Performance of Cluster-based High Availability Database in Cloud Containers

Raju Shrestha
OsloMet - Oslo Metropolitan University, Oslo, Norway
raju.shrestha@oslomet.no

Keywords: Performance, High availability, Database, Cloud, Galera cluster, Virtual Machine, Container, Docker

Abstract: Database is an important component in any software application, which enables efficient data management. High availability of databases is critical for an uninterruptible service offered by the application. Virtualization has been a dominant technology behind providing highly available solutions in the cloud including database, where database servers are provisioned and dynamically scaled based on demands. However, containerization technology has gained popularity in recent years because of light-weight and portability, and the technology has seen increased number of enterprises embracing containers as an alternative to heavier and resource-consuming virtual machines for deploying applications and services. A relatively new cluster-based synchronous multi-master database solution has gained popularity recently and has seen increased adoption against the traditional master-slave replication for better data consistency and high availability. This article evaluates the performance of a cluster-based high availability database deployed in containers and compares it to the one deployed in virtual machines. A popular cloud software platform, OpenStack, is used for virtual machines. Docker is used for containers as it is the most popular container technology at the moment. Results show better performance by HA Galera cluster database setup using Docker containers in most of the Sysbench benchmark tests compared to a similar setup using OpenStack virtual machines.

1 INTRODUCTION

Most of today's modern cloud-based applications and services, in general, are dynamic database-driven web-based applications. In order to provide always-on and always-connected service, which is one of the major goals of cloud computing, it is vital that the database is highly available. In other words, a high availability (HA) database is critical for interrupted service. HA database aims for reduced downtime as well as for reduced response time and increased throughput (Hvasshovd et al., 1995). As the HA database cannot be retrofit into a system, it should be designed in the early stages of the system architecture design to provide minimal downtime, optimal throughput and response time. HA database is realized through redundancy, where multiple database servers are used and data is replicated in all the servers.

There are mainly two major replication technologies that are used for high availability database: master-slave database replication and cluster-based database replication. Master-slave (single master, multiple slaves) database replication (Ladin et al., 1992; Wiesmann et al., 2000; Earl and Oderov, 2003; Wiesmann and Schiper, 2005; Curino et al., 2010) is a solution which is being used traditionally since long time. A master database server handles data writes, while multiple slave servers are used to read data, thus supporting read scalability. Any changes in the master database are replicated in the slaves asynchronously (Wiesmann et al., 2000; Elnikety et al., 2005). Since asynchronous replication doesn't guarantee the delay between applying changes on the master and propagation of changes to the slaves, this may cause data inconsistencies when something goes wrong in the master database server in the middle of a transaction. Thus, master-slave replication doesn't guarantee consistency, one among the three consistency, availability, and partition tolerance in the Brewer's CAP theorem (Brewer, 2012).

Multi-master cluster-based database replication techniques such as MariaDB Galera cluster (MariaDB Galera, 2019a) and MySQL cluster (MySQL, 2019) offer effective alternatives to master-slave replication by addressing its data inconsistency problem through synchronous replication. A comparative study between master-slave and cluster-based high availability database solutions showed that master-slave replication performs equal or better in terms of throughput and response time (Shrestha, 2017). However, the cluster-based solution is superior when it comes to high availability, data consistency, and scalability as it offers instantaneous failover, no loss of data, and both read and write scalability.

Virtualization is a key enabling technology of cloud computing. Most HA solutions rely on redundant hardware, and virtualization technology enables efficient and effective HA in the cloud through provisioning and deployment of virtual machines (VMs) and the whole infrastructure (IaaS). Virtual machines are then scaled based on demands (Xing and Zhan, 2012). Since each VM includes an operating system (OS) and a virtual copy of all the hardware, virtualization requires significant hardware resources such as CPU, RAM, and Disk space. Moreover, because of their large size and resource requirements, moving VMs between different clouds can be challenging. Container technology (simply called as a container) addresses this, whereby an application or service is packaged so that it can be provisioned and run isolated from other processes (He et al., 2012; Kang et al., 2016). Unlike virtual machines, which are separate full-blown machines with their own OS, containers run on top of a single OS. As such, there is a lot of overhead and resource requirement with virtual machines, whereas containers are more efficient requiring significantly less overhead and resources (MaryJoy, 2015). Because of the container's lightweight nature, many containers can be deployed onto a single server. Studies showed that containers, in general, take significantly less boot time, and use fewer resources (Seo et al., 2014; MaryJoy, 2015; Felter et al., 2015; Kozhirbayev and Sinnott, 2017; Zhang et al., 2018). At the same time, generating, distributing, and deploying container images is fast and easy. (Mardan and Kono, 2016) did a comparative study between KVM based virtualization and Linux LXC container in terms of disk I/O and isolation in a database management system, and they found KVM to perform better than LXC in disk I/O without violating the isolation. However, this may no longer be true with the recent development of container technology and with different virtualization and container technologies. In the meantime, container technology has gained popularity in recent years and enterprises are embracing containers as an alternative to virtual machines for deploying applications and services. There are several container software and platform available such as Docker, Linux container (lxc), Apache Mesos, and rkt (rocket) (Bernstein, 2014). Among them, Docker is the most popular and widely used at the time.

Even though some research has been done in studying performance comparison of the container with the virtual machine in general, to the author's knowledge, not much study being done regarding the performance of container in the specific application domain of database, especially cluster-based HA database. This paper has made an attempt in this direction, where the performance of the container-based HA cluster database setup is evaluated and compared with a similar setup using virtual machines. Since MariaDB database is not only free, and open-source, but also has more cutting edge features and storage engines, compatible with MySQL, performs better, and easy to migrate (Seravo, 2015; Nayyar, 2018), and Docker is the most popular container platform, Galera cluster implementation using the latest MariaDB database server at the time (v.10.4) and Docker container are used to implement cluster-based HA database in cloud.

After this introduction section, Section 2 describes the cluster-based high availability database. Section 3 describes the experimental setup, tests carried out, and evaluation metrics used to compare performance. Section 4 presents and discusses the experimental results. Finally, Section 5 concludes the paper.

2 CLUSTER-BASED HIGH AVAILABILITY DATABASE

A typical cluster-based HA database is relatively new, which uses a multi-master architecture consisting of a cluster (or group) of servers, called nodes. Any node within a cluster can respond to both read and write requests. Any change of data in a node is replicated across all nodes in the cluster instantly, thus providing system redundancy and minimizing the possibility of service downtime. Cluster nodes can be load-balanced using a database proxy such as MariaDB MaxScale (MariaDB MaxScale, 2019) to make the database highly available. Figure 1 illustrates a cluster-based architecture.

Data replication in a cluster-based architecture is performed synchronously, which unlike asynchronous replication, guarantees that if any change happens on one node of the cluster, the change is applied in other nodes as well at the same time. Therefore, even when a node fails, the other nodes continue working and hence the failure has no major consequences. The data gets replicated from the existing nodes when the failed node joins the cluster again later. It thus guarantees data consistency.

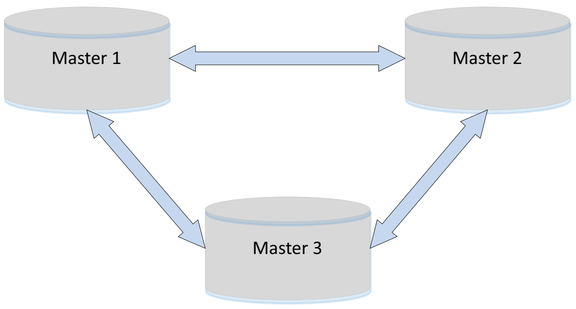


Figure 1: Cluster-based multi-master architecture for a high availability database.

The most recent version of the MariaDB database server and it's inbuilt Galera cluster is used in this work for HA database implementation as it is a community-developed open-source software. Galera cluster uses a write-set replication (WSREP) service for synchronous multi-master replication. Transactions are either applied to every node or not at all, thus guaranteeing data consistency. Galera is capable of automatic node provisioning. If one master fails, the cluster continues so that users can continue reading and writing on other nodes.

3 EXPERIMENTAL SETUP AND TESTS

OpenStack-based (OpenStack.org, 2019) cloud setup at Oslo Metropolitan University (OsloMet), which is presumably one of the largest cloud environments in Norwegian higher education, is used to conduct experiments. Two OpenStack projects, one for the VM-based HA cluster database setup and one for the Docker container-based HA cluster database setup are created. In each case, a Galera cluster of three database servers is set up as a part of a LAMP stack (IBM Cloud, 2019) using the latest version of MariaDB (v10.4) and built-in Galera v4 (MariaDB Galera, 2019a). Galera clusters are setup by using the same configuration in all the database servers in both VM-based and container-based implementations. `rsync` is used as the WSREP snapshot transfer method (`wsrep_sst_method`) in the Galera cluster configuration. Figure 2 shows the common configuration added within the [mysqld] section in the MariaDB configuration file `my.cnf`. MariaDB's MaxScale v2.4, (MariaDB MaxScale, 2019), an intelligent database proxy, is used to load balance the three database servers on a round-robin basis. Since all the database servers are the same specification, equal weight is given to each server. Figure 3 depicts the Galera cluster setup. Figure 4 shows an example listing of running database servers as obtained with MaxCtrl command in MaxScale in VM-based HA setup.

```
[mysqld]
bind-address = 0.0.0.0
default_storage_engine = innodb
binlog_format = row
query_cache_size = 0
query_cache_type = 0
innodb_autoinc_lock_mode = 2
innodb_flush_log_at_trx_commit = 0
wsrep_on = ON
wsrep_provider = /usr/lib/galera/libgalera_smm.so
wsrep_sst_method = rsync
```

Figure 2: MariaDB configuration used to setup Galera clusters in both VM-based and container-based HA database implementations.

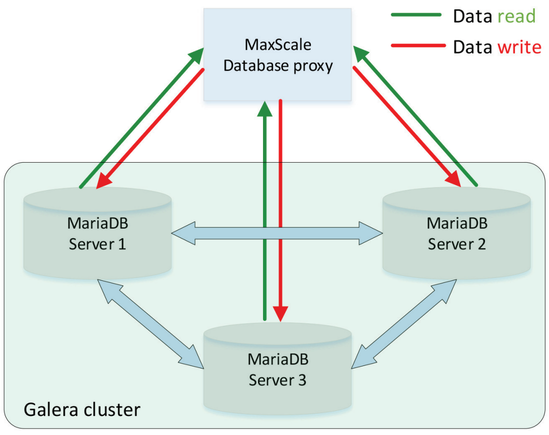


Figure 3: Experimental setup of Galera cluster-based high availability database using three MariaDB database server nodes and MariaDB MaxScale database proxy.

Server	Address	Port	State
db1	dbgc1	3306	Master, Synced, Running
db2	dbgc2	3306	Slave, Synced, Running
db3	dbgc3	3306	Slave, Synced, Running

Figure 4: List showing running database servers and their roles at the time.

The two HA database setups and benchmark tests carried out, and the metrics used to evaluate their performance are described below.

Galera cluster setup using VMs: All servers (VMs) including three database servers are created of small flavor (1VCPU, 2GB RAM, and 20GB Disk) and deployed in a private network. Necessary security rules are defined and ports are opened to allow TCP network communication between them. Ubuntu 18.04 OS is used in all the servers. MariaDB database server v10.4 is installed in all three database servers and they are configured for the HA Galera cluster database.

Galera cluster setup using Containers: A single Ubuntu 18.04 VM of extra-large flavor (8VCPU, 16GB RAM and 160GB Disk) is created in OpenStack and used as a Docker host to provision all the containerized servers. Galera cluster is set up with the three database servers using the standard Docker image `mariadb/server:10.4` from MariaDB. Three separate external storage SSD data volumes each of 20GB size are attached to the VM. One data volume is used as a persistent data storage for a database server, by connecting to the container using the `--volume` parameter when running the container. MariaDB's MaxScale Docker image, `mariadb/maxscale:latest` (v2.4), is used for setting up the database proxy. Containers are provisioned in a private network so that they can communicate with each other.

In Docker, a container has no resource constraints by default and can use as much of a given resource as the host's kernel scheduler allows. In order to make a fair comparison with the VM-based setup, the database containers are enforced to limit the same number of CPUs, memory, and Disk size as in the database VMs, which are 1CPU, 2GB, and 20GB respectively. CPU and memory constraints are defined by setting the limits to the `--cpus` and `--memory` parameters when running the docker containers. For limiting the memory in Docker, control group (cgroup) is enabled in Docker host by adding `GRUB_CMDLINE_LINUX="cgroup_enable=memory swapaccount=1"` in `/etc/default/grub` and then updating the grub. The same configuration is set in all the VMs in the VM-based setup as well.

Benchmark tests: A mature and widely used open-source benchmark suite, Sysbench, developed by Alexy Kopytov (Kopytov, 2014; Kopytov, 2019), which provides most of the database benchmark tests, is used to perform various benchmark tests under intensive loads with common database operations. The same tests are run in both VM-based HA database setup and container-based HA database setup using the same set of parameters in each test. The latest version of Sysbench (v1.0.19) is installed and executed from a separate test VM in case of the VM-based setup, and from the Docker host in case of the container-based setup. MaxScale proxy server is used as the database host when running Sysbench tests so that all the database requests are made to the Galera cluster via the database proxy.

Five different types of benchmark tests are performed, namely Readonly, WriteOnly, ReadWrite, Bulkinsert, and Deleteonly, using OLTP (Online Transaction Processing) benchmark that comes with Sysbench. OLTP is used as it is close to common real-world dynamic database-driven web applications. Sysbench's default option 'special' for data distribution and default values for the related parameters are used as it is quite common in web applications. Tests are pre-warmed or warmed up, allowing to warm up the cache and buffer pools in order to collect regular statistical results and then run with a time limit of two minutes. As we wanted to test regular queries, prepared statements were disabled in all the tests. With the default settings, Sysbench data preparation shows that it takes about 375MB disk space for one million rows. In order to test in-memory workload, the parameters `--tables` (number of database tables or table count) and `--table-size` (number of rows in a table) are set to thirty and one hundred respectively to fit well into innoDB buffer pool.

Five benchmark tests are briefly described and corresponding Sysbench run commands used are given below. `tblcount`, `tblsize`, and `runtime` variables used in the commands are set to 30, 100000, and 120 respectively.

- *Readonly* tests performance of the database when it comes to reading from the database, which consists of different types of SELECT queries.

```
sysbench oltp_read_only --db-driver=mysql \
--mysql-host=$maxscalehost \
--mysql-user=root --db-ps-mode=disable \
--events=0 --time=$runtime \
--tables=$tblcount --table-size=$tblsize \
--threads=$thrdcount run
```

- *Writeonly* tests performance while doing database writes, a mix of INSERT, UPDATE, and DELETE operations.

```
sysbench oltp_write_only --db-driver=mysql \
--mysql-host=$maxscalehost \
--mysql-user=root --db-ps-mode=disable \
--events=0 --time=$runtime \
--tables=$tabl_count \
--threads=$thrdcount run
```

- *ReadWrite* tests performance of the database when doing both reads and writes. By default, Sysbench OLTP ReadWrite test consists of 70% read and 30% write operations.

```
sysbench oltp_read_write --db-driver=mysql \
--mysql-host=$maxscalehost \
--mysql-user=root --db-ps-mode=disable \
--events=0 --time=$runtime \
--tables=$tblcount --table-size=$tblsize \
--threads=$thrdcount run
```

- *Deleteonly* test determines the amount of purging/delete the database setup can handle.

```
sysbench oltp_delete --db-driver=mysql \
--mysql-host=$maxscalehost \
--mysql-user=root --db-ps-mode=disable \
--events=0 --time=$runtime \
--tables=$tblcount --table-size=$tblsize \
--threads=$thrdcount run
```

- *Bulkinsert* test is used to benchmark the ability of the HA database setup to perform multi-row inserts. This test helps finding how fast data can be inserted given the replication lag kick in.

```
sysbench bulk_insert --db-driver=mysql \
--mysql-host=$maxscalehost \
--mysql-user=root --db-ps-mode=disable \
--events=0 --time=$runtime \
--threads=$thrdcount run
```

Tests are run with a different number of threads (in multiples of 8) to study the performance patterns with the increasing number of threads. Thread size of 64 is found to reach the limit to communicate and sync data and break down in Bulkinsert test, in case of VM setup, and therefore, we stopped there.

Evaluation metrics: The two most widely used evaluation metrics, namely throughput, and response time, are used to evaluate performance of the two HA database setups.

- *Throughput* is defined as the number of transactions per second. A database transaction is a unit of tasks performed within a database, which must be atomic (all the changes are either committed or rolled back), consistent, isolated, and durable, referred to as ACID (Jacobs and Satran, 2018).

- *Response time* or *latency* is measured as an average time (in millisecond) taken to complete a task, an event in the Sysbench tests.

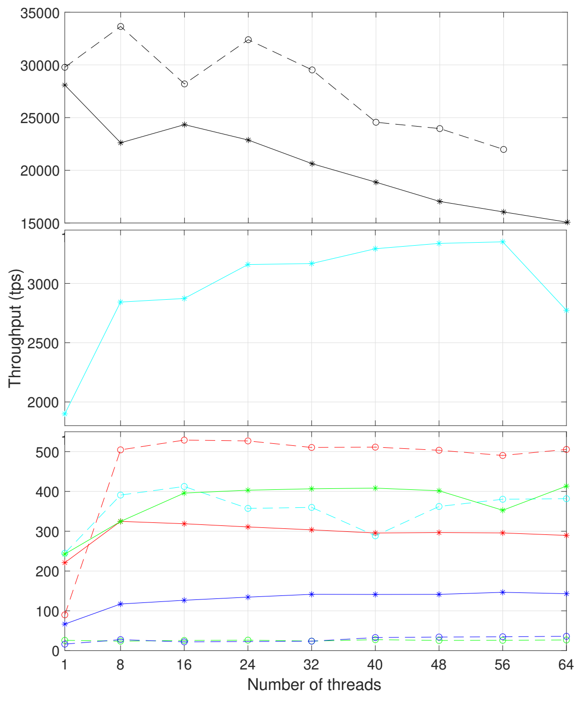
4 RESULTS AND DISCUSSION

Results from the five benchmark tests on the two HA database setups are given in terms of the two evaluation metrics ,throughput and response time, in Figure 5. The figure shows the average metric values for the different number of threads from 1 to 64 in multiples of 8. Tests failed with the VM setup when the number of threads equals 64, hence no result available in this case.

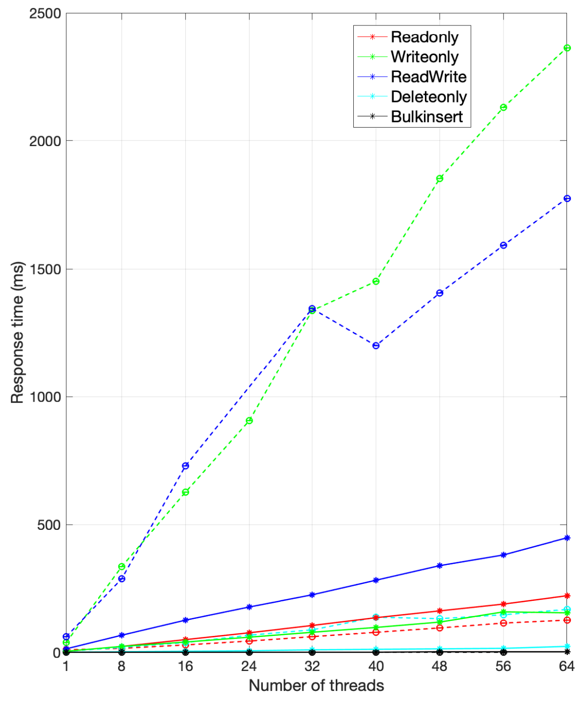
Results show that throughput increased in all the tests except Bulkinsert when the number of threads (or thread count) is increased from 1 to 8. However, it remains more or less unchanged when the thread count increased further. This is true in both HA database setups. The performance is higher in case of container setup compared to VM setup in Writeonly, ReadWrite, and Deleteonly tests. However, Readonly and Bulkinsert show the opposite performance. The difference is small in the Readonly test but relatively bigger in Bulkinsert test. VMs ability to work in isolation whereas scheduling of host's resources in case of containers could possibly contribute to this difference. Moreover, Bulkinsert performance is much higher compared to other tests, but the performance decreases with the increase in the number of threads. Higher throughput is explained by the fact that bulk insertion is more efficient compared to individual writes when it comes to writing into the disk. The decrease in throughput with the increase in thread count is possibly due to internal contention and rowlocks.

In terms of response time, performance increases with the increase in the number of threads almost linearly. This is anticipated as the number of threads increases, they have to wait for long for the resources and get hold of them as scheduled by the OS. Like with throughput, container setup shows better (lower) response time with Writeonly, Readewrite, and Deleteonly tests, compared to VM setup. Likewise, in Readonly and Bulkinsert tests, performance is reversed, but by a relatively small margin.

Vadim Tkachenko (Tkachenko, 2012) stated that using throughput only may not reflect the true performance and highlighted the importance of response time. He suggested a benchmark test model where a transaction rate is assumed and given the rate, 95% or 99% response times are measured. The benchmark tests done here is based on a stress test where the rate is not limited as the aim is to push the database as much as possible. Based on throughput and response time, choosing the thread count of 16 as an optimal value (see Figure 5), the stability of



(a) Throughput (Transactions per second).



(b) Response time or latency.

Figure 5: Performance results of the five benchmark tests with the two HA database setups [Solid line - Container setup, Dashed line - VM setup].

response times can be studied from a plot of 95% response time measured at a regular interval. Figure 6 shows a plot based on a measurement at an interval of five milliseconds. The plot shows that response time in VM setup is pretty unstable, with Writeonly and ReadWrite tests. In the meantime, the container setup has reasonably good stability in all the tests. Results are found to be similar with thread count values of eight and sixteen as well. Quantitative results of the performance of the two HA database setups on different benchmark tests with sixteen thread count is given in Table 1.

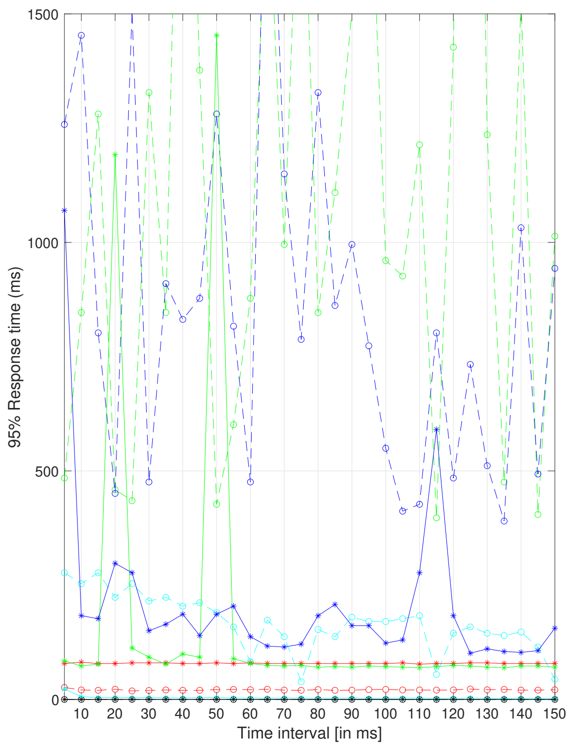


Figure 6: 95% response time as measured at every 5-second interval in the tests with thread counts set to an optimal value of 16. Response time higher than 1500ms is clipped in the plot for better visibility in the lower region.

It is to be noted here that the performance comparison is made based on the OLTP benchmark tests, which is designed to make it close to common web applications. However, it may not truly represent all kinds of real-world applications. Secondly, experiments are carried out using small VMs, which is not necessarily the case in many large applications. However, we can anticipate a similar pattern (relative results) irrespective of the VM size.

Table 1: Comparative performance results (in terms of throughput and response time) between the VM-based and the container-based HA database setups on five different benchmark tests when the number of threads is set to 16. Numbers shown in green indicate better performance over the corresponding ones shown in red.

Test	VM-based HA database		Container-based HA database	
	Throughput (tps)	Response time (ms)	Throughput (tps)	Response time (ms)
Readonly	504	15.85	325	24.62
Writeonly	24	336.06	325	24.64
ReadWrite	28	288.65	117	68.27
Deleteonly	391	20.46	2843	2.81
Bulkinsert	33648	0.21	22610	0.35

It has been found that the Galera cluster provides a highly available database with guaranteed consistency. However, it's response time is not as low as in master-slave replication (Shrestha, 2017). Therefore, it is possibly not the right solution if one needs very low latency or even real-time behavior. A list of other limitations of Galera cluster can be found in (MariaDB Galera, 2019b).

5 CONCLUSIONS

The paper presented a comparative study of the performance of highly available Galera cluster-based database setup in Docker containers and OpenStack virtual machines. The results from benchmark tests show that container-based setup performs better in most of the tests such as ReadWrite, Writeonly and Deleteonly tests that correspond to real-world applications that require both reading, writing, and deleting data in the database. VM setup performs better in Readonly and Bulkinsert tests. The results, therefore, support the wider belief that container, in general, outperforms virtual machines.

REFERENCES

Bernstein, D. (2014). Containers and Cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing*, 1(3):81–84.

Brewer, E. (2012). Pushing the CAP: Strategies for Consistency and Availability. *Computer*, 45(2):23–29.

Curino, C., Jones, E., Zhang, Y., and Madden, S. (2010). Schism: A workload-driven approach to database replication and partitioning. *Proc. VLDB Endow.*, 3(1-2):48–57.

Earl, L. and Oderov, S. (2003). Database replication system. US Patent App. 10/426,467.

Elnikety, S., Pedone, F., and Zwaenepoel, W. (2005). Database replication using generalized snapshot isolation. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, pages 73–84.

Felter, W., Ferreira, A., Rajamony, R., and Rubio, J. (2015). An updated performance comparison of virtual machines and linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 171–172.

He, S., Guo, L., Guo, Y., Wu, C., Ghanem, M., and Han, R. (2012). Elastic application container: A lightweight approach for cloud resource provisioning. In *2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, pages 15–22.

Hvasshovd, S. O., Torbjørnsen, O., Bratsberg, S. E., and Holager, P. (1995). The clustra telecom database: High availability, high throughput, and real-time response. In *Proceedings of the 21th International Conference on Very Large Data Bases*, VLDB '95, pages 469–477, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

IBM Cloud (2019). LAMP stack. `https://www.ibm.com/cloud/learn/lamp-stack-explained`. Last access: Feb. 2020.

Jacobs, M. and Satran, M. (2018). What is transaction? `https://docs.microsoft.com/en-us/windows/win32/ktm/what-is-a-transaction?redirectedfrom=MSDN`. Last access: Feb. 2020.

Kang, H., Le, M., and Tao, S. (2016). Container and microservice driven design for cloud infrastructure devops. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pages 202–211.

Kopytov, A. (2014). Sysbench manual. `https://imysql.com/wp-content/uploads/2014/10/sysbench-manual.pdf`. Last access: Feb. 2020.

Kopytov, A. (2019). Sysbench. `https://github.com/akopytov/sysbench/`. Last access: Feb. 2020.

Kozhirbayev, Z. and Sinnott, R. O. (2017). A performance comparison of container-based technologies for the cloud. *Future Generation Computer Systems*, 68:175 – 182.

Ladin, R., Liskov, B., Shrira, L., and Ghemawat, S. (1992). Providing high availability using lazy replication. *ACM Trans. Comput. Syst.*, 10(4):360–391.

Mardan, A. A. A. and Kono, K. (2016). Containers or hypervisors: Which is better for database consolidation? In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 564–571.

MariaDB Galera (2019a). MariaDB Galera Cluster. `https://mariadb.com/kb/en/library/what-is-mariadb-galera-cluster/`, `http://galeracluster.com/`. Last access: Feb. 2020.

MariaDB Galera (2019b). MariaDB Galera Cluster - Known limitations. `https://mariadb.com/kb/en/mariadb-galera-cluster-known-limitations/`. Last access: Feb. 2020.

MariaDB MaxScale (2019). MariaDB MaxScale. `https://mariadb.com/kb/en/maxscale/`. Last access: Feb. 2020.

MaryJoy, A. (2015). Performance comparison between linux containers and virtual machines. In *2015 International Conference on Advances in Computer Engineering and Applications*, pages 342–346.

MySQL (2019). MySQL Cluster CGE. `https://www.mysql.com/products/cluster/`. Last access: Feb. 2020.

Nayyar, A. (2018). Why mariadb scores over mysql. `https://opensourceforu.com/2018/04/why-mariadb-scores-over-mysql/`. Last access: Feb. 2020.

OpenStack.org (2019). OpenStack. `openstack.org`. Last access: Feb. 2020.

Seo, K.-T., Hwang, H.-S., Moon, I.-Y., Kwon, O.-Y., and Kim, B.-J. (2014). Performance comparison analysis of linux container and virtual machine for building cloud. *Networking and Communication*, 66(25):105–111.

Seravo (2015). 10 reasons to migrate to MariaDB. `https://seravo.fi/2015/10-reasons-to-migrate-to-mariadb-if-still-using-mysql`. Blog, Last access: Feb. 202.

Shrestha, R. (2017). High availability and performance of database in the cloud: Traditional Master-slave replication versus modern Cluster-based solutions. In *Proceedings of the 7th International Conference on Cloud Computing and Services Science*, CLOSER 2017, pages 413–420, Portugal. SCITEPRESS - Science and Technology Publications, Lda.

Tkachenko, V. (2012). Introducing new type of benchmark. `https://www.percona.com/blog/2012/02/25/introducing-new-type-of-benchmark/`. Last access: Feb. 2020.

Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., and Alonso, G. (2000). Database replication techniques: a three parameter classification. In *Reliable Distributed Systems, 2000. SRDS-2000. Proceedings of The 19th IEEE Symposium on*, pages 206–215.

Wiesmann, M. and Schiper, A. (2005). Comparison of database replication techniques based on total order broadcast. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):551–566.

Xing, Y. and Zhan, Y. (2012). Virtualization and cloud computing. In Zhang, Y., editor, *Future Wireless Networks and Information Systems*, pages 305–312, Berlin, Heidelberg. Springer Berlin Heidelberg.

Zhang, Q., Liu, L., Pu, C., Dou, Q., Wu, L., and Zhou, W. (2018). A comparative study of containers and virtual machines in big data environment. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 178–185.