Technical Perspective
DIAMetrics: Benchmarking Query Engines at Scale

Peter Boncz
CWI, The Netherlands
boncz@cwi.nl

Benchmarking database systems has a long and successful history in making industrial database systems comparable, and is also a cornerstone of quantifiable experimental data systems research. Creating good benchmarks has been described as something of an art [3]. One can inspire dataset and workload design from "representative" use cases queries, typically informed by domain experts; but also exploit technical insights from database architects in what features, operations, and data distributions should come together in order to invoke a particularly challenging task[1].

While this methodology has served the database community well by creating reference points such as TPC-C and TPC-DS, even in a way creating the narrative on what database workloads are (i.e. transactional vs. analytical), such synthetic benchmarks typically fail to represent actual workloads, which are more complex and thus hard to understand, mixed in nature, and constantly changing. *Automatic benchmark extraction* from real-life workloads therefore provides a powerful pathway towards quantifying database system performance that matters most for an organization, though this requires techniques for summarizing query logs into more compact workloads, and for extracting real data and anonymizing it to create benchmark datasets.

Also, benchmarking in the cloud age needs to deal with complex set-ups in dynamically provisioned environments where distributed compute, storage and network resources need to be orchestrated throughout the benchmarking workflow of dataset creation, loading, workload execution, performance measurement and correctness checking. PEEL[1] has been an earlier effort to facilitate such distributed benchmarking. However, it lacked automatic workload extraction as it focused on executing synthetic benchmarks on Hadoop-based distributed systems running either Spark or Flink.

This paper on DIAMetrics from Deep et al., describes a versatile framework, developed in Google, for automatic extraction of benchmarks and their distributed execution and performance monitoring. The structure of the system is clever, particularly having the components run separately and allowing multiple entry-points into the system to cater to different use cases.

[1]This was coined "choke point"-guided benchmark design by the Linked Data Benchmark Council (ldbcouncil.org)

The observation that input data is not necessarily under the query engines' control, and storage formats may vary, is important. Classic benchmarks assume the engine can preprocess data at will and allows access to detailed statistics about the dataset, which is often not possible in practice.

The state-of-the-art workload extraction approach in DIAMetrics takes into account both structural query features as well as actual execution metrics. This algorithm is described in a separate publication [2].

The data scrambler is a very desirable component, but the paper is light on details here, raising some questions. Certain information leakage is bound to occur, limiting safe use cases. The technique of removing correlations between columns, will also remove important query optimization challenges. This crucial aspect of automatic benchmark extraction seems a fruitful area of future research, e.g. into formal privacy-preservation bounds or privacy-respecting preservation of certain correlations.

At Google, the DIAMetrics framework has proven itself useful for database developers, for performance monitoring and regression testing, as engines evolve. Database users also benefit, using the system to find out which of the multiple Google engines (e.g. F1, Procella, Dremel) best suits their use case, but also to provide performance accountability: to identify and communicate performance problems.

This paper is, in short, a highly recommended read. It inspired me to think about novel directions, possibly taking automatic benchmark extraction from performance monitoring accountability also towards correctness testing: one could envision enriching workload summarization with new dimensions such as code coverage [5] and automatic generation of query correctness oracles [4].

1. REFERENCES

[1] C. Boden, A. Alexandrov, A. Kunft, T. Rabl, and V. Markl. PEEL: A framework for benchmarking distributed systems and algorithms. In *TPCTC 2017*.

[2] S. Deep, A. Gruenheid, P. Koutris, J. F. Naughton, and S. Viglas. Comprehensive and efficient workload compression. *PVLDB*, 14(3):418–430, 2020.

[3] K. Huppler. The art of building a good benchmark. In *TPCTC 2009*.

[4] M. Rigger and Z. Su. Finding bugs in database systems via query partitioning. In *OOPSLA*. ACM, 2020.

[5] R. Zhong, Y. Chen, H. Hu, H. Zhang, W. Lee, and D. Wu. SQUIRREL: testing database management systems with language validity and coverage feedback. In *CCS*, pages 955–970. ACM, 2020.