Buzzy: Towards Realistic DBMS Benchmarking via Tailored, Representative, Synthetic Workloads

Vision Paper

Jörg Domaschka, Mark Leznik, Daniel Seybold
Ulm University
Ulm, Germany
first.last@uni-ulm.de

Simon Eismann, Johannes Grohmann, Samuel Kounev
University of Würzburg
Würzburg, Germany
first.last@uni-wuerzburg.de

ABSTRACT

Distributed Database Management Systems (DBMS) are a crucial component of modern IT applications. Understanding their performance and non-functional properties is of paramount importance.

Yet, benchmarking distributed DBMS has proven to be difficult in practice. Either, a realistic workload is often mapped to a synthetic workload without knowing if this mapping is correct or available workload traces are replayed. While the latter approach provides more realistic results, real-world traces are hard to obtain and their scope is limited in time scale and variance.

We propose collecting real-world traces and then applying data generation techniques to synthesize similar realistic traces based on it. Based in this approach, we can obtain workloads for benchmarking, exhibit variability with respect to different aspects of interest while still being similar to the original traces. Varying generation parameters, we are able to support benchmarking what-if scenarios with hypothetical workloads and introduced anomalies.

CCS CONCEPTS

• **Information systems → Database performance evaluation**.

KEYWORDS

Measurement-based Performance Evaluation, Data Synthesis, Database Management Systems

ACM Reference Format:
Jörg Domaschka, Mark Leznik, Daniel Seybold
and Simon Eismann, Johannes Grohmann,
Samuel Kounev. 2021. Buzzy: Towards Realistic DBMS Benchmarking via Tailored, Representative, Synthetic Workloads: Vision Paper. In *Companion of the 2021 ACM/SPEC International Conference on Performance Engineering (ICPE '21 Companion), April 19–23, 2021, Virtual Event, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3447545.3451175

1 INTRODUCTION

Modern Database Management Systems (DBMS) are the backbone of many applications in various domains (IoT, smart cities, Big Data, etc.) [1, 9]. Hence, their continuous, seamless operation within predefined non-functional boundaries is of utmost importance.

Due to the large configuration space for distributed DBMS, there is no one-size-fits-all configuration suiting all possible situations [26]. Instead, literature shows that the experienced quality of service with respect to all non-functional aspects heavily depends on the type of workload applied to a DBMS [2, 8, 11]. Hence, choosing and configuring a DBMS for a dedicated application scenario requires insight into the expected workload as well as running carefully crafted evaluation tests using a representative workload. Yet, the majority of existing benchmarks uses static or only slightly configurable workloads against a DBMS and it is up to a performance engineer to ensure that the evaluation workload matches the expected workload [23]. This task only increases in complexity with a more diverse real-world workload. On the other hand, trace-based benchmarking captures real-world system behavior in the form of execution traces and replays the traces against the system under test (SUT) [6, 22]. Here, we face the challenges that real-world traces are finite in length (usually rather short than long), hard to obtain (and even harder to share) [5], and usually do not contain sufficient anomalous behavior to assess system performance under abnormal circumstances [28]. This makes it hard to evaluate non-functional properties based on multiple representative workloads and perform realistic what-if analysis.

This paper envisions Buzzy to address these shortcomings: Buzzy centers around the concept of classifying and synthesizing arbitrary amounts of workload data from real-world DBMS traces. We base our work on established concepts for synthetic data generation using artificial neural networks applied to time-series oriented traces from DBMS workloads. This allows us to generate artificial but representative synthetic workload traces increasing the size of available data sets and replacing sensitive data with artificial data.

Clustering traces and knowing cluster properties combined with the ability to generate data also enables us to additionally augment imbalanced classes in our original data set. Finally, synthesizing representative workloads enables us to enhance the result with different types of anomalies.

2 RELATED WORK

2.1 DBMS Benchmarking

A *DBMS benchmark* generates a *workload* which is applied to the DBMS under test. Further, it provides features such as metric reporting, metric processing, and the coordinated execution of multiple

ICPE '21 Companion, April 19–23, 2021, Virtual Event, France

ACM ISBN 978-1-4503-8331-8/21/04.
https://doi.org/10.1145/3447545.3451175

workloads [5]. *DBMS workloads* consist of a sequence of operations issued on a DBMS. The workload defines the executed types of operations as well as access patterns. The operations may be processed by time or by sequence. In the former case, each operation is assigned a timestamp with respect to the first operation. In the latter case, an operation is performed once its predecessor has been completed. Scheduling the parallel execution of workload sequences is the responsibility of the encapsulating benchmark. Workloads are either based on traces or on synthetic data modeled after real-world load patterns and generated under a set of configurable constraints [6]. A single DBMS benchmark can support multiple workload types [11].

With the continuously evolving DBMS landscape [9] a multitude of NoSQL DBMS benchmarks have been established over the last decade [13, 23]. The workloads issued by these benchmarks range from basic CRUD operations (e.g., YCSB [8]) to more advanced data model-specific workloads(e.g. NoWog [2]). These benchmarks aim at emulating the workloads of realistic applications by specifying synthetic workloads based on a set of configurable constraints such as request distribution or data set size. Yet, they do not support trace-based workload generation by replaying real-world queries nor do they consider varying workloads as caused by seasonality. Trace-based benchmarks, in contrast, clearly increase the significance of benchmark results [6, 15]. Therefore, trace-based benchmarks are preferable for DBMS benchmarking. Due to their additional technical complexity [6], trace-based DBMS benchmarks are rare and so far only support relational DBMS [5, 11].

2.2 Trace-based Benchmarking

Trace-based benchmarking requires modelling the concrete user behavior and the workload intensity. It is common in many areas, such as mobile applications [16, 22], large-scale analytics [18, 29], and web applications [16, 24]. In order to model the concrete user behaviour multiple authors propose graph-based workload models that capture the behavior and tasks triggered by different types of users [4, 27]. Others partition workloads according to the user types, and then sample workload traces for each user type to capture the user behavior [30]. Regarding workload intensity, most approaches try to capture it using statistical distributions [12, 21], while others focus on extracting the statistical properties of a workload trace into a descriptive model in order to reproduce similar traces with the same statistical characteristics [28].

2.3 Artificial Time-Series Generation

Current data synthesis breakthroughs and the research field in general are driven by *generative adversarial networks* (GAN), which were introduced by Goodfellow et al. [14]. GANs are a type of artificial neural network architecture where two networks, a generator, and a discriminator compete in a mini-max game. Hereby, the generator's task is to learn the fidelity and distribution of the underlying ground truth data and effectively "fool" the discriminator into accepting fake data as real. The discriminator, in turn, learns to distinguish real samples from fake samples based on the original ground truth data. Prominent GAN examples include StyleGANs 1 and 2 [17] able to synthesize realistic images of human faces based on high-level attributes (head pose, identity). GANs have also shown success in domains outside the image synthesis domain. For example, Delaney et al. [10] use GANs to synthesize multidimensional ECG data, albeit image-oriented convolutional neural networks (CNN).

3 THE BUZZY APPROACH

This section describes the overall vision followed by Buzzy. We introduce conceptual, technical, and research challenges, and sketch our work plan to tackle these challenges.

3.1 Vision

We envision implementing Buzzy as a multi-step process sketched in Figure 1. The goal is to collect workload traces from a production system and then generate representative, synthetic workload traces. In that respect, a *workload trace* is a time-stamped sequence of operations issued against a DBMS; a *representative, synthetic workload* as created by Buzzy is a synthetic workload in the sense that it is artificially generated. In contrast to classical synthetic workloads like those from TPC that aim at representing a larger class of applications, our generated workload is specifically tailored to match the patterns observed in the original real-life workload it is based on. Buzzy's steps are detailed in the following sections.

Data collection. This step takes care of obtaining traces out of a production DBMS using established probes or newly developed tools. It collects the necessary metadata needed by later steps. Such metadata may include information about the content of the DBMS.

Preprocessing and clustering. This step processes the collected data. This includes data curation (i.e., dropping data with insufficient quality) but also extrapolating missing values. Furthermore, this step performs clustering to identify, for example different user groups with different usage patterns. The resulting *observed workload clusters* are then stored immutably.

Data generation. This step generates representative, synthetic workload traces for each of the observed workload clusters. It is based on a specification defining for which of the clusters to synthesize (e.g., for a specific user group), and the amount of data to generate. This step contains large intrinsic complexity as it requires a full train-test-validate cycle for the applied machine learning mechanisms. The resulting *synthetic workload clusters* are again stored.

Anomaly enhancement. This optional step enhances the data generation process. It enriches the synthesized data with anomalies either found in the observed workload traces (e.g., querying invalid data items or sending corrupt queries) or purely artificial (creating volume/date spikes [7]). This step also contains a specification defining which anomalies to create, how many of them, etc.

Composition. This step selects some of the generated traces and combines them to a *synthetic workload trace* ready for benchmarking. The composition may be a simple replay of a representative, synthetic workload trace, aiming at benchmarking the DBMS with variations of the current workload. A composition may also create a hypothetical workload for running what-if analyses. These may combine multiple synthetic traces from various workload clusters, e.g., in order to evaluate the system behavior when allowing

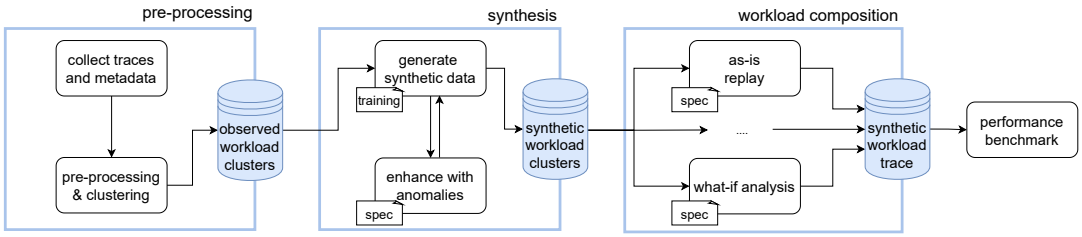

Figure 1: Overview of Buzzy

more users on the system; it may also contain so-far unexperienced anomalies to evaluate the system resilience.

Benchmark execution. This step takes care of executing the workload on the SUT. During a benchmark run, a scheduler retrieves entries from the trace and submits them in a semantically correct and timely manner [5] (e.g., inserts have been completed before reads). Integrating the benchmark in higher-level evaluation frameworks like Mowgli [25] even supports higher-level evaluation objectives including scalability, elasticity or availability.

3.2 Challenges and Approach

Realizing the outlined vision requires overcoming severe challenges in almost all steps except for the benchmark execution step which is well understood [5, 11, 15]. This section summarises the challenges and sketches our initial plans to address them.

Data collection and traces. Buzzy relies on the collection of realistic and detailed traces from real-world DBMS installations. Ideally, collecting this information is non-intrusive to the client applications (with minimal impact on the quality of service experienced by users) and further does not impact the performance of the DBMS. It is currently unclear what level of detail is needed for the traces in order to generate realistic, synthetic workloads. Also, the relevant data to construct meaningful traces is currently unknown just as the amount and required length of the observed workload traces.

Many DBMS provide high-level metrics out-of-the-box, e.g., read-write ratio. However, such traces do not provide many details. In contrast, some relational DBMS provide tools to record fine-grained traces at the level of SQL statements. The situation is less clear for distributed DBMS, particularly NoSQL DBMS.

Approach: In order to keep the complexity of the synthesis process under control, we will initially work with NoSQL DBMS and exclusively focus on CRUD operations. Here, we plan to move from single-instance to distributed deployments. This is due to the fact that data-intensive application components are commonly operated in (geo)-distributed environments and DBMS workloads are composed of heterogeneous DBMS request patterns created by distributed client applications [1]. Later, we also plan to move towards more complex NoSQL queries [23]. SQL-based and NewSQL DBMS are currently not on our roadmap, but may be added later on. This does not weaken our approach, as NoSQL DBMS have become a preferred storage backend for data-intensive applications [9].

Regarding trace granularity, we plan to start working with fine-grained traces providing as much detail as possible. Besides a timestamp, such fine-grained traces should contain the actual operation and ideally data on the client issuing the operation. From that point, we plan to gradually reduce the details used for clustering and synthesis. This allows us to identify (potentially multiple) intersections between an optimal outcome and the details of the traces.

To the best of our knowledge, there is no DBMS-independent trace recording tool available. Therefore, we initially plan to rely on an HTTP tracing tool such as Jaeger[1], for a NoSQL DBMS with an HTTP(S) interface.

System state. The performance of a stateful system not only depends on the current workload but also on the short-term and long-term past. More specifically, a read operation may fail if the item has not been stored before, or it may complete faster if all data is in memory. This depends on the history of operations as well as local and global read patterns.

Approach: Using long-enough traces and a warm-up phase per benchmark rules out the impact of caching provided that the benchmarking environment allows for similar specifications. Collecting metadata on the DBMS (data distribution, data per collection, etc.) enables the workload generation process to synthesize a meaningful prefilling of the system under test.

Synthesis. The generation process is an essential part of Buzzy. While related work shows that GANs are well suited for generating images and numeric time-series data, their wide applicability to non-numeric time-series data still remains to be shown. Furthermore, the number of data points that a GAN is able to generate depends on the available (GPGPU) memory.

Further, the question of data validity needs to be addressed: In particular, for scenarios that go beyond CRUD operations (e.g., scan and search), the synthesizer not only needs to learn the distribution of operations, but also the types and structures of data items. But even when only CRUD operations are used, the size and internal structure of the different data items may impact results.

Approach: For Buzzy, we plan to utilize GANs for workload synthesis. While there are other viable alternatives, such as variational autoencoders [3], our earlier work [19, 20] shows that GANs are able to synthesize class specific time-series data. This gives us confidence in the potential of GANs and we expect them to be able to learn the correct fidelity of the underlying original data and reflect characteristic (statistical) properties. Early work on anomaly synthetization using conditional GANs shows promising results.

Regarding the structure of data items, we initially avoid this problem, as we purely focus on CRUD operations and use data items with a fixed size and structure. Later, we plan to use statistical

[1] https://www.jaegertracing.io/

methods to generate suited data items. In a final step, we plan to apply GAN-based synthesis to data items as well.

Query semantics. Not all sequences of DBMS commands issued against a DBMS are valid (in the sense that user-level protocols are violated) or meaningful (in the sense that for example only benchmarking requests for non-existent items do not provide any insights). Hence, Buzzy needs to consider both operational semantics and also benchmarking semantics.
Approach: We expect that the GAN used learns the correct query semantics and that they do not have to be hard coded. This allows us to apply our approach to different DBMS using different APIs.

Evaluation and validation. With respect to evaluation and validation, Buzzy needs to address multiple issues: (i) define a metric describing the representativeness of the generated synthetic traces; (ii) measure the similarity between observed and generated traces providing means to quantify the variability between multiple synthetic workloads; (iii) measure and evaluate the variance between the various results when benchmarking a set of synthetic workloads; (iv) measure and quantify the similarity of benchmark results for observed and generated traces.

Yet, none of the above questions can be answered before having initial solutions for the previous challenges.

Specifications and configurations. It is unclear how to describe and define the anomalies that should be added to a generated synthetic workload. It is further unclear how to specify a meaningful and trustable what-if analysis and how to visualize the results. As validation, this challenge requires initial results for the other challenges.

4 CONCLUSION

This paper presented Buzzy, our vision towards classifying and synthesizing arbitrary amounts of workload data from real-world DBMS traces. Doing so, Buzzy is capable of generating representative, synthetic workloads tailored for a particular DBMS installation. These realistic and representative but not identical workloads allow benchmarking DBMS installations in a realistic manner. Dynamically recomposing and scaling a generated workload enables us to further evaluate meaningful what-if scenarios. Due to that Buzzy is able to enhance synthetic workloads with anomalies.

Work on this vision has just started by looking for partners willing to provide/collect workload traces. Once traces are available, we aim at first realizing the full process of collection, synthesis, and workload composition for a single DBMS. Afterwards, we plan to investigate the generalizability and portability of Buzzy aiming at transferring all mechanisms to a different DBMS or even different types of DBMS. Finally, we extend the presented methodology to not only cover workload traces but also include other notable events such as node failures, networks problems, or disk degradation.

REFERENCES

[1] D. Abadi, A. Ailamaki, D. Andersen, P. Bailis, M. Balazinska, P. Bernstein, P. Boncz, S. Chaudhuri, and et al. 2020. The Seattle Report on Database Research. *SIGMOD Rec.* 48, 4 (2020), 44–53.

[2] P. Ameri, N. Schlitter, J. Meyer, and A. Streit. 2016. NoWog: A Workload Generator for Database Performance Benchmarking. In *2016 IEEE DASC/PiCom/DataCom/CyberSciTech*. 666–673.

[3] Wei Bao, Jun Yue, and Yulei Rao. 2017. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS one* 12, 7 (2017), e0180944.

[4] Aaron Beitch, B. Liu, Timothy Yung, R. Griffith, A. Fox, and D. Patterson. 2010. Rain: A Workload Generation Toolkit for Cloud Computing Applications.

[5] 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*.

[6] David Bermbach, Erik Wittern, and Stefan Tai. 2017. *Cloud service benchmarking*.

[7] Peter Bodik, Armando Fox, Michael J. Franklin, Michael I. Jordan, and David A. Patterson. 2010. Characterizing, Modeling, and Generating Workload Spikes for Stateful Services. In *Proceedings of the 1st ACM Symposium on Cloud Computing*.

[8] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*. 143–154.

[9] Alejandro Corbellini, Cristian Mateos, Alejandro Zunino, Daniela Godoy, and Silvia Schiaffino. 2017. Persisting big-data: The NoSQL landscape. *Information Systems* 63 (2017), 1 – 23.

[10] A. Delaney, E. Brophy, and T. Ward. 2019. Synthesis of Realistic ECG using Generative Adversarial Networks. *arXiv preprint arXiv:1909.09150* (2019).

[11] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudre-Mauroux. 2013. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *Proc. VLDB Endow.* 7, 4 (Dec. 2013), 277–288.

[12] Dror G. Feitelson. 2002. Workload Modeling for Performance Evaluation. In *Performance Evaluation of Complex Systems: Techniques and Tools*. 114–141.

[13] Steffen Friedrich, Wolfram Wingerath, Felix Gessert, and Norbert Ritter. 2014. Nosql OLTP benchmarking: A survey. In *Informatik 2014*. 693–704.

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *Advances in neural information processing systems* 27 (2014), 2672–2680.

[15] Jim Gray. 1992. *Benchmark Handbook: For Database and Transaction Processing Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[16] Y. Huang, Z. Zha, M. Chen, and L. Zhang. 2014. Moby: A mobile benchmark suite for architectural simulators. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 45–54.

[17] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. 2020. Analyzing and Improving the Image Quality of StyleGAN. In *Proc. CVPR*.

[18] K. Kim, K. Jeon, H. Han, S. Kim, H. Jung, and H. Y. Yeom. 2008. MRBench: A Benchmark for MapReduce Framework. In *2008 14th IEEE International Conference on Parallel and Distributed Systems*. 11–18.

[19] Thang Le Duc, Mark Leznik, Jörg Domaschka, and Per-Olov Östberg. 2020. Workload Diffusion Modeling for Distributed Applications in Fog/Edge Computing Environments. In *ACM/SPEC International Conference on Performance Engineering (ICPE '20)*. ACM, 218–229.

[20] Mark Leznik, Patrick Michalsky, Benjamin Schanzel, Peter Willis, Per-Olov Östberg, and Jörg Domaschka. 2021. Multivariate Time Series Synthesis Using Generative Adversarial Networks. In *Proceedings of the 2021 ACM/SPEC International Conference on Performance Engineering*.

[21] H. Li. 2010. Realistic Workload Modeling and Its Performance Impacts in Large-Scale eScience Grids. *IEEE Transactions on Parallel and Distributed Systems* (2010).

[22] Alexander Lochmann, Fabian Bruckner, and Olaf Spinczyk. 2017. Reproducible Load Tests for Android Systems with Trace-Based Benchmarks. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. 73–76.

[23] 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*. 107–112.

[24] G. Ruffo, R. Schifanella, M. Sereno, and R. Politi. 2004. WALTy: a user behavior tailored tool for evaluating Web application performance. In *Third IEEE International Symposium on Network Computing and Applications, (NCA)*. 77–86.

[25] Daniel Seybold, Moritz Keppler, Daniel Gründler, and Jörg Domaschka. 2019. Mowgli: Finding Your Way in the DBMS Jungle. In *ACM/SPEC International Conference on Performance Engineering (ICPE '19)*. ACM, 321–332.

[26] Michael Stonebraker and Uğur Çetintemel. 2018. *"One Size Fits All": An Idea Whose Time Has Come and Gone.* ACM and Morgan and Claypool, 441–462.

[27] André van Hoorn, Matthias Rohr, and Wilhelm Hasselbring. 2008. Generating Probabilistic and Intensity-Varying Workload for Web-Based Software Systems. In *Performance Evaluation: Metrics, Models and Benchmarks*. 124–143.

[28] Jóakim von Kistowski, Nikolas Herbst, Samuel Kounev, Henning Groenda, Christian Stier, and Sebastian Lehrig. 2017. Modeling and Extracting Load Intensity Profiles. *ACM Transactions on Autonomous and Adaptive Systems* 11, 4 (2017).

[29] X. Wu, V. Deshpande, and F. Mueller. 2012. ScalaBenchGen: Auto-Generation of Communication Benchmarks Traces. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. 1250–1260.

[30] Netanel Zakay and Dror G. Feitelson. 2013. Workload Resampling for Performance Evaluation of Parallel Job Schedulers. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE)*. 149–160.