) **processes** including experiment configuration, workload, data collection, statistical analysis, transformation of raw output data, storage and management of results, and measurement dimensions; and (iii) **features/functionality** desired in a benchmarking framework. We also provided respondents with a list of potential features for our new framework, and asked which features would be most applicable to their workflow. Finally, respondents were given an opportunity to suggest new features to resolve existing issues they had identified in their own benchmarking processes. Our full set of interview questions is available online[1].

Interviewees included some academic researchers, and some engineers working in industry on deployed DBMS systems. All respondents were asked identical questions regardless of their respective backgrounds, and encouraged to provide as much detail as possible. Some additional impromptu questions were asked to clarify responses or request further detail. Interviews were conducted verbally in-person or via teleconference, in a single block of time between 30 and 45 min, with responses transcribed as the respondents spoke. While the number of people involved is small, they cover a variety of situations, and so we expect that improving the issues they mentioned will have wide benefits.

These interviews were conducted in December 2015. In March 2018 we conducted a follow-up email asking some of the original respondents from both industry and academia if any significant changes to their benchmarking tools, processes, or methodology had occurred since 2015. In their replies, they reported no significant changes. Hence, we are confident that our analysis of pain points, and our proposed framework, remain relevant to the community in 2018.

2.2 Insights of Interest

Based on interview responses, we created a summary of the key challenges faced by each respondent in their benchmarking workflows. We then used those summaries to build the following taxonomy, in which each "pain point" corresponds to a key challenge raised by at least one respondent:

PP1. Initial setup and configuration. The deployment and configuration of a benchmarking experiment is often time-consuming and unintuitive.

PP2. Script writing. It is often necessary to write and debug custom testbed scripts, which is laborious and time-consuming.

PP3. Reproducibility. Repeatability and reproducibility are difficult to accomplish, usually involving a manual process of referring to multiple information sources to configure and re-execute an experiment.

PP4. Debugging. Unexpected results are difficult to substantiate, usually requiring time-consuming manual debugging.

PP5. Distributed clients. Distributed benchmarks often require manual coordination of clients and collation of output.

PP6. Metadata collection. Collecting additional system metadata (e.g. system calls) during a benchmark run requires writing custom scripts, coordinating these, and manually correlating output.

[1] https://github.com/lexibrent/benchfw-resources/blob/master/interview-qns.pdf.

PP7. Log correlations. Correlating events recorded by benchmark clients with those recorded in DBMS logs requires manual inspection or custom scripts.

PP8. Statistical analysis. Statistical exploration of benchmark metrics (e.g. computing correlation coefficients) is often fruitful but typically too laborious and time-consuming to be feasible.

Some notable similarities and differences were observed between the responses from researchers and industry professionals:

– Industry respondents tended to focus more on reliability and efficiency than academic respondents. For example, industry respondents expressed a desire to measure "consistency of throughput", "response time variance", and "latency with a threshold".
– Industry respondents tended to focus on applications surrounding debugging and continuous integration, whereas academic respondents primarily focused on scientific applications such as experimentation with novel technologies.
– Academic respondents were more concerned with statistical and experimental validity and repeatability than industry respondents. For example, academic respondents discussed conducting multiple trials, and methods of dealing with statistical outliers. Industry respondents did not pay much attention to these topics, with some indicating they would typically only run a benchmark multiple times to assist with debugging, rather than to improve statistical reliability.
– All respondents indicated they use cloud services such as Amazon EC2 extensively in their benchmarking processes.
– All respondents agreed that distributed benchmark clients are difficult to coordinate, but industry respondents appeared to exhibit less interest in conducting distributed benchmarks than academic respondents.
– Industry respondents' processes tended to focus on short-length, single-client workloads, whilst academic respondents emphasised the importance of longer-running and mixed-client workloads.

2.3 Further Analysis of Benchmarking Processes

Our interviews and our own experiences suggested that benchmark execution can be represented as a *pipeline* of three main processes: (i) **initial configuration**, (ii) **benchmark runs**, and (iii) **results processing/analysis**. This pipeline model, depicted in Fig. 1, is consistent with observations of others in the community [5] who also approach benchmarking as a pipeline, albeit from a different perspective.

In Table 1 we assign each identified pain point to one or more pipeline processes. We observe that *initial configuration* and *results processing/analysis* are the two major sources of pain and time consumption, potentiating the greatest improvements in efficiency and usability for the overall benchmarking pipeline. Hence, our new benchmarking framework proposed in Sect. 3 is motivated by improving the efficiency of the pipeline by finding solutions to these pain points.

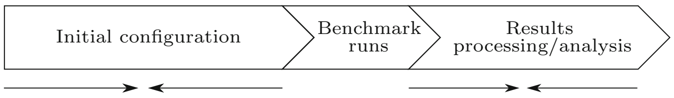

Fig. 1. Benchmarking represented as a pipeline. *Initial configuration* and *results processing/analysis* are the most painful processes, potentiating the largest efficiency and usability improvements (indicated by inward arrows).

Table 1. Association of identified pain points with benchmark pipeline processes.

Pipeline process	Major pain points
Initial configuration	PP1, PP2, PP3, PP4, PP5, PP6
Benchmark runs	PP4, PP5
Results processing/analysis	PP4, PP5, PP6, PP7, PP8

3 New Benchmarking Framework

In this Section, we propose and envision a new framework towards addressing the major pain points described in Sect. 2, with the overall goal of decreasing the inefficiency and pain associated with DBMS benchmarking. The framework we envision is founded on several key principles:

1. **Generality and versatility**—no restrictions should be imposed regarding the: benchmark; configuration parameters (DBMS/benchmark); target DBMS; workload; dataset; experimental method; or outputs. A relevant benchmarking framework should be capable of handling the heterogeneity of modern DBMS benchmarking.
2. **Extensibility and abstraction**—the framework should be highly extensible in all directions, with a modular design utilizing multiple abstraction layers. It should be straightforward to implement or extend benchmarks, workloads, target DBMSes, experimental methodologies, etc. This principle responds to the rapid pace of development within the database community and it aims to ensure the proposed framework's ongoing relevance.
3. **Usability and configurability**—the framework should be simple to install, configure, and run, painless to extend, and provide intuitive output. All aspects of the benchmarking pipeline should be separately and extensively configurable using a simple self-documenting configuration format. Running traditionally complex distributed benchmarks should be as simple as specifying a few configuration parameters.
4. **Track everything**—as much information as practically possible (i.e. without interfering with results) should be collected about the benchmarking environment. Metadata about relationships between information and objects within the system should also be collected. More context is better than less when reviewing benchmarking outputs.
5. **Repeatability/reproducibility**—replicating an experiment for result verification and consistency should be as simple as running a command.

6. **Flexibility of output**—the framework should be capable of outputting any well-defined format specified by the user, and of re-writing output in different formats after experiment completion. It should be straightforward to extend the framework with custom output formats.

Through consistent focus on these key principles, our proposed framework would make leaps in resolving the major pain points identified in Sect. 2.2. The remainder of Sect. 3 envisions our new framework through these key principles, and explains how each of the major pain points would be addressed. Though not the focus of this paper, we also developed a set of nonfunctional requirements and UML class diagrams for our proposed framework; these are available online[2].

3.1 Versatility and Extensibility

We propose a highly modular design suitable for implementation in any object-oriented programming language. In particular, our design supports any industry-standard benchmark or micro-benchmark. These could be implemented natively within our framework or run as separate programs. The minimum implementation required to run an existing benchmark would be writing methods to launch the existing benchmark's executable, process its output, and handle its input configuration parameters. The framework would similarly support any possible target DBMS, either implemented natively or accessed via a separately-running benchmark program.

Different experimental methods and repeatability (PP3) would be supported by abstracting the concept of a benchmark from that of an experiment. In our model, an experiment could use any benchmark or combination thereof, any number of times, with any number of warm-up/cool-down period, termination, and data collection triggers. This would allow expressing complex experiments such as the hypothetical scenario in Table 2.

Since our framework would be capable of handling the whole benchmarking pipeline, the need to write and debug custom testbed scripts (PP1–PP4) would be completely eliminated. Experiments designed within our framework could be easily ported to new scenarios without the traditional script-modifying and re-debugging that would otherwise be required with a custom testbed. Our framework would also provide new opportunities for collaboration and data sharing because anyone who could run the framework could also load and explore the output of any experiment performed using it.

3.2 Configuration

Simplifying configuration of benchmarks and DBMSes would be a significant usability accomplishment. Many systems are configured by setting values for a set of predefined configuration keys, often using a key-value configuration format such as Java `Properties` files. We would take advantage of this commonality

[2] https://github.com/lexibrent/benchfw-resources/.

Table 2. Hypothetical YCSB benchmarking experiment in our framework.

Benchmark config	`ycsb/workloads/workloada`
Benchmark clients	localhost
DBMS servers	bench1
Method	5 trials, non-distributed
Vary	YCSB thread count from 1 to 32, stepping by 1
Targets	MongoDB
Warm-up until	server disk I/O is stable
Output	YCSB throughput and aggregates: avg, SD, min, max
Output formats	`CSV` and `JSON`
Collect (from servers)	disk, CPU, RAM, and network utilization every 2 s
Start condition	start at `2018-02-10 00:00:00 UTC`

with a simple self-documenting key-value configuration format for every aspect of the framework. Our framework could automatically generate configuration files for other software components based on values set in the framework's own configuration, provided the framework is first extended with an implementation of the appropriate parse and generation logic (for non-key-value formats).

Any configuration file within our proposed framework can reference any other configuration file, allowing large or complicated scenarios to be split into manageable chunks. The need to copy entire configuration files to change options between experimental runs (PP1) is eliminated because our framework would allow all desired values to be expressed within the same configuration file using, for example, a concise range syntax. These value set declarations would be processed independently to the general configuration syntax, allowing custom syntax and parse logic to "just work" when implemented as an extension.

In addition, settings could be configured at different granularity levels. In order of granularity from course to fine, these would be: framework, DBMS, benchmark, experiment, and run. Any key configured at a given granularity level would override any values set for it in courser granularity levels. This design would allow, e.g., multiple experiments to share a common base configuration, with different sets of benchmark runs overriding specific configuration values.

Since flexibility is a major goal of our proposed framework, every component would be extensively configurable. Resource-intensive components such as comprehensive real-time system monitoring could be readily disabled. This design provides finer control of overhead and the associated trade-offs.

3.3 Distributed Benchmarks

Challenges associated with manually coordinating distributed benchmark clients running in parallel (PP5) would be eliminated by our framework's ability to automatically configure and coordinate multiple DBMS server systems and benchmark clients. An experiment configuration could list multiple machines on the same network that are all running the framework, partitioned into DBMS server and benchmark client machines. If any machine in such a configuration contains a more recent version of an experiment configuration than its peers, all peers would update to the later version before beginning execution. This would enable painless configuration modifications post-deployment, by simply modifying the configuration stored on one of the connected machines.

Partitioning the benchmark workload could be accomplished by specializing specific configuration values for each client or group of clients. For example, to split the workload based on operation type, one group of clients could be configured to perform only reads and another to perform only inserts and updates. Any configuration options supported by the benchmark could be used, so other examples may include partitioning based on operation count, primary key, table, or database (for multi-tenant benchmarks).

Our framework would track time stamps for all captured events, including error log entries and output from the benchmark and target DBMS. At the conclusion of a distributed benchmark run, all machines running the framework would collate these events based on times tamp, while still tracking which machine captured each event. Correlation coefficients between different datasets could be automatically computed for collected metrics, assisting the user with post-experiment analysis and debugging unexpected results.

3.4 Repeatability/Reproducibility and Debugging

Despite being a core principle of the scientific method, reproducibility is often overlooked by the database community. We speculate that this may be due to factors such as experimental complexity, inability to replicate hardware configurations, closed-source or proprietary software licences, and incomplete or imprecise descriptions of experiments in literature. Our proposed framework would simplify reproducibility (PP1, PP3) for benchmarking by automatically configuring the experimental method, benchmark, and DBMS. Each DBMS under test would simply need a corresponding adapter class implementation.

Using our framework, anyone with the necessary hardware and software environment could replicate any previous experiment performed with the framework by running a command and providing the output data from the experiment to be replicated. Our framework would warn the user if any detectable differences were found between the current environment and the environment used in the original experiment, reducing the chance of small differences going unnoticed. If the full output of a previous experiment is unavailable, the experiment could still be replicated by using identical configuration files, in which case our framework would be unable to report environmental differences. During replication of an experiment, values that were originally sampled from statistical distributions could either be re-sampled or re-used.

The comprehensive targeted metadata collection capabilities of our framework (described in Sect. 3.5) would simplify debugging by providing a more configurable level of detail than what is traditionally available. Debugging tools could also be attached to processes within the pipeline by setting appropriate configuration keys. This would eliminate the need to repeat an experiment with debugging tools manually attached, making heisenbugs easier to catch (PP4).

3.5 Metadata Collection

The framework's general approach to environment metadata collection would be that "more is better" provided it can be used effectively. So, it should be traceable, comprehensive, and relevant. Environment information would be collected for each system running our framework within an experiment. It would include details of (for example) the kernel, operating system, hardware, resource use, runtime and library versions, network connections, running processes and threads, system calls, memory access violations, crash dumps, and exact configuration files used. The collection mechanism would be modular and extensible, so additional data collection could be implemented with ease.

Traceability of collected metadata would be accomplished with a relational model linking each piece of information to its origin and other related data. For example, system environment information is related to the: machine on which it was collected, benchmark being executed, configuration used in that benchmark execution, experimental method used and its configuration, framework version, etc. These proposed metadata collection capabilities offer significant usability and efficiency improvements over typical benchmarking methodologies (PP6).

3.6 Output and Analysis

Industry-standard DBMS benchmarks output a variety of different formats, only some of which are extensible and configurable. Many write directly to `stdout`, relying on the user to manually perform collection and analysis (or automate it with custom testbed scripts). Our framework would improve this (PP2, PP4) by automatically extracting metrics of interest from raw benchmark output and storing these internally so they can be exported to any desired format such as `CSV`, `JSON`, or perhaps a plot. For each supported benchmark, our framework would require logic to process the benchmark's raw output; typically involving running regular expressions or parsers over captured `stdout`. Each output format would have its own writer implementation, so it would be straightforward to extend the framework with support for new output formats.

Comprehensive data collection in our proposed framework would make it possible to extract additional metrics from existing experimental output without needing to re-run the benchmark itself. This could also be used to convert a completed experiment's data to a different output format.

Statistical analysis of experimental results is traditionally performed entirely manually after data extraction (PP8). Our framework would improve the efficiency of this process by automating some common calculations. For example, the user could configure the framework to output aggregate metrics such as sum, average, min, max, median, standard deviation, variance, confidence interval, and correlation coefficients (PP7). As with the other aspects of our framework, these would be modular and extensible. The framework would also be capable of reporting new correlations that may be of interest to the experimenter.

4 Future Work

We now need to implement and evaluate our proposed framework. Future work could also explore developing a framework supporting both synthetic and trace-based workloads; combining the framework described in this paper with the functionality of BenchFoundry [3].

5 Conclusion

In this paper we proposed a new versatile and extensible framework for conducting benchmarking of DBMSes, based on a survey of the benchmarking practices of several individuals in the database community from both industry and academia. We showed that a typical benchmarking workflow is well-modelled as a pipeline of three key processes. Based on interview responses, we developed a set of major benchmarking "pain points" and mapped these onto the processes in our pipeline to determine which processes potentiated the greatest overall efficiency and usability improvements. We characterized several core principles upon which our vision is based: extensibility, usability, configurability, extensive data collection, and reproducibility. Our proposed framework was described, including how it would address each of the major pain points we had identified. A future implementation of our proposed framework could greatly improve the coherence of benchmarking for industry and academic purposes.

References

1. Ameri, P., Schlitter, N., Mayer, J., Streit, A.: NoWog: a workload generator for database performance benchmarking. In: 2016 IEEE 14th International Conference on Dependable, Autonomic and Secure Computing, 14th International Conference on Pervasive Intelligence and Computing, 2nd International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress, DASC/PiCom/DataCom/CyberSciTech 2016, Auckland, New Zealand, 8–12 August 2016, pp. 666–673 (2016)
2. Barahmand, S., Ghandeharizadeh, S.: D-Zipfian: a decentralized implementation of Zipfian. In: Proceedings of the Sixth International Workshop on Testing Database Systems, DBTest 2013, pp. 6:1–6:6. ACM, New York (2013)
3. Bermbach, D., Kuhlenkamp, J., Dey, A., Ramachandran, A., Fekete, A., Tai, S.: BenchFoundry: a benchmarking framework for cloud storage services. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10601, pp. 314–330. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69035-3_22
4. Bermbach, D., Kuhlenkamp, J., Dey, A., Sakr, S., Nambiar, R.: Towards an extensible middleware for database benchmarking. In: Nambiar, R., Poess, M. (eds.) TPCTC 2014. LNCS, vol. 8904, pp. 82–96. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15350-6_6

5. Bermbach, D., Wittern, E., Tai, S.: Cloud Service Benchmarking. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55483-9
6. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, pp. 143–154. ACM, New York (2010)
7. Dey, A., Fekete, A., Nambiar, R., Rohm, U.: YCSB+T: benchmarking web-scale transactional databases. In: Proceedings - International Conference on Data Engineering, pp. 223–230 (2014)
8. Difallah, D., Pavlo, A.: OLTP-bench: an extensible testbed for benchmarking relational databases. Proc. VLDB Endow. **7**(4), 277–288 (2013)
9. Ghazal, A., et al.: BigBench: towards an industry standard benchmark for big data analytics. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, 22–27 June 2013, pp. 1197–1208 (2013). https://doi.acm.org/10.1145/2463676.2463712
10. Hoag, J.E., Thompson, C.W.: A parallel general-purpose synthetic data generator. SIGMOD Rec. **36**(1), 19–24 (2007)
11. Lu, J.: Towards benchmarking multi-model databases. In: 8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, 8–11 January 2017, Online Proceedings (2017)
12. 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). https://doi.org/10.1007/978-3-642-18206-8_4
13. Rabl, T., Poess, M., Danisch, M., Jacobsen, H.A.: Rapid development of data generators using meta generators in PDGF. In: Proceedings of the Sixth International Workshop on Testing Database Systems, DBTest 2013, pp. 5:1–5:6. ACM, New York (2013)
14. 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). https://doi.org/10.1007/978-3-642-18206-8_2
15. Seybold, D.: Towards a framework for orchestrated distributed database evaluation in the cloud. In: Proceedings of the 18th Doctoral Symposium of the 18th International Middleware Conference, Middleware 2017, pp. 13–14. ACM, New York (2017)
16. Stephens, J.M., Poess, M.: MUDD: a multi-dimensional data generator. SIGSOFT Softw. Eng. Notes **29**(1), 104–109 (2004)
17. Transaction Processing Performance Council (TPC): TPC-Homepage V5 (2016). http://www.tpc.org/
18. Van Aken, D., Difallah, D.E., Pavlo, A., Curino, C., Cudré-Mauroux, P.: Bench-Press: dynamic workload control in the OLTP-bench testbed. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD 2015, pp. 1069–1073. ACM, New York (2015)
19. Yoon, D.D.Y.: Database Performance Evaluation Framework. Ph.D. thesis, The University of Sydney (2008)
20. van der Zijden, W., Hiemstra, D., van Keulen, M.: MTCB: a multi-tenant customizable database benchmark. In: Proceedings of the 9th International Conference on Information Management and Engineering, ICIME 2017, pp. 17–23. ACM, New York (2017)