Is it Safe to Dockerize my Database Benchmark?

Martin Grambow, Jonathan Hasenburg, Tobias Pfandzelter, David Bermbach
TU Berlin & Einstein Center Digital Future, Mobile Cloud Computing Research Group
mg,jh,tpz,db@mcc.tu-berlin.de

ABSTRACT

Docker seems to be an attractive solution for cloud database benchmarking as it simplifies the setup process through pre-built images that are portable and simple to maintain. However, the usage of Docker for benchmarking is only valid if there is no effect on measurement results. Existing work has so far only focused on the performance overheads that Docker directly induces for specific applications. In this paper, we have studied indirect effects of dockerization on the *results* of database benchmarking. Among others, our results clearly show that containerization has a measurable and non-constant influence on measurement results and should, hence, only be used after careful analysis.

CCS CONCEPTS

• **General and reference** → *Measurement*; *Performance*;

KEYWORDS

Database benchmarking, Docker, Performance

ACM Reference Format:
Martin Grambow, Jonathan Hasenburg, Tobias Pfandzelter, David Bermbach. 2019. Is it Safe to Dockerize my Database Benchmark?. In *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3297280.3297545

1 INTRODUCTION

Benchmarking has long been used for the comparison of software and hardware systems or software versions [11]. When done right, it is surprisingly hard as conflicting goals such as reproducibility, portability, understandability, fairness, ease-of-use, and relevance need to be balanced [2, 9, 12, 16]. When focusing on reproducibility and ease-of-use, an engineer running a systems benchmark is likely to encounter two main challenges: First, correctly installing and configuring both benchmarking client and the system under test (SUT) can be error-prone and challenging, or at least involves a lot of effort. Second, for reproducibility reasons, benchmark runs need to be repeated several times – preferably on a fresh system setup which aggravates the first challenge.

A solution that naturally lends itself to these challenges is to dockerize [13] both benchmarking client and SUT, thus, using containers as a convenient deployment mechanism for preconfigured, ready-to-use experimental setups. This has already been done, e.g., by Palit et al. [15]. However, it is unclear whether this will affect benchmarking results. There are several studies, e.g., [5–7, 13, 17], measuring the overheads that various applications might incur when running inside containers instead of on bare metal or inside a virtual machine. These, however, all quantify the overhead that is induced by Docker for a certain workload or application. Neither of these studies measures indirect effects of Docker that, for instance, a database benchmark running against an SUT on another machine might experience. For such a benchmark, indirect effects might lead to volatile and unpredictable changes in benchmarking results rendering results at least partially obsolete. See also figure 1 which gives a high-level overview of the different perspectives taken in related work (on the left) and in this paper (on the right).

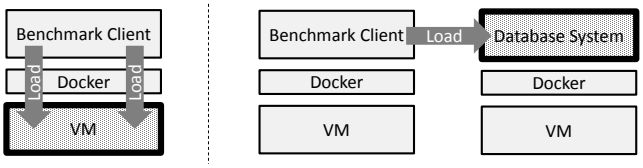

Figure 1: Different Perspectives in Experimentation with Docker

In this paper, we aim to answer the question whether it is safe to dockerize database benchmarks, i.e., whether dockerization of benchmarking client and/or SUT has observable effects on measurement results. For this purpose, we designed a set of experiments that not only quantifies possible dockerization impacts on benchmarking results but also explores whether different standard settings of both benchmarking client and SUT can further influence potential impacts. Based on this, as our main contribution, we discuss the results of extensive experimentation with YCSB[1] (the de-facto standard for benchmarking of database systems) and Apache Cassandra[2] (a widely used NoSQL system) running on Amazon EC2[3]. As a second contribution, we use our observed results to give recommendations and identify implications for database benchmarking with and without Docker[4].

2 RELATED WORK

There are already several publications trying to quantify the performance overhead of dockerization. For instance, Chung et al. [5] have benchmarked high performance computing applications (HPL and Graph500) running in Docker containers and found remarkable differences to the performance without docker.
In difference to the findings of Chung et al, Di Tommaso et al. [6] have tested Docker's impact on the performance of genome analysis

SAC '19, April 8–12, 2019, Limassol, Cyprus

ACM ISBN 978-1-4503-5933-7/19/04.
https://doi.org/10.1145/3297280.3297545

[1] github.com/brianfrankcooper/YCSB
[2] cassandra.apache.org
[3] aws.amazon.com/ec2
[4] An extended version of this paper is available as technical report [10]

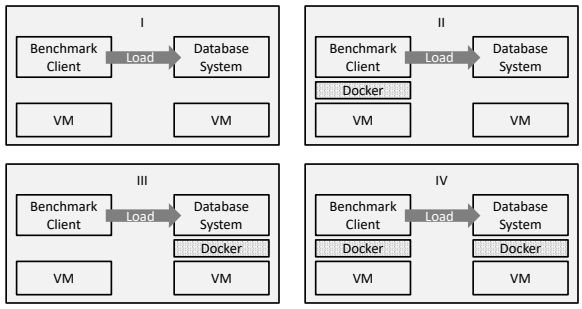

Figure 2: Dockerization Variants in Database Benchmarking

pipelines and concluded that the Docker technology only introduces a negligible performance overhead for their purposes. They executed multiple tests on a cluster of 12 high performance machines and compared the execution time of tasks running in Docker containers to the native performance.

Felter et al. [7] compared the performance of Docker containers and virtual machines utilizing microbenchmarks on a 32 vCPU instance equipped with a not specified but "adequate" amount of memory to execute the given workload. Similar to Di Tommaso et al., they conclude that Docker introduces a negligible computation and memory overhead in most cases, but I/O-intensive workloads should be used carefully as extra cycles are needed for each I/O operation.

Ali et al. [1] benchmarked the performance impact of Docker together with VM technology using microbenchmarks and measured overheads of up to 4%.

In all these publications, the authors measured the *directly* visible overhead of Docker for a certain workload or application. Somewhat comparable to the TLS experiments of Müller et al. [14], we are interested in *indirect* effects of Docker on applications running inside containers. Specifically, we aim to evaluate whether it is safe to dockerize benchmarks, i.e., whether dockerization of benchmarking components does not affect measurement results – neither actual measurement values nor their stability and reproducibility. To the best of our knowledge, this has not been experimentally studied yet. However, such an evaluation is needed as some cloud service benchmarking tools already use Docker as a deployment mechanism. For instance, Palit et al. [15] published a suite of cloud service benchmarking tools as Docker images, Ceesay et al. [4] have built an entire cloud benchmarking framework around Docker, and Ferme et al. [8] have containerized the benchmarking of workflow management systems.

3 EXPERIMENT DESIGN

With our experiments, we analyze whether dockerization of benchmarking client and/or SUT has observable effects on measurement results, figure 2 illustrates the four base configurations to compare: no dockerization (I), dockerized benchmarking client (II), dockerized SUT (III), and full dockerization (IV).

Through this comparison, we can not only determine whether Docker has an impact on benchmarking results but also determine whether the effect is caused at the SUT or the benchmarking client. For each of the four setups, we also tweaked various parameters in the benchmarking client and in the SUT to evaluate whether the dockerization impact can also be affected by other parameters. Therefore, we ran all experiments on both m3.medium and m3.large instances in the AWS region Ireland, with client and SUT running on two instances of the same time within the same availability zone. We chose these instance types as they do not provide credit-based "burst" performance might render results meaningless. Each experiment was repeated 5 times.

Machine	Parameter	Variations
Client	Docker	Yes, No
	Thread Count	10, 25, 50 OR 30, 75, 150
SUT	Docker	Yes, No
	Key Cache Size	0, Auto
	Compaction Strategy	STCS, TWCS, LCS
General	Instance Type	m3.medium, m3.large
Total Number of Experiments		720

Table 1: Experimental Parameter Variations

To ensure a consistent environment, we fully automated the benchmarking process and pre-built all used Amazon Machine Images and Docker containers; both containing only the software necessary for the experiments. We also monitored machine resource utilization to avoid a bottleneck in the client machine. We deployed a single Apache Cassandra node as SUT in our experiments[5]. To simulate a "typical" setup, we used default values for all configuration options in our experiments but also changed Cassandra's compaction strategy (Size Tiered Compaction Strategy (STCS), Time Window Compaction Strategy (TWCS), Level Compaction Strategy (LCS)) and key cache size (disabled or "auto"). On the client machine, we ran YCSB using Workload A (50% reads and 50% writes). For the m3.medium instances, we used 100,000 records and 3,000,000 operations; for the m3.large instances, we used 100,000 records and 9,000,000 operations to achieve sufficiently long-running experiments. In preparatory experiments, we observed that throughput does not increase beyond 50/150 threads for m3.medium/m3.large respectively, while latencies continued to increase. To measure the effects of dockerization under different resource utilizations, we varied the thread counts used (see table 1); in total we ran 720 experiments.

4 RESULTS

In this section, we discuss the results of our experiments starting with aggregates before continuing with a more detailed look.

4.1 General Results

Dockerization introduces an additional layer so that we expected a decrease in throughput when using Docker (setups II and III), especially for the fully dockerized setup (setup IV). As expected, our findings show that the dockerization of benchmarking components typically increases latency and decreases throughput. However, in

[5]We chose a single node deployment to reduce the number of influence parameters in our experiments.

th	op	I	II	III	IV
10	Read	0%	1.42%	1.84%	1.21%
	Update	0%	1.79%	5.13%	4.53%
25	Read	0%	−0.09%	−0.01%	0.69%
	Update	0%	0.50%	3.06%	3.73%
50	Read	0%	2.51%	−0.52%	−0.97%
	Update	0%	2.72%	2.66%	2.20%
	Read avg	0%	1.28%	0.44%	0.31%
	Update avg	0%	1.67%	3.62%	3.49%

Table 2: Rel. Changes of Avg. Latency for $n = 30$ Experiments

th	op	I	II	III	IV
30	Read	0%	5.53%	7.02%	11.74%
	Update	0%	5.81%	7.25%	12.07%
75	Read	0%	6.81%	2.50%	8.85%
	Update	0%	6.73%	2.43%	8.70%
150	Read	0%	−0.63%	−2.38%	1.46%
	Update	0%	2.96%	1.43%	5.36%
	Read avg	0%	3.90%	2.38%	7.35%
	Update avg	0%	5.17%	3.70%	8.71%

Table 3: Rel. Changes of Avg. Latency for $n = 30$ Experiments

some cases Docker even increased throughput which is probably caused by variance of the underlying cloud infrastructure.

In our results read and update latency appear to be closely related; there seems to be no dockerization effect that only affects one of the two.

For our analysis, we calculated the average read and update latency grouped by instance type, thread count (*th*), and degree of dockerization (I-IV), each average value was based on 30 experiment runs. Tables 2 and 3 show relative latency changes of these averaged values compared to the baseline across all four setups: Setup I (no dockerization) is the baseline, setups II and III (partial dockerization), and setup IV (full dockerization); a latency increase is a positive value.

In general, our results show an increasing read and update latency of operations if components are dockerized. Especially update operations provoke a significant overhead, on average 8.71% for m3.large instances. On the other hand, we also observed read operation performance improvements for experiments with higher thread counts on the client side. Furthermore, we discovered that dockerization has a stronger effect on m3.large instances. Finally, the overheads appear to increase for lightly loaded systems. Already these aggregates indicate that dockerization of database benchmarks should only be done after careful analysis.

4.2 Median Experiment Runs

As described in section 3, we repeated each experiment 5 times to account for random fluctuation of the cloud infrastructure. We, here, report the results of the median runs instead of calculating averages as the median is more stable in the presence of outliers. We define the median run for a specific parameter set as the run with the median throughput. In this context, please, note that YCSB reports the average latency. We, here, report the median of 5 such average latency values.

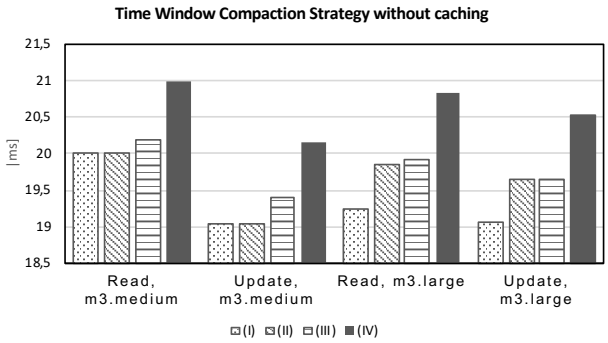

Figure 3: Typical Result: Dockerization Increases Latency

Most experiments had similar results. In figure 3, we show an example (m3.large and m3.medium, TWCS compaction, disabled cache, 50 and 150 threads respectively); Our full data set is available on GitHub[6]: Dockerizing either YCSB or Cassandra typically increases the average latency for read and update operations slightly, while full dockerization has a much stronger effect. The standard deviation of latencies (across the respective five experiment repetitions) varied from 0.21 ms to 1.38 ms for m3.medium instances, and from 0.30 ms to 0.95 ms for m3.large instances respectively. However, we also found very few experiments where results deviated from the patterns set in figure 3. Overall, we believe that the more unexpected results might be caused by random performance variation of the underlying VMs.

4.3 Settings in Cassandra

Besides the general influence of Docker on the benchmarking process, we also evaluated different settings in the configuration of Cassandra. Our experiments indicate that changing the key cache setting does neither influence the performance of Cassandra itself (for our experiment workload), nor does it influence the dockerization effects we reported in section 4.2. Especially for light workloads, our results clearly show that caching neither has an effect on the performance of Cassandra nor does it result in an additional influence on the dockerization effects: At higher load (50 and 150 threads respectively), however, this is still not clear as the overall variance of results (no matter whether dockerized or not) was too high to draw a well-founded conclusion. Besides the cache setting, we also evaluated how Docker influences the benchmarking results when running the experiments with different Cassandra compaction strategies. On m3.medium instances, we found that the chosen compaction strategy has a negligible impact on Cassandra when running a light workload. This, of course, implies that the compaction strategy does not have an influence on the existing dockerization effects. On m3.large instances, again lightly loaded, we found that the compaction strategies have an influence

[6] https://github.com/martingrambow/dockerExperiments

on Cassandra (with leveled compaction being the fastest) but this influence is constant across different dockerization setups. This means that the compaction strategy does not cause an additional effect on the dockerization impacts. On both machine types, we again found large overall performance variability across experiment runs (independent of the degree of dockerization). We assume that this increased variance is caused by the high utilization of the virtual machines which might lead to conflicts in the scheduling of threads and thus results in unstable latencies. A more detailed analysis for heavy workloads requires significantly more experiments and further investigation; this is beyond the scope of our paper.

5 IMPLICATIONS

As described in chapter 4, our results clearly show an influence of Docker on the results of database benchmarking experiments. However, this influence is not constant as it varies for different configurations and can be up to 12% (in our experiments) which may be still acceptable for some use cases. So what does this mean for database benchmarking?

First, results of dockerized benchmarks can be acceptable when comparing different database systems. In such a case, the absolute measurement values should be disregarded; the ordering of system alternatives, however, is unlikely to change if the difference between alternatives is sufficiently large – e.g., greater than 20-30%.

Second, benchmark setups should be as close as possible to the production environment that they try to emulate [3]. This, however, implies that when production systems are supposed to be dockerized, benchmarking systems also need to be dockerized when measurement accuracy matters.

Third, when evaluating system configurations or implementation alternatives, it may be an option to dockerize the benchmark (as is commonly done in build processes). However, such results can only be used to achieve a general "feeling" of a system's performance. Actual numbers are too unreliable.

Fourth, in many cases it may be acceptable to dockerize the benchmark as long as it stays dockerized and no configuration changes are made. Absolute values should still not be compared to non-dockerized values directly (or should be taken with a grain of salt), but the workload generation and measurements should be stable enough for comparison over multiple measurements.

Finally, repeating sufficiently long experiments is always important in benchmarking. When using dockerization, however, even more repetitions and longer experiments should be used to identify random fluctuations introduced through another layer of indirection.

6 CONCLUSION

In this paper, we acknowledged the growing importance of Docker for the benchmarking community. However, we noted that the effect of dockerization on benchmarking results is still unclear as prior studies have only measured the direct overhead of dockerization on certain workloads or applications. To extend prior attempts to quantify the influence of Docker, we ran a series of 720 experiments with and without Docker of a Cassandra-YCSB benchmarking setup.

Our results show that the dockerization of benchmarking tool and/or SUT indeed has an influence on benchmarking results that ranges from -2% to 12%. Furthermore, we observed that the impact of Docker on latency is less prominent for heavy workloads. Based on our results, we conclude that the dockerization of benchmarking system and SUT is not a good idea (if the production environment is not dockerized as well), and that results should only be used for a general ranking of SUTs.

In future work we would like to study Docker's impact on a distributed Cassandra cluster, run additional experiments to better understand the impact of Docker's layered file system, and build a framework for benchmark automation that makes careful use of dockerization based on a knowledge base of dockerization impacts.

REFERENCES

[1] Q. Ali, B. Agrawal, and D. Bergamasco. Docker containers performance in vmware vsphere. *https://blogs.vmware.com/performance/2014/10/docker-containers-performance-vmware-vsphere.html (accessed on Apr 21, 2018)*, 2014.

[2] D. Bermbach, J. Kuhlenkamp, A. Dey, S. Sakr, and R. Nambiar. Towards an Extensible Middleware for Database Benchmarking. In *Proc. of TPCTC*. Springer, 2014.

[3] D. Bermbach, E. Wittern, and S. Tai. *Cloud Service Benchmarking: Measuring Quality of Cloud Services from a Client Perspective*. Springer, 2017.

[4] S. Ceesay, A. Barker, and B. Varghese. Plug and play bench: Simplifying big data benchmarking using containers. *CoRR*, abs/1711.09138, 2017.

[5] M. T. Chung, N. Quang-Hung, M.-T. Nguyen, and N. Thoai. Using docker in high performance computing applications. In *Proc. of ICCE*. IEEE, 2016.

[6] P. Di Tommaso, E. Palumbo, M. Chatzou, P. Prieto, M. L. Heuer, and C. Notredame. The impact of docker containers on the performance of genomic pipelines. *PeerJ*, 3, 2015.

[7] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and linux containers. In *Technical Report RC25482*. IBM Research, 2014.

[8] V. Ferme, A. Ivanckikj, C. Pautasso, M. Skouradaki, and F. Leymann. A container-centric methodology for benchmarking workflow management systems. In *Proc. of CLOSER 2017*. SciTePress, 2016.

[9] E. Folkerts, A. Alexandrov, K. Sachs, A. Iosup, V. Markl, and C. Tosun. Benchmarking in the cloud: What it should, can, and cannot be. In *Proc. of TPCTC 2012*. Springer, 2013.

[10] M. Grambow, J. Hasenburg, T. Pfandzelter, and D. Bermbach. Dockerization impacts in database performance benchmarking. In *Technical Report MCC.2018.1. TU Berlin & ECDF, Mobile Cloud Computing Research Group*. https://github.com/martingrambow/dockerExperiments, 2018.

[11] J. Gray. Database and transaction processing handbook. *The Benchmark Handbook for Database and Transaction Systems*, 1993.

[12] K. Huppler. The art of building a good benchmark. In *Proc. of TPCTC 2009*. Springer, 2009.

[13] D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(vol. 239), 2014.

[14] S. Müller, D. Bermbach, S. Tai, and F. Pallas. Benchmarking the performance impact of transport layer security in cloud database systems. In *Proc. of IC2E*. IEEE, 2014.

[15] T. Palit, Y. Shen, and M. Ferdman. Demystifying cloud benchmarking. In *Proc. of ISPASS*. IEEE, 2016.

[16] J. v. Kistowski, J. A. Arnold, K. Huppler, K.-D. Lange, J. L. Henning, and P. Cao. How to build a benchmark. In *Proc. of ICPE 2015*. ACM, 2015.

[17] B. Varghese, L. T. Subba, L. Thai, and A. Barker. Container-based cloud virtual machine benchmarking. In *Proc. of IC2E*. IEEE, 2016.