High Availability and Performance of Database in the Cloud
Traditional Master-slave Replication versus Modern Cluster-based Solutions

Raju Shrestha
Oslo and Akershus University College of Applied Sciences, Oslo, Norway

Keywords: High Availability, Performance, Cloud, Database Replication, Master-slave, Galera Cluster, MariaDB.

Abstract: High availability (HA) of database is critical for the high availability of cloud-based applications and services. Master-slave replication has been traditionally used since long time as a solution for this. Since master-slave replication uses either asynchronous or semi-synchronous replication, the technique suffers from severe problem of data inconsistency when master crashes during a transaction. Modern cluster-based solutions address this through multi-master synchronous replication. These two HA database solutions have been investigated and compared both qualitatively and quantitatively. They are evaluated based on availability and performance through implementation using the most recent version of MariaDB server, which supports both the traditional master-slave replication, and cluster based replication via Galera cluster. The evaluation framework and methodology used in this paper would be useful for comparing and analyzing performance of different high availability database systems and solutions, and which in turn would be helpful in picking an appropriate HA database solution for a given application.

1 INTRODUCTION

In computing, high availability (HA) refers to a system or service or component that is continuously operational for a desirably long length of time. Availability is often expressed as expected system uptime (in percentage) in a given period of time (such as in a week or a year). 100% availability indicates that the system is "always on" or "never failing". For instance, 90% availability (one nine) in a period of one year means the system can have up to 10%, i.e., 36.5 days of down time.

There are basically three principles, which can help achieve high availability. They are (a) elimination of single point of failure (SPOF) by adding redundancy, (b) reliable crossover from a failed to a standby component, and (c) detection of failures as they occur (Piedad and Hawkins, 2001).

In today's always-on, always-connected world, high availability is one of the important quality attributes of a cloud-based application or service. High availability of all the system components such as the application itself, storage, database etc. contributes to the high availability of the entire application. Since most of the cloud-based applications and services, in general, are dynamic database-driven web-based applications, database plays an important role in the system's high availability. Therefore, the focus of this paper is on high availability database.

Traditionally, master-slave (single master, multiple slaves) based database replication techniques (Ladin et al., 1992; Wiesmann et al., 2000; Earl and Oderov, 2003; Wiesmann and Schiper, 2005; Curino et al., 2010) is being used as a solution for high availability databases since more than 15 year. Most recently, multi-master cluster-based database replication techniques such as MariaDB Galera cluster (Galera Cluster, 2016), MySQL cluster (MySQL, 2016) have been made available as more effective alternatives.

In this paper we have studied and evaluated master-slave and cluster-based HA database solutions, qualitatively as well as quantitatively. In our study, the most recent version of MariaDB server at the time (v10.1) that has built-in Galera cluster support is used. MariaDB was chosen because it is not only free, open and vibrant, but also has more cutting edge features and storage engines, compatible with MySQL, performs better, and easy to migrate (Seravo, 2015). Performance comparison has been made based on the benchmark results obtained on an OpenStack/DevStack cloud platform.

Shrestha, R.
High Availability and Performance of Database in the Cloud - Traditional Master-slave Replication versus Modern Cluster-based Solutions.
DOI: 10.5220/0006294604130420
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 385-392
ISBN: 978-989-758-243-1

2 TWO MAJOR HIGH AVAILABILITY DATABASE SOLUTIONS

HA databases use an architecture which is designed to continue working normally even in case of hardware or network failures within the system and end users do not experience any interruption or degradation of service. HA database aims not only for reduced downtime, but also for reduced response time and increased throughput (Hvasshovd et al., 1995). HA database cannot be retrofit into a system; rather it should be designed in the early stages to provide minimal downtime, optimal throughput and response time, in order to maximize customer satisfaction.

Different applications can have very different requirements for high availability. For instance, a mission critical system needs 100% availability with 24x7x365 read & write access while many other systems are better served by a simpler approach with more modest high availability ambitions. Adding high availability increases complexity and cost. Therefore, an appropriate solution needs to be chosen for a given application at hand.

Various techniques and solutions have been proposed and used for HA databases. Widely used open-source databases such as MariaDB and MySQL has an array of high availability solutions ranging from simple backups, through replication and shared storage clustering all the way up to 99.999% availability (i.e. 5mins. of downtime per year), shared nothing, geographically replicated clusters.

In general, HA database solutions can be broadly categorized into two types: master-slave (master-replica) architecture and multi-master cluster-based architecture. The two architectures offer different guarantees on Brewer's CAP theorem, which states that it is impossible for a distributed system to simultaneously provide more than two out of three guarantees among consistency, availability, and partition tolerance (Brewer, 2012). The two architectures are described in the following two subsections.

2.1 Master-slave Architecture

Master-slave architecture is a traditional solution, which is commonly used since long time in the database world, where one master handles data writes and multiple slaves are used to read data. In many applications, a large number of end users use services, which require data reading from the database and small number (mostly by system administrators) does data writing for database updates. Such systems use read-write splitting mechanism whereby a primary database server (master) is used for data writing while slave multiple database servers (slaves) are load balanced for data reading. The architecture thus supports read scalability as the number of slaves can be scaled out easily in a cloud according to needs at a given time. Figure 1 illustrates a master-slave architecture. Most of the popular databases such as MariaDB, MySQL, Microsoft SQL Server support master-slave architecture.

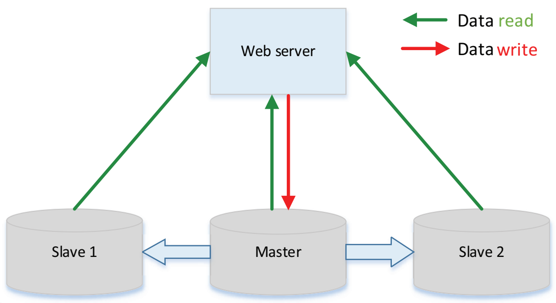

Figure 1: Master-slave architecture.

Whenever there is a change in the master database, the change is applied in all the slaves, the process known as database replication. The replication process is normally asynchronous (Wiesmann et al., 2000; Elnikety et al., 2005), which means the master does not check to confirm if slaves are always up-to-date. Asynchronous replication does not guarantee about delay between applying changes on master node and propagation of changes to slave nodes. Therefore, slave databases are always bit outdated. Moreover, as the delay could be short and long, some latest changes may be lost, when the master crashes. This may cause data inconsistencies when something goes wrong in the master in the middle of a transaction. Newer versions of widely adopted open-source databases such as MariaDB and MySQL provide support for semi-synchronous replication, which makes sure at least one slave is synchronized with the master. After committing a transaction, master waits for an acknowledgment from that slave. However, this still does not prevent from data inconsistencies in other slaves. Therefore, according to the CAP theorem, master-slave replication provides guarantee of availability and partition tolerance at the sacrifice of consistency.

Master-slave replication does not provide high availability of the master itself. It is used along with a tool such as MariaDB replication manager (MRM) (MariaDB, 2016) and master high availability (MHA) (Matsunobu, 2016) in order to promote a slave to a master (automatic failover) when master fails. These tools keep monitoring master and if the master server goes down, it quickly promotes a slave as the new master. It will also switch the IP of the old master to the new master, so that applications do not need to be reconfigured. In theory, master-slave HA database solution looks quite simple and straightforward. However, in practice, the transfer of the master role during failover might result in some downtime.

2.2 Cluster-based Multi-master Architecture

This is a relatively new solution where HA database is often achieved through a multi-master architecture that uses clustering, in which multiple servers (called nodes) are grouped together. Any node within a cluster can respond to both read and write requests. Change of data in a node is replicated across all nodes in the cluster instantly, thus providing system redundancy and minimizing the possibility of downtime. Figure 2 illustrates a cluster-based architecture.

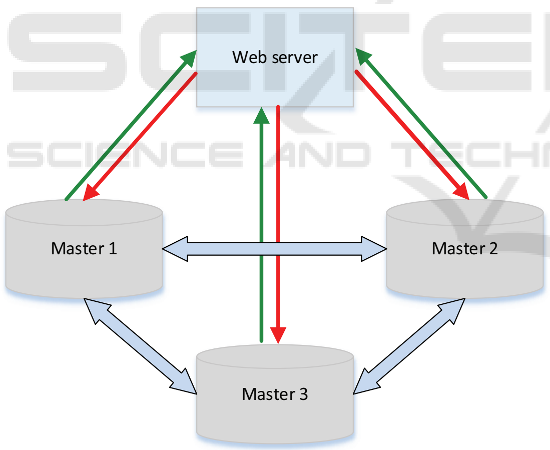

Figure 2: Cluster-based multi-master architecture.

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 maintains data consistency. However, in practice, synchronous database replication has traditionally been implemented via the so-called *2-phase commit* or distributed locking, which proved to be very slow. Because of low performance and complexity of implementation of synchronous replication, asynchronous replication has long been remained as the dominant solution for high availability database.

MySQL cluster (MySQL, 2016) stores a full copy of the database in multiple servers so that each server will be independently function as the master database. But this method results in data conflicts when there is a network connection problem. Galera cluster addresses this problem through conditional updates whereby each server accepts new updates only if it can reach a majority of other servers. It uses active eager replication based on the optimistic approach (Gautam et al., 2016) to ensure local consistency, and handles data write conflicts using certification-based replication based on group communication and transaction ordering techniques (Pedone et al., 1997; Pedone, 1999; Kemme and Alonso, 2000) to enforce global consistency. Galera cluster replication combines these techniques to provide a highly available, transparent, and scalable synchronous replication solution. According to the CAP theorem, Galera cluster guarantees consistency and availability of the database. We have used MariaDB Galera cluster in our study.

2.3 Selection of a Database Server

In both master-slave and cluster-based replication setups, there should be a way to determine which database server is to be selected for a database access (or transaction). In general, scaled-out database servers are load balanced. There are mainly two ways database selection is implemented. First method is to handle the selection mechanism at the application tier, and the second method is to handle it separately using a database proxy.

An example of the first method is to handle the database selection using the PHP MySQL Native driver (PHP-MySQLnd) and its master-slave plugin MySQLnd_MS (Shafik, 2014). This is applicable for master-slave architecture only, and MySQLnd_MS is so far not yet available for newer versions of PHP including the current version 7. Load-balancing and failover strategies are defined in MySQLnd_MS configuration. The plugin redirects database query to a database server based on read-write splitting, where all write access is directed to master and all read statements to a slave selected based on a load balancing strategy. Automatic master failover is done based on failover strategy defined in the configurations. An advantage of this method is that it does not require any additional hardware.

However, there are several disadvantages with this method. A conceptual drawback of this method is that handling database selection in the application side (or tier) blurs the separation between application and database. Read-write splitting with MySQLnd_MS is naive as it assumes all queries starting with SELECT as read statements, but there could be write statements starting with SELECT. Master failover is not instantaneous as it takes some time (which might be several minutes) to determine if the master is failed before promoting a slave to master, and this causes database unavailability for that time. Since all the web servers (PHP servers) needs to be configured for MySQLnd_MS plugin, this could be annoying when one need to change environment, for example when adding a new slave. A possible solution to minimize the changes that need to be done on the PHP-MySQLnd configuration file is to use a proxy server such as HAProxy between PHP driver and database together with PHP-MySQLnd (Severalnines, 2016). Proxy server provides a single point database access to the application. Since HAProxy is not SQL-aware, MySQLnd_MS is required to understand the type of SQL statement.

A better approach is to use a SQL-aware proxy server such as MaxScale, which not only provides a single point access to the application but it also fully decouples application and database processes. The proxy server redirects a SQL query to an appropriate database server/node. The method is applicable for both master-slave and cluster-based database setups. MaxScale, which is considered a next generation database proxy, is an open source project developed by MariaDB. It can load balance both connections and individual database requests, filter requests according to a set of defined rules, and monitor availability, utilization, and load on back-end servers to prevent requests from going to overburdened and unavailable resources (Trudeau, 2015; MaxScale, 2016). We use database proxy method using MaxScale in our implementation.

3 EXPERIMENTAL SETUP

Experiments are performed in a single machine OpenStack cloud setup using DevStack. DevStack is installed in a VMware virtual machine (VM) on a MacBook Pro machine. VMware was chosen as it supports nested virtualization. The specification of the computer and the VM used is given below.

Specification of the computer:

- CPU: Intel(R) Core i7 3.1GHz processor
- Memory: 16GB
- OS: MacOS Sierra

Specification of the VMware VM:

- CPU: 2 processor cores
- Memory: 9GB
- Hard disk: 50GB
- Network: two Network Interface Cards (NICs), one with NAT to get access to the Internet via host machine and one with vmnet2 which is used as a public network interface for accessing VMs from outside
- Advanced options: VT-x/EPT and code profiling enabled
- OS: Ubuntu 16.04 LTS (Xenial Xerus)

Eight servers were provisioned in the OpenStack cloud for a widely used LAMP stack (Linux, Apache, MySQL, PHP) based cloud service (Infrastructure as a Service-IaaS) architecture for a high availability dynamic database-driven web application, which consists of two Apache2 web servers that are load balanced with two HAProxy 1.6 load balancers, three MariaDB database servers, and a MariaDB MaxScale server, all running Ubuntu 16.04 (Xenial Xerus) from the cloud image available on the web (Ubuntu, 2016). Figure 3 depicts the cloud system setup. All the three database servers are installed with MariaDB server v10.1, which comes with built-in Galera cluster support. MaxScale v2.1 server is installed and used as a database proxy server in both the master-slave and Galera cluster setups.

In the master-slave database replication setup, one server is used as the master and the other two servers are used as slaves. Global transaction id (GTID) based replication is used as it has many benefits compared to the traditional bin-log based replication (Combaudon, 2013). Automatic master failover mechanism is implemented with MariaDB replication manager (MRM). In the Galera cluster-based setup, MariaDB 10.1 installed in the three database servers are configured to use Galera cluster. The three servers then act as three nodes of a cluster. If there is any change in one node, the changes will be updated in the other two nodes synchronously. Synchronization method is set to use rsync. In both setups, slaves are load balanced with equal server weights.

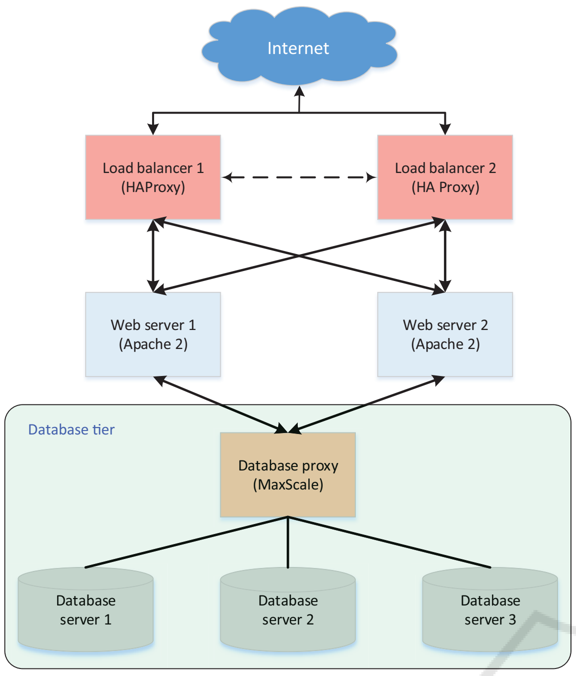

Figure 3: Cloud setup used in experiments.

A widely used open source benchmark suite, Sysbench (Sysbench, 2016), specifically OLTP (Online Transaction Processing) benchmark, is used to evaluate and compare performance of the two setups under intensive loads, using the same set of parameters. Sysbench v0.5 is installed and executed from the DevStack VM. All database is thus accessed through the MaxScale proxy server.

4 EXPERIMENTAL RESULTS

Test database is prepared with 50 tables of size 10000, Pareto distribution for an OLTP test using Sysbench. Then test runs are executed for readonly and read-write with different number of threads n (which corresponds to concurrent clients), for $n = 1, 8, 16, 32, 64, 72, 80$. The three Sysbench commands used for test database preparation, read-only tests, and read-write tests respectively are given below. OLTP read-write benchmark simulates a real-world usage consisting of 70% read (SELECT) and 30% write (INSERT, UPDATE, DELETE) operations.

```
sysbench --test=/usr/share/doc/sysbench/tests/
    db/oltp.lua --oltp_tables_count=50 --
    oltp_table_size=10000 --mysql-host=
    maxscale --mysql-user=root --rand-type=
    pareto --rand-init=on prepare
```

```
sysbench --test=/usr/share/doc/sysbench/tests/
    db/oltp.lua --oltp-read-only=on --oltp-
    reconnect=on --oltp-reconnect-mode=random
    --num-threads=n --mysql-host=maxscale --
    mysql-user=root --max-requests=5000 --rand
    -type=pareto --rand-init=on run
```

```
sysbench --test=/usr/share/doc/sysbench/tests/
    db/oltp.lua --oltp-read-only=off --oltp-
    test-mode=complex --oltp-reconnect=on --
    oltp-reconnect-mode=random --num-threads=n
     --mysql-host=maxscale--mysql-user=root --
    max-requests=5000 --rand-type=pareto --
    rand-init=on run
```

Performance is measured in terms of throughput (transactions per second [tps]) and response time (or latency), for both read-only and read-write tests. For statistically robust results, tests are repeated for twenty times. Figure 4 and 5 show plots from the test results of the readonly and read-write tests on both HA database solutions. Figure 4 shows the performance in terms of throughput, whereas Figure 5 shows the performance in terms of response time. Mean and standard error of the mean (SEM) are computed from the twenty tests. The plots show the mean values, along with the lower and the upper bounds of the 95% confidence intervals shown as vertical lines bounded by shaded areas. Confidence intervals are computed as $Mean - 1.96 \times SEM$ and $Mean + 1.96 \times SEM$ (assuming normal probability distribution), where SEM is given by $Std./\sqrt{N}$; $Std.$ being the standard deviation and N the sample size (Moore et al., 2010).

From the performance plots, we see that in readonly tests, throughput increases sharply when the number of threads increases from 1 to 16, then the increment slows down and drops on further increasing the number of threads beyond 64. This trend is similar in both the master-slave and Galera cluster based implementations. The trend also follows in read-write tests but changes occur at different number of threads (8 and 16). Drop of performance after certain number of threads in general occurs because of internal contention and row locks. Average response time increases with more or less linearly with the increase in the number of threads, indicating a consistent increase in latency under an increasing load. Both HA solutions have similar performance in readonly tests. However, master-slave setup shows better performance both in terms of throughput and response time in read-write tests.

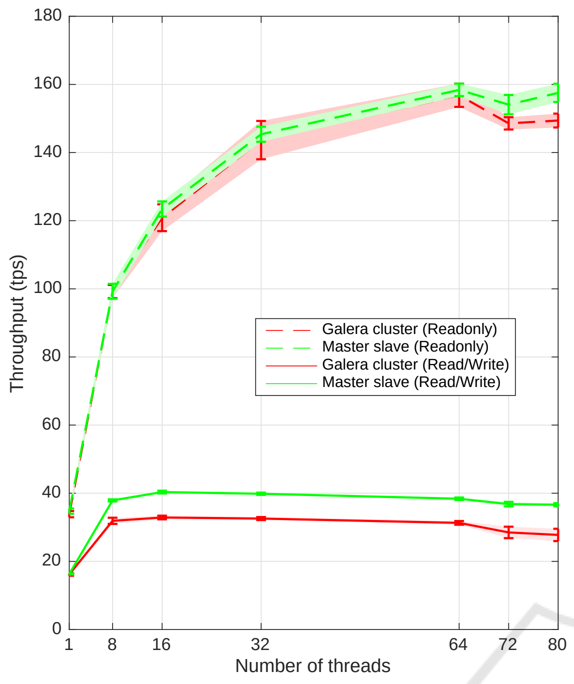

Figure 4: Throughputs from tests with different number of threads.

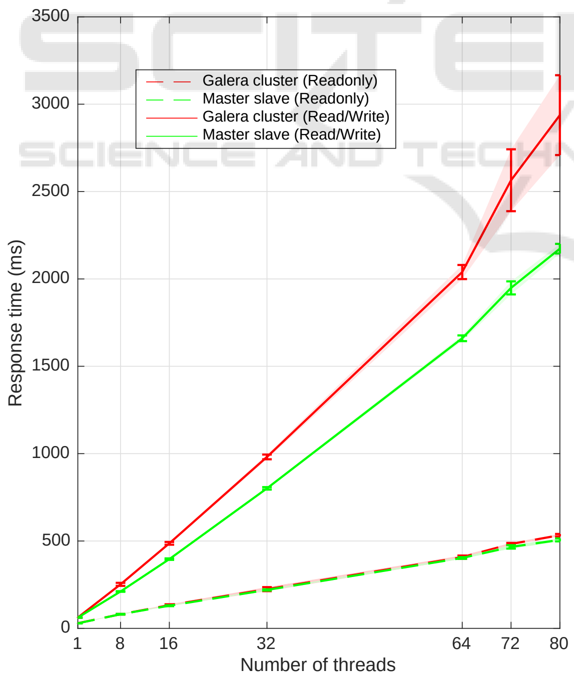

Figure 5: Response time from tests with different number of threads.

Next, the two setups were investigated for their behaviors under various failure scenarios. In master-slave setup, when one or both the slaves were down, the system continued to work as expected as both data read and write was provided by the live master. When the two slaves were working but the master was crashed, it took significant time (more than one minute) for MariaDB replication manager to check if the master is down and then promote a slave to the new master. However, in Galera setup, the system worked smoothly without showing any failure-related symptoms even when one or two nodes were crashed.

5 COMPARATIVE ANALYSIS AND DISCUSSION

In this section, we analyze, compare and discuss the two HA database solutions, qualitatively and quantitatively.

Qualitative Analysis: Because of loosely coupled master-slave nature in asynchronous replication, slave can be arbitrarily behind the master in applying changes, and hence read on a slave can give old data. More seriously, upon the master's failure, slave may not have latest committed data resulting in data loss. This might stall failover to a slave until all transactions have been committed as it is not instantaneous. This is true in the case of semi-synchronous replication as well even though it guarantees one slave to be synchronized with the master. Read scale-out is straightforward, in master-slave setup, but write scale-out is not.

Synchronous replication in a cluster-based setup guarantees all slaves to receive and commit changes in the master and this in turn guarantees latest data to be read from any slave. Master failover to a slave is integrated and automatic, which makes sure data writes to continue on new master almost instantaneously. Therefore, this high availability solution is more effective compared to the master-slave replication based solution. Moreover, the setup supports both read and write scalability.

In both master-slave and cluster-based solutions, replication overhead increases with the number of nodes/slaves present in the system. Failure test results show that cluster-based setup do not suffer from any failure related hiccups unlike with master-slave setup as failure situations are handled smoothly and transparently.

Quantitative Analysis: Quality of a database system is measured quantitatively by high availability, throughput, and response time (Hvasshovd et al., 1995). In this paper, we compare the two HA database solutions based on throughput and response time under heavy database loads.

From the experimental results, we see that throughput is much higher and response time is much lower from readonly tests compared to those read-write tests, in both HA setups. This is expected as data writing takes significantly longer time than data reading. Moreover, data needs to be replicated or synchronized after writing in the master, which takes some time. In the case of read-write tests, throughput from the Galera cluster setup was lower compared to the master-slave setup. This is because there is significant overhead due to synchronous replication in the former setup. This is also reflected by the higher response time. However, both HA solutions show similar performance for readonly tests, since there is no synchronization required and hence no delay in readonly tests,

Based on both the qualitative and the quantitative analysis of the two HA database solutions, Table 1 provides a comparative summary.

Table 1: Comparison table between master-slave and cluster-based HA database solutions. Texts shown in green indicate advantageous over the one shown in red.

	Master-slave HA solution	Galera cluster-based solution
Replication type	Asynchronous or semi-synchronous	Synchronous
Automatic failover	Need to use a tool such as MRM, MHA	Built-in
Extra hardware	Can be implemented without one	Required for at least one database proxy
Minimum number of servers	1	3
Failover delay	Not instantaneous	Instantaneous
Data loss	Possible	No
Data inconsistency	Possible	No
Scale-out	Read scale-out	Both read and write scale-out
Availability	Lower	Higher
Throughput	Higher	Lower
Response time	Lower	Higher

It is important to note the limitations of the experiments carried out in this work. Firstly, even though OLTP benchmark is designed to make it close to a common real-world scenario, it may not truly represent real-world application. Secondly, experiments are carried out in a single machine OpenStack cloud setup using DevStack within a VMware virtual machine and role of virtualization has not been considered. However, it can be expected that the relative results obtained reflect the performance and behavior that is good enough to have a general impression about the two HA solutions.

This work can be considered our first step towards the study of major HA database solutions. Future work could be to extend this with an extensive study of various other major solutions as well including database sharding techniques, which has gain popularity over the past several years. The study could include more testing and detailed failover scenarios.

6 CONCLUSIONS

This paper investigated effectiveness of the two major solutions to high availability database solutions: traditional master-slave replication and modern cluster-based techniques. Performance evaluation of both solutions implemented using MariaDB 10.1 with Galera cluster has shown that traditional master-slave replication solution performs equal or better in terms of throughput and response time. Because of simpler setup and better performance, this method is still being widely used. However, cluster-based solution is superior when it comes to high availability, data consistency and scalability as it offers instantaneous failover, no data inconsistency and loss of data, and at the same time providing both read and write scalability. Therefore, despite some performance lag, Galera cluster is an effective solution for applications and services where data consistency and high availability is critical.

REFERENCES

Brewer, E. (2012). Pushing the cap: Strategies for consistency and availability. *Computer*, 45(2):23–29.

Combaudon, S. (2013). Replication in MySQL 5.6: GTIDs benefits and limitations - Part 1 & 2. https://www.percona.com/blog/2013/05/21/replication-in-mysql-5-6-gtids-benefits-and-limitations-part-1/, https://www.percona.com/blog/2013/05/30/replication-in-mysql-5-6-gtids-benefits-and-limitations-part-2/. Blog.

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.

Galera Cluster (2016). MariaDB Galera Cluster. https://mariadb.com/kb/en/mariadb/galera-cluster/, http://galeracluster.com/. Last access: Nov. 2016.

Gautam, B. P., Wasaki, K., Batajoo, A., Shrestha, S., and Kazuhiko, S. (2016). Multi-master replication of enhanced learning assistant system in IoT cluster. In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pages 1006–1012. IEEE.

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.

Kemme, B. and Alonso, G. (2000). Don’t be lazy, be consistent: Postgres-R, a new way to implement database replication. In *VLDB*, pages 134–143.

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

MariaDB (2016). MariaDB Replication Manager. https://github.com/tanji/replication-manager. Last access: Dec. 2016.

Matsunobu, Y. (2016). MySQL Master High Availability (MHA). https://code.google.com/p/mysql-master-ha/. Last access: Dec. 2016.

MaxScale (2016). MariaDB MaxScale. https://mariadb.com/products/mariadb-maxscale. Last access: Nov. 2016.

Moore, D., McCabe, G. P., and Craig, B. (2010). *Introduction to the Practice of Statistics*. W. H. Freeman, 7th edition.

MySQL (2016). MySQL Cluster. https://www.mysql.com/products/cluster/. Last access: Dec. 2016.

Pedone, F. (1999). *The database state machine and group communication issues*. PhD thesis, École Polytecnique Fédérale de Lausanne, Switzerland.

Pedone, F., Guerraoui, R., and Schiper, A. (1997). Transaction reordering in replicated databases. In *Reliable Distributed Systems, 1997. Proceedings., The Sixteenth Symposium on*, pages 175–182.

Piedad, F. and Hawkins, M. (2001). *High availability: Design, techniques and processes*. Prentice Hall.

Seravo (2015). 10 reasons to migrate to MariaDB. https://seravo.fi/2015/10-reasons-to-migrate-to-mariadb-if-still-using-mysql. Blog.

Severalnines (2016). High availability read-write splitting with PHP-MySQLnd, MySQL replication and HAProxy. http://severalnines.com/blog/high-availability-read-write-splitting-php-mysqlnd-mysql-replication-and-haproxy. Last access: Nov. 2016.

Shafik, D. (2014). Easy read/write splitting with PHPs MySQLnd. https://blog.engineyard.com/2014/easy-read-write-splitting-php-mysqlnd. Blog.

Sysbench (2016). Sysbench 0.5. http://repo.percona.com/apt/pool/main/s/sysbench/. Last access: Nov. 2016.

Trudeau, Y. (2015). MaxScale: A new tool to solve your MySQL scalability problems. https://www.percona.com/blog/2015/06/08/maxscale-a-new-tool-to-solve-your-mysql-scalability-problems/. Blog.

Ubuntu (2016). Ubuntu 16.04 LTS (Xenial Xerus). https://cloud-images.ubuntu.com/releases/xenial/release/. Last access: Nov. 2016.

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.