International Journal of Research in Computing (2820-2139)
Volume 02, No. 01, July 2023.

Performance Analysis of Docker-based Database Management Systems Compared to Virtual Machine-based Systems: A Comparative Study

WMCJT Kithulwatta[1,2], KPN Jayasena[3], BTGS Kumara[3] and RMKT Rathnayaka[4]
[1]Faculty of Graduate Studies, Sabaragamuwa University of Sri Lanka, Belihuloya, Sri Lanka
[2]Department of Information and Communication Technology, Faculty of Technological Studies, Uva Wellassa University of Sri Lanka, Badulla, Sri Lanka
[3]Department of Computing and Information Systems, Faculty of Computing, Sabaragamuwa University of Sri Lanka, Belihuloya, Sri Lanka
[4]Department of Physical Sciences and Technology, Faculty of Applied Sciences, Sabaragamuwa University of Sri Lanka, Belihuloya, Sri Lanka

Corresponding Author: W. M. C. J. T. Kithulwatta, Email: chirantha@uwu.ac.lk and chiranthajtk@gmail.com

ABSTRACT Computer virtualization is a very old technology. Due to a lot of technical barriers, computer containerization has been introduced recently. Nowadays, computer containerization is playing a major role in information technology and containerization is a trending topic. Among the practitioner of information technology, a lot of software services are moving to containerization instead of traditional virtual machines. Among the most famous software services: database management systems are a leading service. Among most computer containerization technologies, Docker is the most popular and trending container vendor. Therefore, identification of the performance of database management systems on the Docker-based platform is an essential task for the practitioner. This research study aims to identify the practical aspects of each database management system on the Docker-based infrastructure for main database management system operations. For the study: Ubuntu 18.04 Long Term Support (used package with architecture: GNU/Linux 4.15.0-112-generic x86_64) cloud-based operating system was used and on that operating system the Docker infrastructure was launched. Docker version 19.03.9 was launched for the study. On Docker: MySQL, PostgreSQL, and MongoDB database management system containers were launched separately. SELECT, DELETE, UPDATE, and INSERT operations were used for the performance evaluations of database management system response times. This research identified that there was an increase in the performance of the Docker platform with a 95% confidence interval level for all data records to virtual machine-based platforms. Finally, the research study identified that Docker-based database management system has a quick response time than virtual machine-based database management systems.

KEYWORDS: Containers, Database Management Systems, Docker, MongoDB, MySQL, PostgreSQL

I. Introduction

Computer containerization is a trending technology that brings the computing environment as a logically packaged mechanism. Packaged containers consist of all required environments with essential binaries, dependencies, libraries, and configuration files to execute any such kind of software application and/or service. Containers can be deployed in any computer environment such as a public cloud centre, private cloud centre, or any personal computer.

Within the practitioner of containers, Docker is one of the trending container management technologies. Other than Docker: Rkt and Linux containers are available as container technologies.

As mentioned in the official Docker documentation, most of the widely used computing tools are engaged with Docker containerization. A few of them are Bitbucket, GitLab, GitHub, NGINX, Redis, Jenkins, JFrog, MongoDB, Visual Studio Code, etc. [3].

Relational Database Management Systems (RDBMS) is a specific database management system specification, which is based on the relational model and Structured Query Language (SQL). Most modern database systems are RDBMSs. MySQL, PostgreSQL, IBM DB2, MS SQL Server, and Oracle are the best examples of RDBMSs [6].

Within the existing research studies, the authors have evaluated the database management systems by considering the taken time to particular SQL queries and response time commonly.

This research study aims to identify the practical aspects of each database management system on the Docker-based infrastructure for main database management system operations. By considering the response time, practically, information technology academics and practitioners will be able to select suitable database management system technology for the applications.

The overall research study provides answers to the below research questions.

RQ1: *How are the performances of relational database management system Docker containers over virtual machine-based database management systems?*

RQ2: *How are the performances of the no-SQL database management system Docker containers over virtual machine-based database management systems?*

RQ3: *How to launch no-SQL and relational database management systems on Docker and virtual machine-based infrastructures?*

II. Literature review

Server containerization has emerged with various reviews here and there. Practically, the vulnerability of data, complex criteria for resources, and network problems are often cited as drawbacks. Nevertheless, the usage of containerization has increased, since most applications are running with containerized databases which are migrated from traditional virtual machines. Software development and software infrastructure-providing organizations of all sizes (from small start-ups to multinational, proven microservices companies) are using containers. Even the containerization has been taken over by well-known companies including Google, Amazon, Oracle, and Microsoft, databases are playing a major role.

In addition to the database operating environment, the containerization of the database involves databases inside a container to allow data to be loaded onto a virtual machine and executed separately. The article [14] has mentioned four special factors that support the usage of the database in containers. Those are the usage of the same configurations or ports for all containers, resilience, resource, and storage, cluster upscale or downscale, and data locality and networking.

According to the above first factor, the containerized architecture removes some of the overhead associated with a distributed system that supports various node types. This kind of distributed applications and systems required the management of separate containers that also require multiple configurations. One kind of configuration type is supported for database containerization. As well as, resilience, resources, and storage are considered as the second factor. But containerization should not be left with data inside them. According to typical database scenarios, it is often important to have database replications or export data from a central storage system. This process makes more cost and significantly slows down the performance. Database Management Systems are executing like any other sever-side applications but they are consuming more CPU-intensive and memory-intensive, have high status, and occupy storage space. The article [14] has expressed that all principle functions are in same for containers as well. In addition to that, the states of the database engine can be controlled, resources can be limited and access to the network can be restricted.

According to the third factor, the practice expresses the ambiguity as to how effective the application will be and the volume needed by enhancing the elasticity of the network. Database containerization takes into account the elasticity of the software applications. As well as by growing and shrinking most suitable infrastructure is provided. The article [14] has presented that data can be replicated in the background by adding extra nodes to the container cluster. According to the last factor, network scaling was a major challenge within the modern virtualized infrastructure. Load balancers typically take all traffic on the first run and then distribute it to the application containers. Thereafter, application containers interact with databases that produce more traffic. Hence containerization puts the database and the application back together, by removing any network troubles.

As presented in the white paper [15], databases may be available on standard stand-alone servers, on-premise clusters, or in PaaS (Platform as a Service) cloud services such as Azure SQL databases. For the development and test environments, however, it is practical to run the databases as containers, since no external dependencies are required and the entire application is started by simply executing the docker-compose-up command. The existence of these databases as containers is also ideal for integration tests since the database is started in the container and is always filled with the same data so that tests are more predictable.

Commonly, NoSQL database management systems are suitable for geospatial data and big data environments. The authors of [16] have mentioned that MySQL is a very mature RDBMS, a popular and inexpensive option. Furthermore, MySQL is an open-source RDBMS that is distributed, developed, and supported by Oracle Corporation. RDBMSs have identified that they are with remarkable features to reform transactional updates and handle the underlying consistency issues considerably well.

III. Methodology

To evaluate and make the comparison for Docker container-based database management systems, the Docker containerized

infrastructure was launched on Ubuntu 18.04 Long Term Support (used package with architecture: GNU/Linux 4.15.0-112-generic x86_64) cloud-based operating system. The host computer was with 15 GB memory capacity and 1 Gbps network bandwidth. On that host computer infrastructure, Docker version 19.03.9 was launched. Both Docker client and server engine communities are version 19.03.9. Docker API (Application Program Interface) version was 1.40 [17].

For the experimental evaluation: 5, 50, 500, 5000, and 50000 data records were used for each database management system. For the evaluation: MySQL and PostgreSQL database management systems were used as relational database management systems. MongoDB was used as the no-SQL database management system. To access each database management system remotely, MySQL Workbench, pgadmin, and Robo 3T (formerly Robomongo) were used respectively for the MySQL, PostgreSQL, and MongoDB database management systems.

The used database schema is as follows. For the queries, join based queries were used by considering the tables *oderdetails, orders,* and *customers*.

```
Table "customers" {
  "customerNumber" int(11) [pk, not null]
  "customerName" varchar(50) [not null]
  "contactLastName" varchar(50) [not null]
  "contactFirstName" varchar(50) [not null]
  "phone" varchar(50) [not null]
  "addressLine1" varchar(50) [not null]
  "addressLine2" varchar(50) [default: NULL]
  "city" varchar(50) [not null]
  "state" varchar(50) [default: NULL]
  "postalCode" varchar(15) [default: NULL]
  "country" varchar(50) [not null]
  "salesRepEmployeeNumber" int(11) [default: NULL]
  "creditLimit" decimal(10,2) [default: NULL]


Indexes {
  salesRepEmployeeNumber [name: "salesRepEmployeeNumber"]
}
}


Table "employees" {
  "employeeNumber" int(11) [pk, not null]
  "lastName" varchar(50) [not null]
  "firstName" varchar(50) [not null]
  "extension" varchar(10) [not null]
  "email" varchar(100) [not null]
  "officeCode" varchar(10) [not null]
  "reportsTo" int(11) [default: NULL]
  "jobTitle" varchar(50) [not null]

Indexes {
  reportsTo [name: "reportsTo"]
  officeCode [name: "officeCode"]
}
}


Table "offices" {
  "officeCode" varchar(10) [pk, not null]
  "city" varchar(50) [not null]
  "phone" varchar(50) [not null]
  "addressLine1" varchar(50) [not null]
  "addressLine2" varchar(50) [default: NULL]
  "state" varchar(50) [default: NULL]
  "country" varchar(50) [not null]
  "postalCode" varchar(15) [not null]
  "territory" varchar(10) [not null]
}

Table "orderdetails" {
  "orderNumber" int(11) [not null]
  "productCode" varchar(15) [not null]
  "quantityOrdered" int(11) [not null]
  "priceEach" decimal(10,2) [not null]
  "orderLineNumber" smallint(6) [not null]

Indexes {
  productCode [name: "productCode"]
  (orderNumber, productCode) [pk]
}
}


Table "orders" {
  "orderNumber" int(11) [pk, not null]
  "orderDate" date [not null]
  "requiredDate" date [not null]
  "shippedDate" date [default: NULL]
  "status" varchar(15) [not null]
  "comments" text
  "customerNumber" int(11) [not null]

Indexes {
  customerNumber [name: "customerNumber"]
}
}


Table "payments" {
  "customerNumber" int(11) [not null]
  "checkNumber" varchar(50) [not null]
  "paymentDate" date [not null]
  "amount" decimal(10,2) [not null]

Indexes {
  (customerNumber, checkNumber) [pk]
}
}


Table "productlines" {
  "productLine" varchar(50) [pk, not null]
  "textDescription" varchar(4000) [default: NULL]
  "htmlDescription" mediumtext
  "image" mediumblob
}

Table "products" {
  "productCode" varchar(15) [pk, not null]
  "productName" varchar(70) [not null]
  "productLine" varchar(50) [not null]
  "productScale" varchar(10) [not null]
  "productVendor" varchar(50) [not null]
  "productDescription" text [not null]
  "quantityInStock" smallint(6) [not null]
  "buyPrice" decimal(10,2) [not null]
  "MSRP" decimal(10,2) [not null]

Indexes {
  productLine [name: "productLine"]
}
}

Ref "customers_ibfk_1":"employees"."employeeNumber" < "customers"."salesRepEmployeeNumber"

Ref "employees_ibfk_1":"employees"."employeeNumber" < "employees"."reportsTo"

Ref "employees_ibfk_2":"offices"."officeCode" < "employees"."officeCode"

Ref "orderdetails_ibfk_1":"orders"."orderNumber" < "orderdetails"."orderNumber"
```

Ref "orderdetails_ibfk_2":"products"."productCode" < "orderdetails"."productCode"

Ref "orders_ibfk_1":"customers"."customerNumber" < "orders"."customerNumber"

Ref "payments_ibfk_1":"customers"."customerNumber" < "payments"."customerNumber"

Ref "products_ibfk_1":"productlines"."productLine" < "products"."productLine"

After launching the Docker-based database management system containers, the same database management systems were launched on a virtual machine-based environment. For that, the host computer was with the same configurations as the Docker infrastructure host computer.

Applied computational steps to evaluate the performance are mentioned below pseudocode.

[Pseudocode]

```
(1)  INPUT: EXECUTED_QUERY
(2)  OUTPUT: QUERY_EXECUTION_TIME
(3)  BEGIN
(4)       ESTABLISH the Database Connection
(5)       CHECK the Database Connection
(6)       IF (Connection == Success)
(7)       SELECT the Option

(8)       IF (Option == Selection)
(9)       SELECT the Number of Records to Be Selected
(10)      PASS the Value to Proceed to DBMS
(11)      MEASURE the execution time

(12)      ELSE IF (Option == Deletion)
(13)      SELECT the Number of Records to Be Deleted
(14)      PASS the Value to Proceed to DBMS
(15)      MEASURE the execution time

(16)      ELSE IF (Option == Updating)
(17)      SELECT the Number of Records to Be Updated
(18)      PASS the Value to Be Proceed to DBMS
(19)      MEASURE the execution time

(20)      ELSE IF (Option == Insertion)
(21)      SELECT the Number of Records to Be Inserted
(22)      PASS the Value to Proceed to DBMS
(23)      MEASURE the execution time

(24)                DISCONNECT the Database Connection
(25)      ELSE
(26)                DISPLAY the connection error
(27) FINISH
```

For the experiment, MySQL version 8.0.31, PostgreSQL version 14.5, and MongoDB version 5.0 were used. Between the cloud-hosted Docker infrastructure and local remote computer a strong internet connection [upload speed 93.10 Mbps and download speed 94.41 Mbps] was established to eliminate all kinds of external traffics and omit all kinds of external effects during the experimental study.

IV. Results and discussion

After launching the Docker-based database management system containers, the respective database management systems were evaluated at the next stage. Any software application engages with a database management system for the basic CRUD (Create/Insert, Read/Select, Update, and Delete) operations as a thumb rule. Hence to evaluate the database engine responses, the response time for each SQL operation was measured. For the study, the most popular and open-source two relational database management systems and one no-SQL database management system were used. The response time was measured for the SELECT, UPDATE, INSERT, and DELETE operations.

A. SELECT Operation

The SELECT statement is used to select data from a database. The SELECT operation was executed for the selected three database management systems for the Docker-hosted and virtual machine-based infrastructures. The corresponding response time/query execution time was presented below in Table 1 for all infrastructures.

Table 1: Response time for SELECT operation

Data Records	Response Time (s)					
	MySQL		PostgreSQL		MongoDB	
	Docker	Virtual Machine	Docker	Virtual Machine	Docker	Virtual Machine
5	0.3	0.32	0.37	0.3	0.014	0.015
50	0.3064	0.35	0.4284	0.4147	0.087	0.092
500	0.6684	0.8172	1.6137	1.8835	0.1681	0.1763
5000	1.476	2.3146	1.9478	2.0250	0.4274	0.4431
50,000	3.5891	7.5341	4.8790	3.6941	1.0654	1.4145

.

Figure 1 given below, presents the graphical representation of the SELECT query response time (execution time) for Docker-hosted and virtual machine-based MySQL database management system engines. The y-axis is denoting the response time and the x-axis is denoting the number of data records. The blue-coloured line is presenting the Docker-hosted MySQL database management system engine and the red-coloured line is presenting the virtual machine-based MySQL database management system engines.

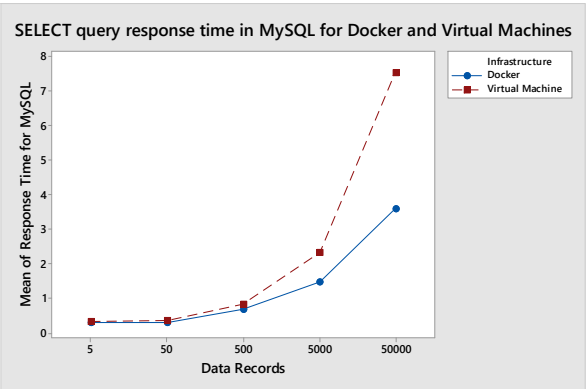

Figure 1: SELECT query response time for MySQL

The above figure 1 presents that the Docker-based MySQL database management system engine has a lower response time for the particular SELECT query than the corresponding virtual machine-based MySQL database management system engine. For the lower data records, both MySQL database management system engine infrastructures present approximately the same response time. But for the higher data records, the Docker-based MySQL database management system engine infrastructure has a lower query response time than the virtual machine-based MySQL database management system engine.

A dependent t-test was steered to assess the performance of MySQL DBMS for 50000 data records for SELECT query execution. The results showed a significant performance improvement in query execution time on Docker (Mean=3.5891, Standard Deviation=0.00008) to query execution time on VM(Mean=7.53407, Standard Deviation=0.00007), t(9)=107590.09, p-value=0.000(two-tailed). This means the increase in the performance of Docker was 3.94489 with a 95% confidence interval level.

Figure 2 below, presents the graphical representation of the SELECT query response time (execution time) for Docker-hosted and virtual machine-based PostgreSQL database management system engines. The y-axis is denoting the response time and the x-axis is denoting the number of data records. The blue-coloured line is presenting the Docker-hosted PostgreSQL database management system engine and the red coloured line is presenting the virtual machine-based PostgreSQL database management system engines.

.

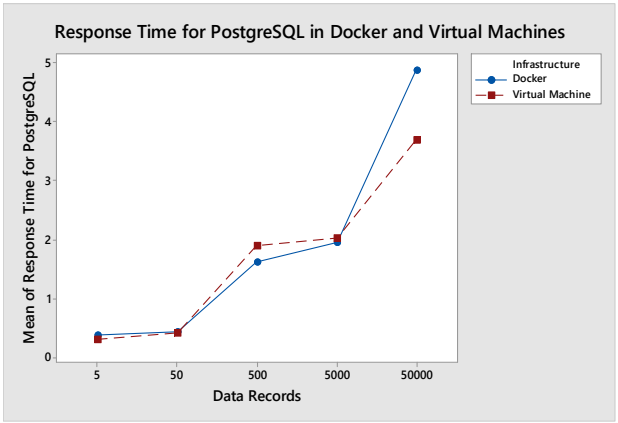

Figure 2: SELECT query response time for PostgreSQL

The above figure 2 presents that the Docker-based PostgreSQL database management system engine has a higher response time for the particular SELECT query than the corresponding virtual machine-based PostgreSQL database management system engine. For the lower data records, both PostgreSQL database management system engine infrastructures present approximately the same response time but for the 500 data records, the Docker-based PostgreSQL database management system engine has a lower response time than the virtual machine-based PostgreSQL database management system engine.

A dependent t-test was steered to assess the performance of PostgreSQL DBMS for 5000 data records for SELECT query execution. The results showed a significant performance improvement in query execution time on Docker (Mean=1.9478, Standard Deviation=0.00005) to query execution time on VM(Mean=2.025, Standard Deviation=0.00008), t(9)=3661.92, p-value=0.000(two-tailed). This means the increase in the performance of Docker was 0.077152 with a 95% confidence interval level for 5000 data records.

Figure 3 below, presents the graphical representation of the SELECT query response time (execution time) for Docker-hosted and virtual machine-based MongoDB database management system engines. The y-axis is denoting the response time and the x-axis is denoting the number of data records. The blue-coloured line is presenting the Docker-hosted MongoDB database management system engine and the red-coloured line is presenting the virtual machine-based MongoDB database management system engines.

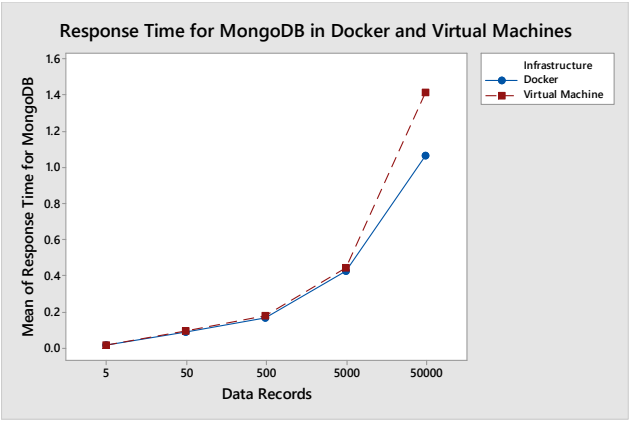

Figure 3: SELECT query response time for MongoDB

The above figure 3 presents that the Docker-based MongoDB database management system engine has a lower response time for the particular SELECT query than the corresponding virtual machine-based MongoDB database management system engine. For the lower data records, both MongoDB database management system engine infrastructures present approximately the same response time. But for the higher data records, the Docker-based MongoDB database management system engine infrastructure has a lower query response time than the virtual machine-based MongoDB database management system engine.

A dependent t-test was steered to assess the performance of MongoDB DBMS for 50000 data records for SELECT query

execution. The results showed a significant performance improvement in query execution time on Docker (Mean=1.06540, Standard Deviation=0.00005) to query execution time on VM(Mean=1.41450, Standard Deviation=0.00009), t(9)=8851.30, p-value=0.000(two-tailed). This means the increase in the performance of Docker was 0.349011 with a 95% confidence interval level.

Overall, Docker-based database management systems are presenting better performance than the virtual machine-based approach on the query execution time.

B. DELETE Operation

The DELETE statement is used to delete data from a database. The DELETE operation was executed for the selected three database management systems for the Docker-hosted and virtual machine-based infrastructures. The corresponding response time/query execution time was presented below in Table 2 for all infrastructures.

Table 2:Response time for DELETE operation

Data Records	Response Time (s)					
	MySQL		PostgreSQL		MongoDB	
	Docker	Virtual Machine	Docker	Virtual Machine	Docker	Virtual Machine
5	0.4	0.43	0.39	0.432	0.063	0.063
50	0.4157	0.494	0.4198	0.4277	0.088	0.097
500	0.6891	0.9641	0.7642	0.9471	0.1170	0.1287
5000	1.561	2.5873	1.7341	1.7753	0.2197	0.2947
50,000	3.9172	7.8973	5.6917	8.4782	0.9784	1.6810

Figure 4 below, presents the graphical representation of the DELETE query response time (execution time) for Docker-hosted and virtual machine-based MySQL database management system engines. The y-axis is denoting the response time and the x-axis is denoting the number of data records. The blue-coloured line is presenting the Docker-hosted MySQL database management system engine and the red-coloured line is presenting the virtual machine-based MySQL database management system engines.

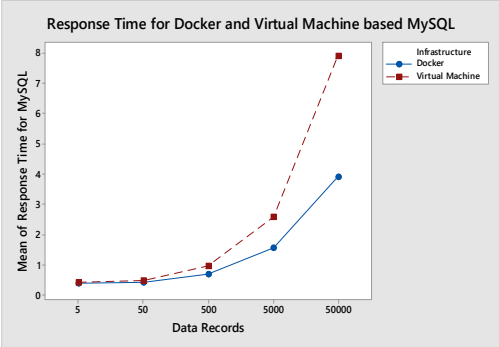

Figure 4: DELETE query response time for MYSQL

Figure 4 above, presents that the Docker-based MySQL database management system engine has a lower response time for the particular DELETE query than the corresponding virtual machine-based MySQL database management system engine. For the lower data records, both MySQL database management system engine infrastructures present approximately the same response time. But for the higher data records, the Docker-based MySQL database management system engine infrastructure has a lower query response time than the virtual machine-based MySQL database management system engine.

A dependent t-test was steered to assess the performance of MySQL DBMS for 50000 data records for DELETE query execution time. The results showed a significant improvement in the query execution time on Docker (Mean=3.9172, Standard Deviation=0.00014) to query execution time on VM(Mean=7.8973, Standard Deviation=0.00005), t(9)=94396.36, p-value=0.000(two-tailed). This means the increase in the performance of the proposed Docker platform was 3.98 with a 95% confidence interval level.

Figure 5 below, presents the graphical representation of the DELETE query response time (execution time) for Docker-hosted and virtual machine-based PostgreSQL database management system engines. The y-axis is denoting the response time and the x-axis is denoting the number of data records. The blue-coloured line is presenting the Docker-hosted PostgreSQL database management system engine and the red coloured line is presenting the virtual machine-based PostgreSQL database management system engines.

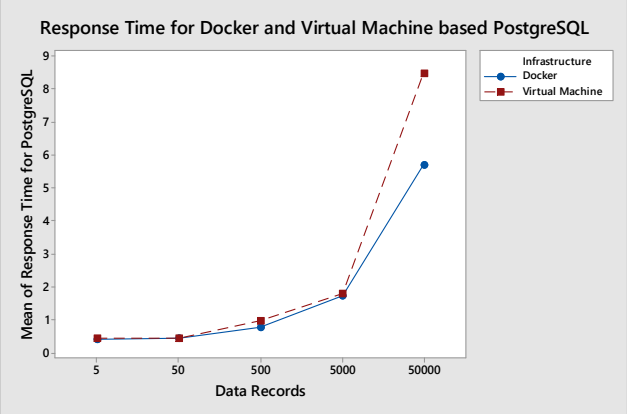

Figure 5: DELETE query response time for PostgreSQL

The above figure 5 presents that the Docker-based PostgreSQL database management system engine has a lower response time for the particular DELETE query than the corresponding virtual machine-based MySQL database management system engine. For the lower data records, both PostgreSQL database management system engine infrastructures present approximately the same response time. But for the higher data records, the Docker-based PostgreSQL database management

system engine infrastructure has a lower query response time than the virtual machine-based PostgreSQL database management system engine.

A dependent t-test was steered to assess the performance of PostgreSQL DBMS for 50000 data records for DELETE query execution time. The results showed a significant improvement in the query execution time on Docker (Mean=5.6917, Standard Deviation=0.00015) to query execution time on VM(Mean=8.4782, Standard Deviation=0.00007), t(9)=59110.59, p-value=0.000(two-tailed). This means the increase in the performance of the proposed Docker infrastructure was 2.78639 with a 95% confidence interval level.

Below figure 6 presents the graphical representation of the DELETE query response time (execution time) for Docker-hosted and virtual machine-based MongoDB database management system engines. The y-axis is denoting the response time and the x-axis is denoting the number of data records. The blue-coloured line is presenting the Docker-hosted MongoDB database management system engine and the red-coloured line is presenting the virtual machine-based MongoDB database management system engines.

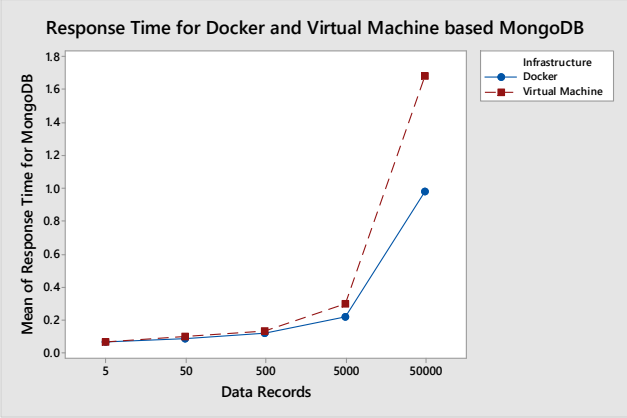

Figure 6: DELETE query response time for MongoDB

The above figure 6 presents that the Docker-based MongoDB database management system engine has a lower response time for the particular DELETE query than the corresponding virtual machine-based MongoDB database management system engine. For the lower data records, both MongoDB database management system engine infrastructures present approximately the same response time. But for the higher data records, the Docker-based MongoDB database management system engine infrastructure has a lower query response time than the virtual machine-based MongoDB database management system engine.

A dependent t-test was steered to assess the performance of MongoDB DBMS for 50000 data records for DELETE query execution time. The results showed a significant improvement in the query execution time on Docker (Mean=0.9784, Standard Deviation=0.00005) to query execution time on VM(Mean=1.68100, Standard Deviation=0.00007), t(9)=21078.00, p-value=0.000(two-tailed). This means the increase in the performance of Docker was 0.702525 with a 95% confidence interval level.

Overall, the Docker-based PostgreSQL database management system engine has a higher response time than the Docker-based MySQL database management system engines for the particular DELETE query.

C. UPDATE Operation

The UPDATE statement is used to update data from a database. The UPDATE operation was executed for the selected three database management systems for the Docker-hosted and virtual machine-based infrastructures. The corresponding response time/query execution time was presented in below table 3 for all infrastructures.

Table 3: Response time for UPDATE operation

Data Records	**Response Time (s)**					
	MySQL		**PostgreSQL**		**MongoDB**	
	Docker	**Virtual Machine**	**Docker**	**Virtual Machine**	**Docker**	**Virtual Machine**
5	0.4961	0.5084	0.5149	0.5331	0.0743	0.087
50	0.5618	0.6347	0.7841	0.8146	0.0971	0.0991
500	0.7156	1.1433	0.7547	0.9947	0.0997	0.1973
5000	1.7891	1.8216	1.8759	1.1724	1.1679	1.2640
50,000	4.0870	8.1735	5.1157	8.8875	1.5441	1.9718

Below figure 7 presents the graphical representation of the UPDATE query response time (execution time) for Docker-hosted and virtual machine-based MySQL database management system engines. The y-axis is denoting the response time and the x-axis is denoting the number of data records. The blue-coloured line is presenting the Docker-hosted MySQL database management system engine and the red-coloured line is presenting the virtual machine-based MySQL database management system engines.

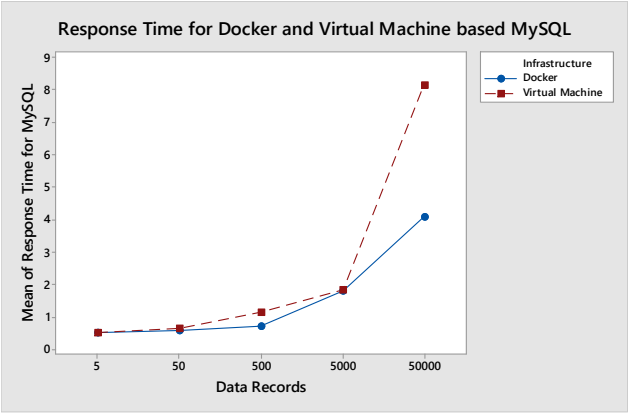

Figure 7: UPDATE query response time for MySQL

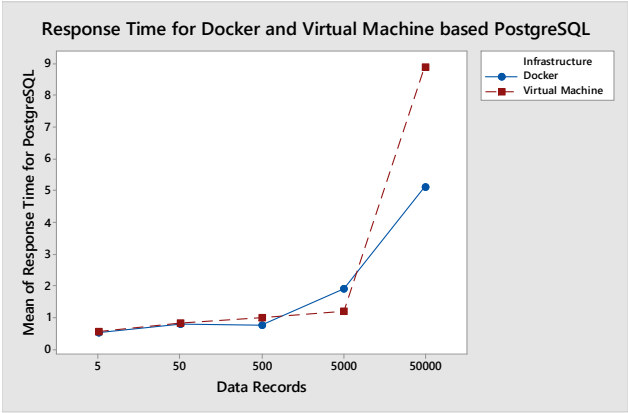

Figure 8: UPDATE query response time for PostgreSQL

The above figure 7 presents that the Docker-based MySQL database management system engine has a lower response time for the particular UPDATE query than the corresponding virtual machine-based MySQL database management system engine. For the lower data records, both MySQL database management system engine infrastructures present approximately the same response time. But for the higher data records, the Docker-based MySQL database management system engine infrastructure has a lower query response time than the virtual machine-based MySQL database management system engine.

A dependent t-test was steered to assess the performance of MySQL DBMS for 50000 data records for UPDATE query execution time. The results showed a significant improvement in the query execution time on Docker (Mean=4.087, Standard Deviation=0.00007) to query execution time on VM(Mean=8.1735, Standard Deviation=0.00013), t(9)=137065.38, p-value=0.000(two-tailed). This means the increase in the performance of the proposed Docker was 4.08643 with a 95% confidence interval level.

Below figure 8 presents the graphical representation of the UPDATE query response time (execution time) for Docker-hosted and virtual machine-based PostgreSQL database management system engines. The y-axis is denoting the response time and the x-axis is denoting the number of data records. The blue-coloured line is presenting the Docker-hosted PostgreSQL database management system engine and the red coloured line is presenting the virtual machine-based PostgreSQL database management system engines.

The above figure 8 presents that the Docker-based PostgreSQL database management system engine has a lower response time for the particular UPDATE query than the corresponding virtual machine-based PostgreSQL database management system engine. For the lower data records, both PostgreSQL database management system engine infrastructures present approximately the same response time. But for the higher data records, the Docker-based PostgreSQL database management system engine infrastructure has a lower query response time than the virtual machine-based PostgreSQL database management system engine. But for the 5000 data records, the Docker-based PostgreSQL database management system engine has presented a higher response time than the virtual machine-based PostgreSQL database management system engine.

A dependent t-test was steered to assess the performance of PostgreSQL DBMS for 50000 data records for UPDATE query execution time. The results showed a significant improvement in the query execution time on Docker (Mean=5.1157, Standard Deviation=0.00020) to query execution time on VM(Mean=8.8875, Standard Deviation=0.00022), t(9)=31877.53, p-value=0.000(two-tailed). This means the increase in the performance of Docker was 3.77153 with a 95% confidence interval level.

Below figure 9 presents the graphical representation of the UPDATE query response time (execution time) for Docker-hosted and virtual machine-based MongoDB database management system engines. The y-axis is denoting the response time and the x-axis is denoting the number of data records. The blue-coloured line is presenting the Docker-hosted MongoDB database management system engine and the red-coloured line is presenting the virtual machine-based MongoDB database management system engines.

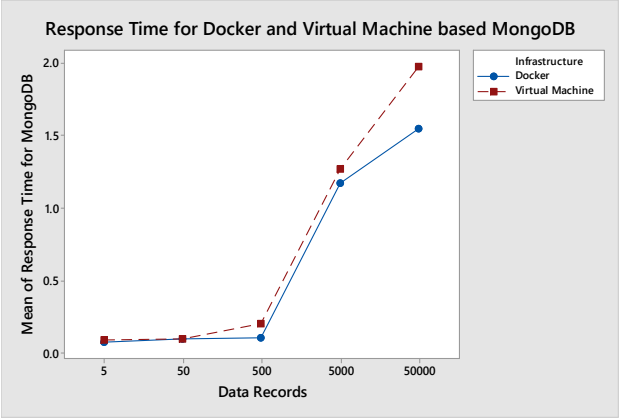

Figure 9: Update query response time for MongoDB

The above figure 9 presents that the Docker-based MongoDB database management system engine has a lower response time for the particular UPDATE query than the corresponding virtual machine-based MongoDB database management system engine. For the lower data records, both PostgreSQL database management system engine infrastructures present approximately the same response time. But for the higher data records, the Docker-based PostgreSQL database management system engine infrastructure has a lower query response time than the virtual machine-based MySQL database management system engine. But, the above figure is presenting a special behaviour for the 5000 data records. That is, the response time has a massive increment for the 5000 data records than other all scenarios.

A dependent t-test was steered to assess the performance of MongoDB DBMS for 50000 data records for UPDATE query execution time. The results showed a significant improvement in the query execution time on Docker (Mean=1.5441, Standard Deviation=0.00011) to query execution time on VM(Mean=1.9718, Standard Deviation=0.00005), t(9)=10844.17, p-value=0.000(two-tailed). This means the increase in the performance of the proposed Docker infrastructure was 0.427611 with a 95% confidence interval level.

The MySQL, PostgreSQL, and MongoDB database management system engines have lower response times for the Docker-based infrastructure than the virtual machine-based infrastructure. The Docker-based PostgreSQL database management system engine has a higher response time than Docker-based MySQL database management system engines for the particular UPDATE query.

D. INSERT Operation

The INSERT statement is used to insert or create data into a database. The INSERT operation was executed for the selected three database management systems for the Docker-hosted and virtual machine-based infrastructures. The corresponding response time/query execution time was presented below in Table 4 for all infrastructures.

Table 4:Response time for INSERT query

Data Records	Response Time (s)					
	MySQL		PostgreSQL		MongoDB	
	Docker	Virtual Machine	Docker	Virtual Machine	Docker	Virtual Machine
5	0.6482	0.6641	0.7244	0.7573	0.1157	0.2748
50	0.7137	0.7237	0.7784	0.8104	0.2649	0.3113
500	0.8232	1.2424	1.1607	1.2115	0.4381	0.5719
5000	1.8716	2.4104	2.4467	2.5491	1.0816	1.1670
50,000	6.7360	10.1133	6.9818	10.6970	2.7366	3.4108

Below figure 10 presents the graphical representation of the INSERT query response time (execution time) for Docker-hosted and virtual machine-based MySQL database management system engines. The y-axis is denoting the response time and the x-axis is denoting the number of data records. The blue-coloured line is presenting the Docker-hosted MySQL database management system engine and the red-coloured line is presenting the virtual machine-based MySQL database management system engines.

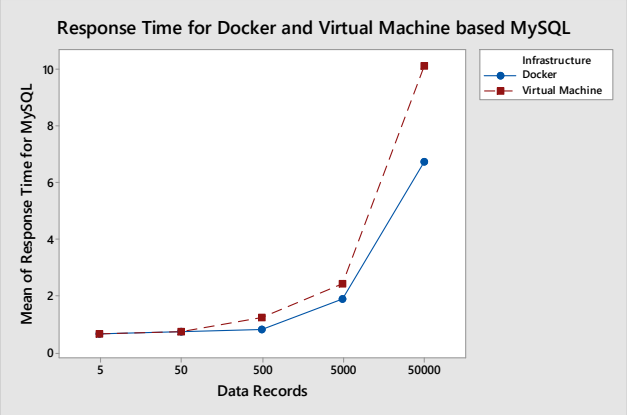

Figure 10: INSERT query response time for MySQL

The above figure 10 presents that the Docker-based MySQL database management system engine has a lower response time for the particular INSERT query than the corresponding virtual machine-based MySQL database management system engine. For the lower data records (less than 50), both MySQL database management system engine infrastructures present approximately the same response time. But for the higher data records (more than 50), the Docker-based MySQL database management system engine infrastructure has a lower query response time than the virtual machine-based MySQL database management system engine.

A dependent t-test was steered to assess the performance of MySQL DBMS for 50000 data records for INSERT query execution time. The results showed a significant improvement in the query execution time on Docker (Mean=6.7360, Standard Deviation=0.0002) to query execution time on VM(Mean=10.1133, Standard Deviation=0.0001), t(9)=75518.72, p-value=0.000(two-tailed). This means the increase in the performance of the proposed Docker was 3.7720 with a 95% confidence interval level.

Below figure 11 presents the graphical representation of the INSERT query response time (execution time) for Docker-hosted and virtual machine-based PostgreSQL database management system engines. The y-axis is denoting the response time and the x-axis is denoting the number of data records. The blue-coloured line is presenting the Docker-hosted PostgreSQL database management system engine and the red coloured line is presenting the virtual machine-based PostgreSQL database management system engines.

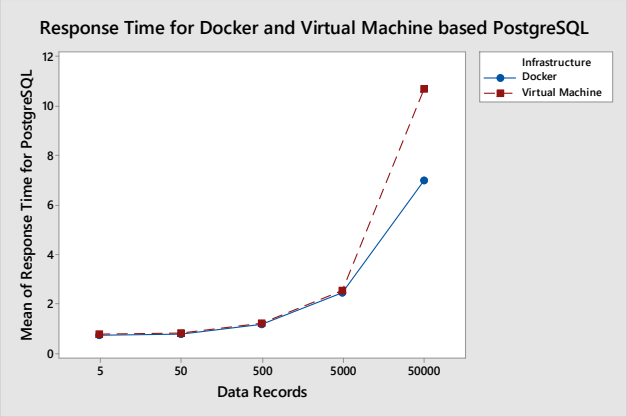

Figure 11:INSERT query response time for PostgreSQL

The above figure 11 presents that the Docker-based PostgreSQL database management system engine has a lower response time for the particular INSERT query than the corresponding virtual machine-based PostgreSQL database management system engine for the higher data records. For the lower data records 5-5000, both PostgreSQL database management system engine infrastructures present approximately the same response time.

A dependent t-test was steered to assess the performance of PostgreSQL DBMS for 50000 data records for INSERT query execution time. The results showed a significant improvement in the query execution time on Docker (Mean=6.6918, Standard Deviation=0.0000) to query execution time on VM(Mean=10.6970, Standard Deviation=0.0002), t(9)=71944.54, p-value=0.000(two-tailed). This means the increase in the performance of the proposed Docker was 3.71508 with a 95% confidence interval level.

Below figure 12 presents the graphical representation of the INSERT query response time (execution time) for Docker-hosted and virtual machine-based MongoDB database management system engines.

The y-axis is denoting the response time and the x-axis is denoting the number of data records. The blue-coloured line is presenting the Docker-hosted MongoDB database management system engine and the red-coloured line is presenting the virtual machine-based MongoDB database management system engines.

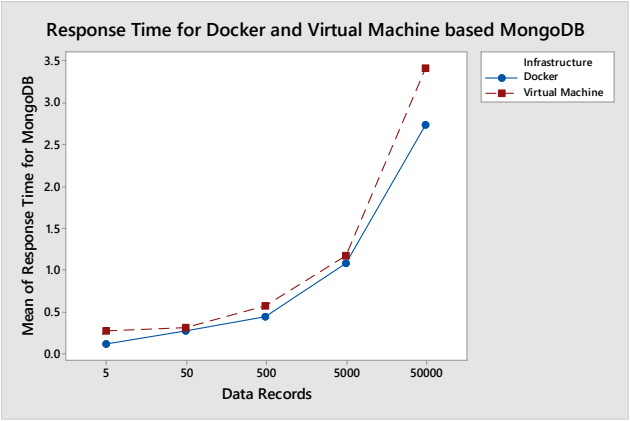

Figure 12: INSERT query response time for MongoDB

The above figure 12 presents that the Docker-based MongoDB database management system engine has a lower response time for the particular INSERT query than the corresponding virtual machine-based MongoDB database management system engine.

A dependent t-test was steered to assess the performance of MongoDB DBMS for 50000 data records for INSERT query execution time. The results showed a significant improvement in the query execution time on Docker (Mean=2.7366, Standard Deviation=0.00034) to query execution time on VM(Mean=3.4108, Standard Deviation=0.00005), t(9)=6333.01, p-value=0.000(two-tailed). This means the increase in the performance of the proposed Docker infrastructure was 0.673959 with a 95% confidence interval level.

The MySQL, PostgreSQL, and MongoDB database management system engines have lower response times for the Docker-based infrastructure than the virtual machine-based infrastructure.

V. Conclusions

After the initial launch of the Docker engine, on the host computer infrastructure: MySQL, PostgreSQL, and MongoDB database management system containers were launched. The

particular queries were executed through the remote database client software.

According to the experimental evaluation: Docker-based relational database management system containers presented the quickest response time for each query than traditional virtual machine-based database management systems. As well for the no-SQL database management systems also, Docker-based containers presented the quickest response time than traditional virtual machine-based database management systems. Furthermore, overall no-SQL database management system containers presented quicker response time than relational database management system containers.

Shortly, Docker containers will play a major role in practical information technology. As well the cloud computing, image processing, artificial intelligence, and data science domains will have oriented to the Docker container-based infrastructure.

REFERENCES

[1] John Paul Martin, A. Kandasamy, and K. Chandrasekaran. 2018. Exploring the support for high performance applications in the container runtime environment. Hum.-centric Comput. Inf. Sci. 8, 1, Article 124 (December 2018), 15 pages. DOI:https://doi.org/10.1186/s13673-017-0124-3

[2] B. I. Ismail et al., "Evaluation of Docker as Edge computing platform," 2015 IEEE Conference on Open Systems (ICOS), 2015, pp. 130-135, doi: 10.1109/ICOS.2015.7377291.

[3] 8 surprising facts about real Docker adoption, 2021. [Online]. Available: https://www.datadoghq.com/docker-adoption/. [Accessed: 25- Jun- 2021].

[4] Empowering App Development for Developers | Docker, Docker, 2021. [Online]. Available: https://www.docker.com/. [Accessed: 25- Jun- 2021].

[5] Database Management System Tutorial - Tutorialspoin, Tutorialspoint.com, 2021. [Online]. Available: https://www.tutorialspoint.com/dbms/index.htm. [Accessed: 25- Jun- 2021].

[6] SQL - RDBMS Concepts - Tutorialspoint, Tutorialspoint.com, 2021. [Online]. Available: https://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm. [Accessed: 25- Jun- 2021].

[7] docker stats, Docker Documentation, 2021. [Online]. Available: https://docs.docker.com/engine/reference/commandline/stats/. [Accessed: 25- Jun- 2021].

[8] F. Paraiso, S. Challita, Y. Al-Dhuraibi and P. Merle, "Model-Driven Management of Docker Containers," 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), 2016, pp. 718-725, doi: 10.1109/CLOUD.2016.0100.

[9] J. Stubbs, W. Moreira and R. Dooley, "Distributed Systems of Microservices Using Docker and Serfnode," 2015 7th International Workshop on Science Gateways, 2015, pp. 34-39, doi: 10.1109/IWSG.2015.16.

[10] Peinl, R., Holzschuher, F. & Pfitzer, F. Docker Cluster Management for the Cloud - Survey Results and Own Solution. J Grid Computing 14, 265–282 (2016). https://doi.org/10.1007/s10723-016-9366-y

[11] D. Liu and L. Zhao, "The research and implementation of cloud computing platform based on docker," 2014 11th International Computer Conference on Wavelet Actiev Media Technology and Information Processing(ICCWAMTIP), 2014, pp. 475-478, doi: 10.1109/ICCWAMTIP.2014.7073453.

[12] M. T. Chung, N. Quang-Hung, M. Nguyen and N. Thoai, "Using Docker in high performance computing applications," 2016 IEEE Sixth International Conference on Communications and Electronics (ICCE), 2016, pp. 52-57, doi: 10.1109/CCE.2016.7562612.

[13] M. G. Xavier, I. C. De Oliveira, F. D. Rossi, R. D. Dos Passos, K. J. Matteussi and C. A. F. D. Rose, "A Performance Isolation Analysis of Disk-Intensive Workloads on Container-Based Clouds," 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2015, pp. 253-260, doi: 10.1109/PDP.2015.67.

[14] S. Thorpe, “Databases in containers - dzone,” dzone.com, 14-Jun-2018. [Online]. Available: https://dzone.com/articles/databases-in-containers. [Accessed: 05-Apr-2023].

[15] “Is it recommended to use database as a container in production environment?,” Stack Overflow, 01-Nov-1964. [Online]. Available: https://stackoverflow.com/questions/48515460/is-it-recommended-to-use-database-as-a-container-in-production-environment. [Accessed: 05-Apr-2023].

[16] D. Damodaran B, S. Salim, and S. M. Vargese, “Performance evaluation of mysql and mongodb databases,” International Journal on Cybernetics & Informatics, vol. 5, no. 2, pp. 387–394, 2016.

[17] Kithulwatta W.M.C.J.T., Jayasena K.P.N., Kumara B.T.G.S., Rathnayaka R.M.K.T. (2021), International Conference on Advances in Computing and Technology (ICACT–2021) Faculty of Computing and Technology (FCT), University of Kelaniya, Sri Lanka 7-12.