Enabling Containerized, Parametric and Distributed Database Deployment and Benchmarking as a Service

George Kousiouris
Dept. of Informatics & Telematics, Harokopio University of Athens
9, Omirou Str. 177 78, Tavros, Greece
gkousiou@hua.gr

Dimosthenis Kyriazis
Dept. of Digital Systems, University of Piraeus
Karaoli & A. Dimitriou 80, 18534 Piraeus, Greece
dimos@unipi.gr

ABSTRACT

Containerized environments introduce a set of performance challenges that require extensive measurements and benchmarking to identify and model application behavior regarding a variety of parameters. Databases present extra challenges given their extensive need for synchronization and orchestration of a benchmark run, especially in microservice-oriented technologies (such as container platforms) and dynamic business models such as DBaaS. In this work we describe the adaptation of our open source, baseline load injection as a service tool, Flexibench, in order to enable the automated, parametric launching and measurement of containerized and distributed databases as a service. Adaptation and synchronization needs are described for ensuring test sequence and applied through a case study on MySQL. Therefore a performance engineer can directly test selected configuration and performance of a database in a given target workload with simple REST invocations. Experimentation starts from adapting the official MySQL docker images as well as OLTP Bench Client ones and investigates scenarios such as parameter sweep experiments and co-allocation scenarios where multiple DB instances are sharing physical nodes, as expected in the DBaaS paradigm.

CCS Concepts

• Information systems → Database management systems • General and reference → Cross-computing tools and techniques→ Measurement • General and reference → Cross-computing tools and techniques → Performance

Keywords

Container Platforms; Benchmarking as a Service; Databases

ACM Reference format:

George Kousiouris and Dimosthenis Kyriazis, 2021, Enabling Containerized, Parametric and Distributed Database Deployment and Benchmarking as a Service. In *Companion of the 2021 ACM/SPEC Int'l Conf. on Performance Engineering, April 19-23, 2021, Virtual Event, France.* ACM, NY, NY, USA. 4 pages https://doi.org/10.1145/3447545.3451188

1. INTRODUCTION

Container environments have attracted significant attention in recent years due to the ease of management, advanced packaging and ability to rapidly update, scale and in general manage the respective applications. However, adding an extra layer of virtualization typically introduces further performance delays[1] and needs for extensive benchmarking and measurement of the target application in the new system, for a variety of setups.

Especially when targeting database configurations, a critical aspect of this process includes the extensive need for automation and synchronization of operation sequence, in order to ensure a functional database in which the parameters can be defined dynamically and not statically as is the case in many DB offerings. In order to do that, the respective database docker images need to be adapted in order to enable a fully parameterized deployment, in essence being converted in an as a service offering. This includes (Figure 1 left) means of automating deployment and dynamically creating configuration files (e.g. retrieving IPs of the launched containers prior to the launching of the database daemons). Parameter space exploration is also important [2], varying parameters such as data nodes and replication factors as needed by parameter sweep experiment setups. However, to vary such parameters before an experiment typically needs a number of manual processes to configure and set up the System Under Test.

Another need for synchronization (Figure 1 right) stems from either the database benchmark setup (e.g. load preparation and run phases of YCSB [3]) as well as potential needs to investigate multitenancy aspects in cases of DB as a Service offerings. Concurrent executions of discrete DB instances on the same node/cluster will create interference effects. In this case, suitable synchronization points need to be included and enforced along the way in order to ensure that the measurement phase of all deployed DBs initiate at the same time. The final aspect relates to the fact that common database benchmarks such as YCSB typically do not include or support distributed mode of operations, which means that the execution framework needs to coordinate and orchestrate their lifecycle.

In this work, and in order to fulfill the aforementioned challenges, our baseline Flexibench tool [4], which enables stress testing as a service through virtualized load injector clusters, is extended in order to include a new adapter for launching of DB experiments. The adapter ensures strict sequence of deployment and preparation of the experiment (adapted to the needs of Figure 1) as well as deployment and management of the YCSB client nodes. This enables the inclusion of the investigated System Under Test (a MySQL distributed database) in the service graph to be deployed on the container platform. Thus the framework automates and enables the service oriented launching of experiments, significantly alleviating the efforts of a performance engineer. The remainder of the paper is structured as follows. In Section 2 related work is presented, while in Section 3 the overview of the Flexibench framework is presented. In Section 4 the DB adapter is presented, while in Section 5 the different validation experiments for its operation are presented and Section 6 concludes the paper.

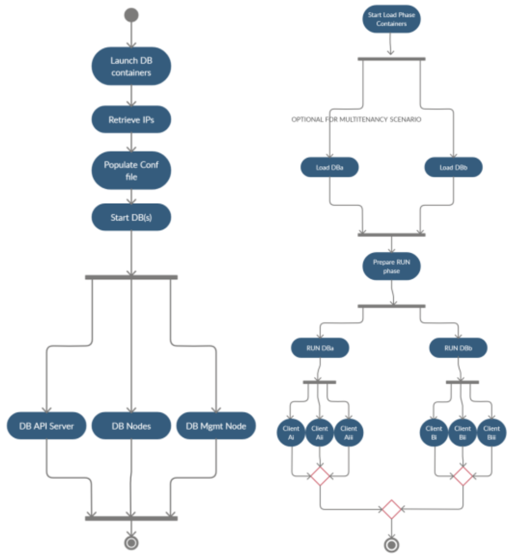

Figure 1: Synchronization Needs of Database experiments aaS

2. RELATED WORK

Specialized benchmark management solutions or investigations have appeared, including test lifecycle management processes, targeting at specific use cases and domains (e.g. cloud hosted DBMS systems elasticity aspects in [5]), security aspects[6] and HPC applications[7]. Mowgli[16] is a framework for managing NoSQL DBMS experiment lifecycles on target Cloud platforms is presented. Differences with Flexibench in this case include the focus of the latter in containerized environments and not VMs, the ability to include synchronized concurrency tests as well as the incorporation of SQL solutions. DeathStarBench[10] is a benchmark suite containing example applications as benchmarks for containerized deployments. Similarly, μSuite[11] and Teastore[12] target microservices and containers, however their main goal is to define and implement a baseline benchmark test application that can be used to measure containerized environments. Comparison of containerized environments and their related overheads are examined in [8]. Overall however, container performance evaluation is still an open issue [9].

While these works cover a range of issues, they do not enable a full scale and adapted deployment and according measurement of a specific database configuration in mind for container platforms. The ability to regulate specific parameters of the database deployment enables a more accurate performance measurement of the target system as well as the ability afterwards for creating generalized prediction models[13]. The latter can aid in predicting a given setup's QoS on varieties of configurations and workloads.

3. FLEXIBENCH MAIN FRAMEWORK

Flexibench (or Application Dimensioning WorkBench) is a layered benchmark and load injection as a service execution framework, that is based on a layered architecture[14]. On top, a REST API (as well as a UI layer) is available in order to submit test execution setups (Figure 2). The call is forwarded to a middle test synchronization layer, which undertakes the role of preparing and launching containerized stress tests through a back-end container platform (based on Docker Swarm). The framework is based on Node-RED, an event driven application framework on top of node.js, enabling the decoupling between test coordination and offloaded test execution (on the container platform). It is available as open source[1]. A demonstration of the base abilities of the framework is available through demo videos[2].

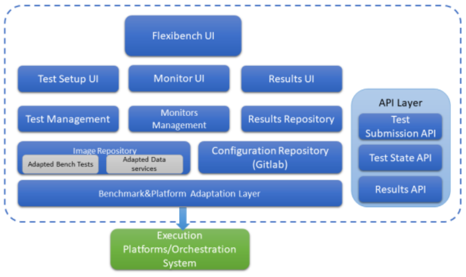

Figure 2: Flexibench Architectural Overview

4. FLEXIBENCH ADAPTERS FOR DBs

In order to enable the extension towards the abilities mentioned in the Introduction, the adaptation started from the official MySQL database images available on Dockerhub (mysql/mysql-cluster). In order to support a fully parameterized configuration and execution, alterations needed to be performed both in the main Dockerfile of the image, as well as the startup script and MySQL configuration file. The latter needs to be populated by the obtained dynamic qualified names of the DB nodes, while the actual DB daemons have not started. These names follow a naming convention based on the test name, which is linked to all created virtual resources (e.g. virtual network, volumes etc) and ensures virtual separation of the different test instances. Thus startup needs to wait until a shared sync file is populated and proceed afterwards in the db daemon setup. To adapt to different types of DBs, the process is largely similar, however it needs to follow the specific configuration templates of each DB type.

For the client side, YCSB was selected as the main load injection tool, but through its more abstracted version in OLTP-Bench[15], that may enable future incorporations of more benchmarks and load injectors that are included in the latter. Adaptations in this case included alterations in the dockerfile, startup script in order to synchronize between the load and run phases of all parallel experiments, reporting adaptation in order to be ingested by the framework afterwards etc. The latter is necessary also for grouping results from the distributed YCSB client containers.

[1] Flexibench Tool Repository, available online at: http://bigdatastack-tasks.ds.unipi.gr/gkousiou/adw

[2] Flexibench Tool Main Functionality Demo, available online at: https://youtu.be/dtAsAtc_v0s

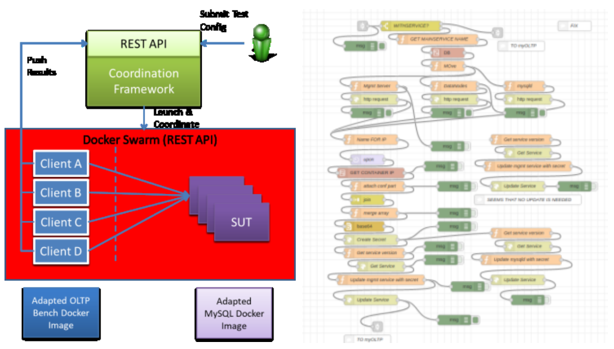

Figure 3: Overview of a) Adapter Diagram b) Node-RED flow programming adapter for DB Launch coordination

YCSB parameters that are passed in this case refer to the set rate per client, the number of threads per client as well as the workload file variant that contains the operation mix. The overall flow and setup (Figure 3) are completely parameterized, therefore discrete instances of the benchmark (including the database under test) can be deployed on demand (and through a REST interface). DB parameters that may be toggled include number of data nodes and API nodes, replication factor, record count (size of database). Metrics obtained from each run are the ones reported by YCSB itself (min/average/max latency and throughput, percentiles etc.).

5. EVALUATION

In order to test and evaluate the framework execution, a number of experiments were performed by invoking the REST API of the framework. Results are also retrieved via REST API calls. A demo video for the DB co-allocation scenario is also available[3].

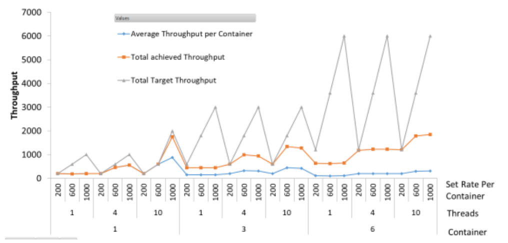

Figure 4: Client setup variations and according achieved throughput

5.1 Client Side Bottleneck Investigation

One of the dangerous points in benchmarking is the case when the client side load generation can not achieve the desired number of requests due to their own resource inadequacy or client implementation. This situation might be mistakenly considered as a bottleneck at the benchmarked service side. Therefore alternative configurations of the load generation clients need to be investigated. Figure 4 demonstrates this process for an indicative YCSB execution towards a target DB. The database itself has been tested beforehand and has been found to be able to service a ~2500 operations/sec rate, thus any shortcoming with relation to this figure can be attributed to the client setup. Client generation is performed through one Swarm node. Different client setups have been tried out, with rates set as 200,600 and 1000 operations/sec per client container, local threads set as 1,4 and 10 and client containers set as 1,3 and 6 accordingly.

Increasing the number of containers (on the same node) while maintaining 1 thread per container reduces the per container request rate but increases the total sum of the generated requests. Changing the number of threads per container is more beneficial (Figure 5).

5.2 DB Launch and Analysis experiments

In this test, the scope included the performance of benchmarks against a bundled data service (clustered MySQL), regulating the execution of both the clients and the db containers. However it needs to be stressed that the aim of these experiments is to validate the framework and not to actually extract conclusions regarding the performance of the services, given that the available Swarm installation was very small (2 nodes, one for the clients and one for the service, each with 4 cores and 8GB of RAM). Each benchmark run phase was set to 5 minutes, with primarily insert operations. The vanilla installation of MySQL was used.

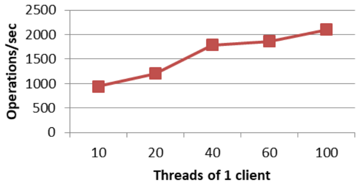

Figure 5: # of local threads effect on 1 client container

5.2.1 Parameter Sweep Tests for a single DB

In this scenario the database is launched with a varying configuration of 2 or 4 data node containers, each of which has two mysql threads (thus acting in total as 4 and 8 nodes), plus a management and API containers. Typically in clustered DB configurations increasing the number of nodes increases availability but reduces performance due to the synchronization and/or locking cases between the nodes, especially when replication is applied (in our case it was set to 2). For the case of 4 datanodes used (Figure 6a), average latency starts from 5 milliseconds and reaches 34 milliseconds in the 2000 requests per second rate. Maximum and 99th latency values show a larger increase but still within reasonable ranges (with an outlier observed at 800 requests/sec). However for the case of 8 data nodes used (Figure 6b), an increase in the request rates indicates that after the level of 1800 requests there is a very high increase in latency which reaches around 4300 milliseconds, an aspect that may be attributed (at least partially) to data synchronization needs.

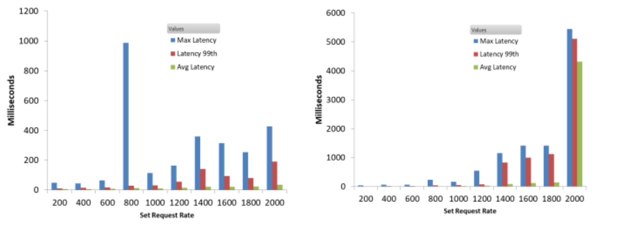

Figure 6: Investigation of query latency results for diverse request rates and a) 4 data nodes b) 8 data nodes

[3] Flexibench DB Co-allocation demo, Available at : https://youtu.be/TIv7rCVNGY8

5.2.2 DB co-allocation scenarios for DBaaS

Collocating database services of different customers on the same node is a common strategy in DBaaS environments. However in order for the provider to respect potential performance guarantees issued to their customers, they need to model the effects of this collocation on the performance of each DB service instance, comparing to standalone DB performance.

For the experiment, initially the DB is launched and benchmarked in a standalone (baseline) mode for a variety of different client rates with a single REST API call. Then two similar but distinct instances of the same configuration are deployed and collocated on the same node and according client rates are launched. This step is easily performed through Flexibench through the parallel mode of testing (1 REST call with 2 configurations and the parallel mode selected). In this process it also enforces the workflow mentioned in Figure 1. In Figure 7 the results of such an analysis are portrayed. From this it can be seen that for small numbers of queries/second there is no significant difference in the performance of the collocated DBs compared to the standalone mode. However when insert queries/second rise above the level of ~400/second then there is a clear degradation of the performance of each DB compared to the standalone version. Further testing was performed to check accumulated values of 800 queries per second (Figure 8).

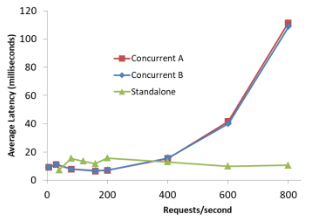

Figure 7: Investigation of collocated DB instances performance in comparison to baseline (standalone) execution

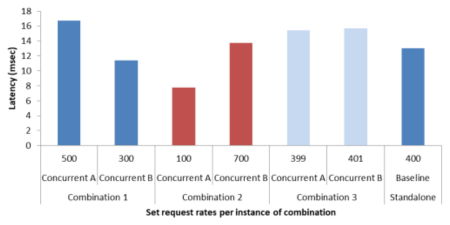

Figure 8: Investigation of different request rates per DB service that add up to the threshold value of 800

6. CONCLUSION

The execution, coordination and deployment of multiple variations of a system such as a distributed database includes various configuration steps, both for the SUT setup as well as for the client distributed setup. The presented framework achieves the automation of this process and enables the submission of test variations through REST interfaces, through a coordination logic that guarantees test synchronization and successful execution of containerized deployments. Thus it enables the investigation and extensive performance analysis of diverse configurations without significant effort from the performance engineer.

7. ACKNOWLEDGMENTS

The research leading to the results has received funding from the European Union's funded Projects BigDataStack (GA No 779747) and PHYSICS (GA No 101017047).

8. REFERENCES

[1] M. Grambow, J. Hasenburg, T. Pfandzelter, and D. Bermbach. 2019. Is it safe to dockerize my database benchmark? In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19). Association for Computing Machinery, New York, NY, USA,341–344. DOI:https://doi.org/10.1145/3297280.3297545

[2] Silva, M., Hines, M.R., Gallo, D., Liu, Q., Ryu, K.D. and Da Silva, D., 2013, March. Cloudbench: Experiment automation for cloud environments. In 2013 IEEE International Conference on Cloud Engineering (IC2E) (pp. 302-311). IEEE. DOI: 10.1109/IC2E.2013.33

[3] Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R. and Sears, R., 2010, June. Benchmarking cloud serving systems with YCSB. In Proceedings of the 1st ACM symposium on Cloud computing (pp. 143-154). ACM.

[4] Kyriazis, D., Doulkeridis, C., Gouvas, P., Jimenez-Peris, R., Ferrer, A.J., Kallipolitis, L., Kranas, P., Kousiouris, G., Macdonald, C., McCreadie, R. and Moatti, Y., 2018, July. BigDataStack: A holistic data-driven stack for big data applications and operations. In 2018 IEEE International Congress on Big Data (BigData Congress) (pp. 237-241)

[5] Seybold, D., Volpert, S., Wesner, S., Bauer, A., Herbst, N.R. and Domaschka, J., 2020, January. Kaa: evaluating elasticity of cloud-hosted DBMS. In Proceedings. The 11th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2019) (Vol. 2019, pp. 54-61).

[6] Osman, A., Hanisch, S. and Strufe, T., 2019, June. SeCoNetBench: A modular framework for Secure Container Networking Benchmarks. In 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW) (pp. 21-28). IEEE.

[7] Wang, Y., Evans, R.T. and Huang, L., 2019. Performant container support for HPC applications. In Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning) (pp. 1-6).

[8] Kozhirbayev, Z. and Sinnott, R.O., 2017. A performance comparison of container-based technologies for the cloud. Future Generation Computer Systems, 68, pp.175-182.

[9] Bachiega, N.G., Souza, P.S., Bruschi, S.M. and de Souza, S.D.R., 2018, April. Container-based performance evaluation: A survey and challenges. In 2018 IEEE International Conference on Cloud Engineering (IC2E) (pp. 398-403). IEEE.

[10] Gan, Y., Zhang, Y., Cheng, D., Shetty, A., Rathi, P., Katarki, N., Bruno, A., Hu, J., Ritchken, B., Jackson, B. and Hu, K., 2019, April. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (pp. 3-18).

[11] Sriraman, A. and Wenisch, T.F., 2018, September. μ Suite: A Benchmark Suite for Microservices. In 2018 IEEE International Symposium on Workload Characterization (IISWC) (pp. 1-12). IEEE.

[12] v. Kistowski, J., Eismann, S., Grohmann, J., Schmitt, N., Bauer, A. and Kounev, S., 2019, March. TeaStore-A Micro-Service Reference Application for Performance Engineers. In Companion of the 2019 ACM/SPEC International Conference on Performance Engineering (pp. 47-48).

[13] Kousiouris, G., Kyriazis, D., Gogouvitis, S., Menychtas, A., Konstanteli, K. and Varvarigou, T., 2011, June. Translation of application-level terms to resource-level attributes across the Cloud stack layers. In 2011 IEEE Symposium on Computers and Communications (ISCC) (pp. 153-160).

[14] BigDataStack Project Deliverable 5.1, Available at: https://bigdatastack.eu/deliverables/d51-dimensioning-modelling-and-interaction-services-bigdatastack

[15] Difallah, D.E., Pavlo, A., Curino, C. and Cudre-Mauroux, P., 2013. Oltp-bench: An extensible testbed for benchmarking relational databases. Proceedings of the VLDB Endowment, 7(4), pp.277-288.

[16] Seybold, D., Keppler, M., Gründler, D. and Domaschka, J., 2019, April. Mowgli: Finding your way in the DBMS jungle. In Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering (pp. 321-332).