RDBMS, NoSQL, Hadoop: A Performance-Based Empirical Analysis

Amal W. Yassien
German University in Cairo
amal.yassien@student.guc.edu.eg

Amr F. Desouky
German University in Cairo
amr.desouky@guc.edu.eg

ABSTRACT

The relational data model has been dominant and widely used since 1970. However, as the need to deal with big data grows, new data models, such as Hadoop and NoSQL, were developed to address the limitation of the traditional relational data model. As a result, determining which data model is suitable for applications has become a challenge. The purpose of this paper is to provide insight into choosing the suitable data model by conducting a benchmark using Yahoo! Cloud Serving Benchmark (YCSB) on three different database systems: (1) MySQL for relational data model, (2) MongoDB for NoSQL data model, and (3) HBase for Hadoop framework. The benchmark was conducted by running four different workloads. Each workload is executed using a different increasing operation and thread count, while observing how their change respectively affects throughput, latency, and runtime.

CCS Concepts

•**Information systems** → **Database performance evaluation;** •**General and reference** → *Performance;* General conference proceedings;

Keywords

Databases; Hadoop; NoSQL; Performance; RDBMS

1. INTRODUCTION

Since relational data model was presented in 1970 by Edgar Codd, it has been the predominant model for database management [9]. Relational database systems support ACID properties (Atomicity, Consistency, Isolation, Durability), which makes it one of the most powerful and sophisticated models. However, despite its robustness, it can fail to fully accommodate an application's demands. For instance, modern web applications require high availability and low latency access to large data sets. In an effort to meet those demands, companies attempted to simplify their relational schema, weaken durability and referential integrity, as well as other optimization techniques to help addressing the model's limitation. However, those techniques did not address the main drawback which is the scalability of the database [9]; this lead to the emergence of NoSQL data models in the late 2000s [5].

NoSQL, Not Only SQL, is a non-relational distributive data model that supports large scale data storage, massive-parallel data processing, and horizontal scaling [8]. Since NoSQL was developed to overcome the disadvantages of relational databases, many companies invested into researching the field, which allowed it to grow to include subtypes to suit various data sets [5]. Common types of NoSQL databases are:
(1) Key-Value store: resembles a hashtable, where a key maps to a set of values. Its simplicity makes it ideal for high-scalable retrieval tasks, such as shopping carts and user profiles. An example: Redis, LinkedIn's Voldemort.
(2) Document store: similar to relational data model where a "collection" corresponds to a table and a "document" corresponds to a record, which makes it ideal for storing documents such as emails and texts. An example: CouchDB, MongoDB.
(3) Column-Family store: is a column-oriented database where the data is stored in column families, which makes it ideal for batch-oriented processing. An example: Google's BigTables, Hadoop's HBase.
(4) Graph databases: adopt graph theory concepts and use nodes, edges ,and properties. This type is ideal for implementing a recommendation system and is widely used in social networking, as it is rather concerned with the link between the data not the data itself.

Since NoSQL supports horizontal scalability, the need for cluster management becomes necessary. As a result, Hadoop framework was developed to provide adequate cluster management [10]. "Apache Hadoop is an open source software framework for storage and large scale processing of data sets on clusters of commodity hardware" [1]. Hadoop consists of four main parts:
(1) Hadoop Distributive File System (HDFS): a file system that stores large data sets [1].
(2) MapReduce: is used to process a large set of data [1].
(3) HBase: a non-relational distributed database model that offers row-level queries and is often used by real-time applications [1].
(4) Zookeeper: a high performance coordination service for distributed applications [1].

A database is simply where data sets are stored. Despite

AMECSE '16, May 28-29, 2016, Cairo, Egypt
© 2016 ACM. ISBN 978-1-4503-4293-3/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2944165.2944174

the simplicity of its role in application development, it is considered the core of any application. Therefore, a wrong choice at the beginning may have drastic effects, as it is difficult to migrate to another database system [5]. Thus, comparing different data models is important as it provides insight into choosing the most well suited data model for applications. One of the most important key factors in comparison is performance [5]. Consequently, many benchmarks were conducted to evaluate and compare NoSQL models' performance to one another or to that of RDBMS models; however, few benchmarks were conducted to evaluate the performance of Hadoop, NoSQL model, and RDBMS. Thus, the purpose of this paper is to observe the effects of changing operation and thread count with respect to runtime, latency, and throughput by running a benchmark using Yahoo! Cloud Serving Benchmark (YCSB) on three different database systems: (1) MySQL: for relational data model, (2) MongoDB: for NoSQL data model, and (3) HBase: for Hadoop framework.

2. BACKGROUND

2.1 RDBMS and NoSQL

Databases tables can be linked to one another are referred to as relational databases [7]. MySQL is a relational database management system (RDBMS) that is developed by Oracle Corporation. It is widely used in web development, as it offers Structured Querying Language (SQL) compatibility, Foreign-key constraints, and platform independence [7]. Among the various RDBMS, MySQL was selected to conduct this benchmark, as it has the second best performance among MS Access, MSSQL (Microsoft SQL Server), Oracle[11] and JDBC driver in which YCSB supports.

NoSQL is a non-relational data model that provides reliability, and support to basic CRUD (Create, Read, Update, Delete) operations [9]. MongoDB is a NoSQL database that has a flexible schema and supports scalability [3]. It is a document store database that uses BSON, binary JSON, to store the data in documents [2]. Due to documents' flexibility, MongoDB is preferred by most developers [6, 3]. Hence, it was selected as the NoSQL candidate for this benchmark.

2.2 Hadoop Framework: HBase

Hadoop is a framework that is used to store and process terabytes of (big) data. The concept originated in 2002 when Nutch started a framework for horizontally scalable web crawler, then in 2006 it became a standalone framework used in building and managing clusters [10].

Hadoop file system divides large data sets into fixed sized chunks and replicates them across all nodes in a cluster to ensure data availability; it also offers scalability, portability, fault tolerance, and high throughput data access [10].

Since data is stored and accessed as blocks in HDFS, row level data retrieval is not offered. Thus, HBase is needed in the Hadoop framework to rectify this drawback. HBase is a column oriented database where the data is stored in column families and tied together by a key. It partitions its records across multiple regions, where each region is administered by a region server (HRegionServer); on the other hand, master server (HMaster) manages the cluster and keeps track of the region servers and their associated regions, by using Zookeeper's route map [2].

Zookeeper is a coordination service that is responsible for maintaining consistency across the cluster [12], by sharing data and server configurations through a hierarchical name space [1].

2.3 Related Work

The "Performance Analysis of Scalable SQL and NoSQL Database: A Quantitative Approach" master's thesis conducts a benchmark on MySQL, HBase, and MongoDB, using also YCSB, to observe how their performance, throughput and latency, is affected by bottleneck nodes. The thesis showed that HBase had the lowest write latency and highest read latency among the three databases [2]. Additionally, the "RDBMS vs NoSQL: Performance and Scaling Comparison" master's thesis evaluates the performance of both MySQL and MongoDB in terms of runtime and scalability. This thesis concluded that MySQL outperformed MongoDB in executing complex queries; however, MongoDB handled insertions better [5].

2.4 Benchmark: YCSB

Yahoo! Cloud Serving Benchmark is a benchmarking tool that is designed primarily to compare cloud databases with one another by conducting a set of workloads that resembles the actual every day workload that the server may sustain. It measures average latency, runtime, and average throughput [4]. Each workload performs basic operations like insert, read, update, scan, and delete. In addition, workloads can be customized per users' needs, specifying the operations' portions and counts [2, 4]. Furthermore, this tool has customizable core properties, including: (1) Operation Count which is the number of operations executed per workload, (2) Thread Count which is the number of client threads used to execute workloads, and (3) Record Count which is the number of records to load in the database.

3. EXPERIMENT SETUP

3.1 Hardware Configurations

This benchmark was conducted on a single machine with no cluster setup. The configurations were as follows:

Model	MacBookPro 15"(mid 2014)
Processor	Intel Core i7 (2.2 GHz)
Operating System	Mac OS X 10.11.3
RAM	16 GB DDR3
Disk	2 TB HDD with USB 3.0 connection
MySQL	version 5.6.22 and JDBC version 5.1.38
MongoDB	version 3.2.1 with single *mongod* instance
Hadoop	version 2.7.2 with pseudo-distributive mode
HBase	version 1.2.1
Zookeeper	version 3.4.6
YCSB	release 0.7.0

In this experiment, all workloads were executed with all database instances running simultaneously.

3.2 Procedure

3.2.1 Generating The Data Set

The YCSB database schema consists of a single table named 'usertable', which contains 11 fields including the key. Each record has a size of approximately 1 KB.

In this benchmark, 10 million records were loaded into MySQL, MongoDB, and HBase, generating approximately a 10 GB data set.

3.2.2 *The Workloads Used*

Four different workloads were executed, all having a zipfian request distribution, on the generated data set which are:

1. Workload A: performs 50% reads and 50% updates.
2. Workload B: performs 95% reads and 5% updates.
3. Workload C: performs 100% reads.
4. Workload F: performs 100% reads, 50% updates, and 50% read-modify-writes, making it the heaviest and longest workload executed throughout this experiment.

3.2.3 *Changing The Operation Count*

As the number of operation increases, the load on the database increases as well. Consequently, this experiment involved running each of the four workloads with increasingly varying operation counts on MongoDB, MySQL, and HBase to observe the servers' performance under relatively small, moderate, and huge load. A sample of 1,000, 10,000, 100,000, 1,000,000, 10,000,000, and 15,000,000 operations were performed to observe the effects of increasing the operation count with respect to runtime, throughput, and latency.

3.2.4 *Changing The Thread Count*

Threads are generally used to enhance performance when the database is heavily loaded. As a result, this experiment also involves running the four workloads while increasing the number of YCSB client threads. The workloads for this experiment had operation count of one million instead of its default value which is one thousand operation.

A sample of 1 to 10 threads were run to examine the effects of increasing the thread count with respect to runtime, throughput, and latency.

4. RESULTS AND ANALYSIS

The results were analyzed by conducting Pearson Correlation Test to determine the correlation type (Coefficient R) between operation/thread count and: (1) Runtime, (2) Throughput, and (3) Latency.

4.1 Workload A: 50% Read - 50% Update

4.1.1 *Runtime:*

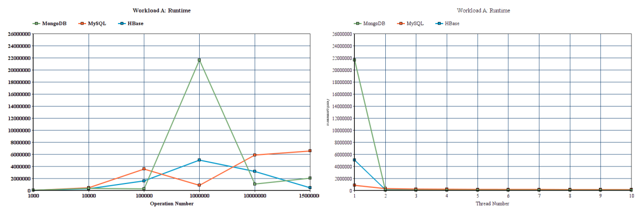

Figure 1: Workload A: Runtime(ms) VS Operation/Thread Count

As shown on the left of Fig.1., as the operation number increases, the runtime becomes longer. As the number of threads increases, the runtime of the workload decreases. As shown on the right of Fig.1, MySQL surprisingly has the shortest runtime, followed by HBase, and MongoDB has the longest runtime.

Table 1: Workload A: Runtime VS Operation count results

DBMS	Correlation	Significance	p-value
MySQL	positive	significant	<0.02
HBase	positive	insignificant	>0.9
MongoDB	negative	insignificant	>0.7

Table 2: Workload A: Runtime VS Thread count results

DBMS	Correlation	Significance	p-value
MySQL	negative	significant	<0.03
HBase	negative	insignificant	>0.1
MongoDB	negative	insignificant	>0.1

4.1.2 *Throughput:*

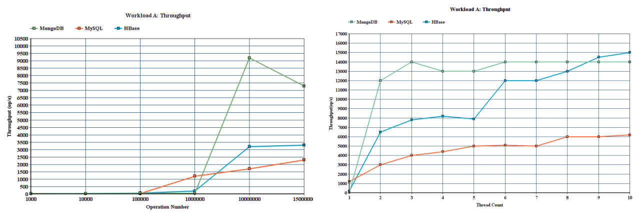

Figure 2: Workload A: Throughput(op/s) VS Operation/Thread Count

Throughput increases as operation number increases, as illustrated on the left of Fig.2. Moreover, MySQL shows a relatively slow increase in throughput compared to HBase and MongoDB. As for MongoDB and HBase, starting from one million operation, throughput showed an immense growth. As shown on the right of Fig.2, as the thread number increases, the throughput also increases. MySQL has the lowest throughput, whereas MongoDB had the highest throughput initially; however, it approached HBase's near the end.

Table 3: Workload A: Throughput VS Operation count results

DBMS	Correlation	Significance	p-value
MySQL	positive	significant	<0.01
HBase	positive	significant	<0.001
MongoDB	positive	significant	<0.009

Table 4: Workload A: Throughput VS Thread count results

DBMS	Correlation	Significance	p-value
MySQL	positive	significant	<0.0002
HBase	positive	significant	<0.00006
MongoDB	positive	insignificant	>0.05

4.1.3 *Read Latency:*

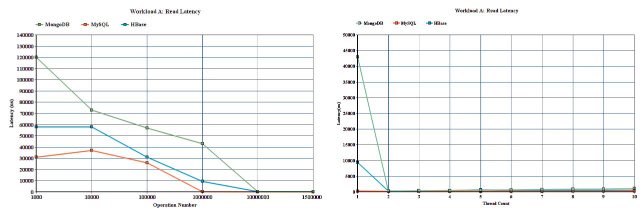

Figure 3: Workload A: Average Read Latency(us) VS Operation/Thread Count

As illustrated on the left of Fig.3., the read latency decreases as the operation number increases. Additionally, MySQL has the lowest read latency, followed by HBase. MongoDB shows an exceedingly high read latency; however, as the operation number increases, it shows a huge decay in read latency, unlike its counterparts that show slightly steady descend. Supposedly, when the thread number increases, the latency also increases; however, as illustrated on the right of Fig. 3, the latency has a negative correlation with thread count. MySQL has the lowest read latency, followed by HBase, leaving MongoDB with the highest read latency.

Table 5: Workload A: Read Latency VS Operation Count results

DBMS	Correlation	Significance	p-value
MySQL	negative	insignificant	>0.1
HBase	negative	insignificant	>0.08
MongoDB	negative	significant	<0.05

Table 6: Workload A: Read Latency VS Thread count results

DBMS	Correlation	Significance	p-value
MySQL	positive	insignificant	>0.06
HBase	negative	insignificant	>0.1
MongoDB	negative	insignificant	>0.1

4.1.4 *Update Latency:*

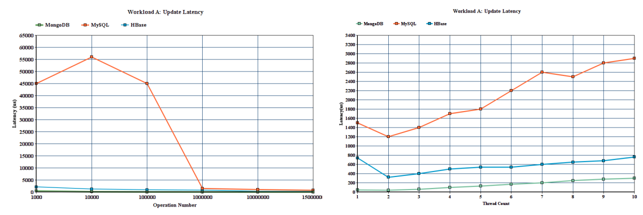

Figure 4: Workload A: Average Update Latency(us) VS Operation/Thread Count

Update latency maintains a steady decay, while MySQL shows a sharp descend starting at 100,000 operation count, as demonstrated on the left of Fig.4. Furthermore, MySQL exhibits the highest update latency, while MongoDB and HBase maintain an exceedingly low update latency. As shown on the right of Fig.4, as the thread number increases, the latency value also increases. MongoDB exhibits the lowest update latency value, followed by HBase, leaving MySQL with the highest latency value.

4.2 Workload B: 95% Read - 5% Update

Table 7: Workload A: Update Latency VS Operation count results

DBMS	Correlation	Significance	p-value
MySQL	negative	insignificant	>0.1
HBase	negative	insignificant	>0.1
MongoDB	negative	insignificant	>0.3

Table 8: Workload A: Update Latency VS Thread count results

DBMS	Correlation	Significance	p-value
MySQL	positive	significant	<0.00001
HBase	positive	insignificant	>0.1
MongoDB	positive	significant	<0.00001

4.2.1 *Runtime:*

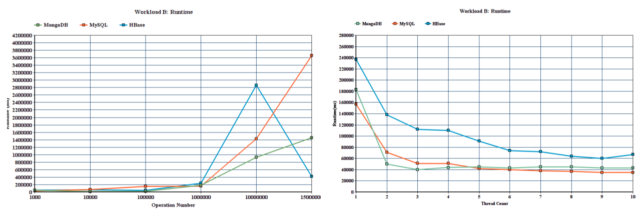

Figure 5: Workload B: Runtime(ms) VS Operation/Thread Count

On the left of Fig.5, MongoDB and MySQL exhibit a huge increase in runtime, unlike HBase, which shows rather slow increase at first, but then shows an enormous increase with an equally enormous decay thereafter. In addition, MongoDB shows the lowest runtime, while MySQL exhibits a nearly exponential increase in runtime. On the other hand, while HBase showed a steady rise initially, it exhibited an alternating increase and decrease behavior thereafter. As illustrated on the right of Fig.5, MongoDB has the lowest runtime value, followed by MySQL, leaving HBase with the highest runtime value.

Table 9: Workload B: Runtime VS Operation count results

DBMS	Correlation	Significance	p-value
MySQL	positive	significant	<0.002
HBase	positive	insignificant	>0.2
MongoDB	positive	significant	<0.00002

Table 10: Workload B: Runtime VS Thread count results

DBMS	Correlation	Significance	p-value
MySQL	negative	significant	<0.02
HBase	negative	significant	<0.003
MongoDB	negative	insignificant	>0.1

4.2.2 *Throughput:*

MongoDB exhibits the highest throughput value and increase rate, while HBase shows the steadiest increase rate, as demonstrated in Fig.6. HBase exhibits the lowest throughput value, while MySQL and MongoDB have relatively close throughput values as demonstrated on the right of Fig.6.

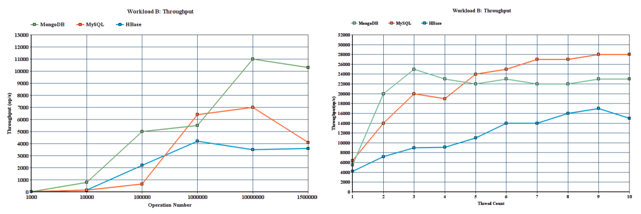

Figure 6: Workload B: Throughput(op/s) VS Operation/Thread Count

4.2.3 *Read Latency:*

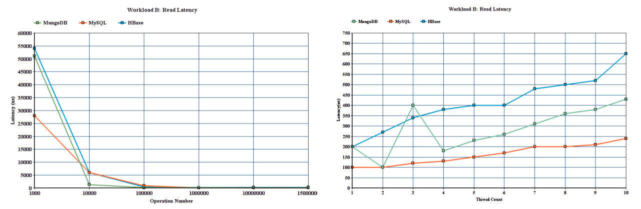

Figure 7: Workload B: Average Read Latency(us) VS Operation/Thread Count

As shown on the left of Fig.7, MySQL exhibits a nearly exponential decay in latency, having the least read latency. Meanwhile, both MongoDB and HBase show a more dramatic descend in their read latencies. As shown on the right of Fig.7, MySQL maintains the slowest increase rate along with lowest read latency, while HBase has the highest read latency.

4.2.4 *Update Latency:*

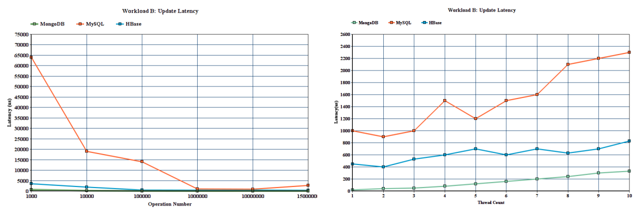

Figure 8: Workload B: Average Update Latency(us) VS Operation/Thread Count

MySQL has the highest update latency and the steepest descend as operation number increases, meanwhile MongoDB and HBase maintain very low steady decaying latency, as illustrated on the left of Fig.8. MongoDB exhibits the lowest update latency, followed by HBase, leaving MySQL with the highest latency value, as illustrated on the right of Fig.8.

4.3 Workload C: Heavy Read (100%)

4.3.1 *Runtime:*

On the left of Fig.9, both MongoDB and MySQL had an exceptional increase in runtime after reaching a million operations. On the other hand, HBase showed a steep ascend but had an equally steep descend after reaching 10 million operations. As illustrated on the right of Fig.9, unexpectedly HBase has the highest runtime values, while MongoDB and MySQL have slightly shorter runtime.

Table 11: Workload B: Throughput VS Operation count results

DBMS	Correlation	Significance	p-value
MySQL	positive	insignificant	>0.2
HBase	positive	insignificant	>0.2
MongoDB	positive	significant	<0.03

Table 12: Workload B: Throughput VS Thread count results

DBMS	Correlation	Significance	p-value
MySQL	positive	significant	<0.0002
HBase	positive	significant	<0.00002
MongoDB	positive	insignificant	>0.1

Table 13: Workload B: Read Latency VS Operation count results

DBMS	Correlation	Significance	p-value
MySQL	negative	insignificant	>0.4
HBase	negative	insignificant	>0.4
MongoDB	negative	insignificant	>0.5

Table 14: Workload B: Read Latency VS Thread count results

DBMS	Correlation	Significance	p-value
MySQL	positive	significant	<0.00001
HBase	positive	significant	<0.00001
MongoDB	positive	significant	<0.00006

Table 15: Workload B: Update Latency VS Operation count results

DBMS	Correlation	Significance	p-value
MySQL	negative	insignificant	>0.3
HBase	negative	insignificant	>0.2
MongoDB	negative	insignificant	>0.4

Table 16: Workload B: Update Latency VS Thread count results

DBMS	Correlation	Significance	p-value
MySQL	positive	significant	<0.00004
HBase	positive	significant	<0.0005
MongoDB	positive	significant	<0.00002

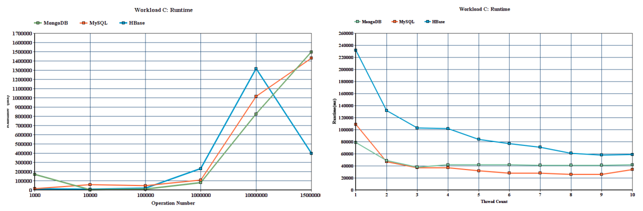

Figure 9: Workload C: Runtime(ms) VS Operation/Thread Count

4.3.2 *Throughput:*

As shown on the left of Fig.10, MongoDB exhibits a nearly exponential throughput growth, while MySQL shows a slightly steady rise, and HBase shows the lowest throughput. MySQL has the highest throughput, followed by MongoDB, while HBase surprisingly maintains the lowest throughput, as shown on the right of Fig. 10.

Table 17: Workload C: Runtime VS Operation count results

DBMS	Correlation	Significance	p-value
MySQL	positive	significant	<0.00002
HBase	positive	insignificant	>0.1
MongoDB	positive	significant	<0.0003

Table 18: Workload C: Runtime VS Thread count results

DBMS	Correlation	Significance	p-value
MySQL	negative	significant	<0.04
HBase	negative	significant	<0.003
MongoDB	negative	insignificant	>0.08

Table 19: Workload C: Throughput VS Operation count results

DBMS	Correlation	Significance	p-value
MySQL	positive	insignificant	>0.4
HBase	positive	insignificant	>0.3
MongoDB	positive	insignificant	>0.4

Table 20: Workload C: Throughput VS Thread count results

DBMS	Correlation	Significance	p-value
MySQL	positive	significant	<0.007
HBase	positive	significant	<0.00002
MongoDB	positive	insignificant	>0.07

4.3.3 *Read Latency:*

MongoDB had an exceedingly high latency starting at 1000 operation, but then decayed dramatically as shown on the left of Fig.11, while MySQL and HBase maintained steadier descend, with HBase surprisingly showing a relatively lower read latency than that of MySQL. HBase exhibits a huge read latency, while MySQL maintains the lowest read latency, followed by MongoDB, as demonstrated on the right of Fig.11.

4.4 Workload F: Heavy Read - Heavy Update

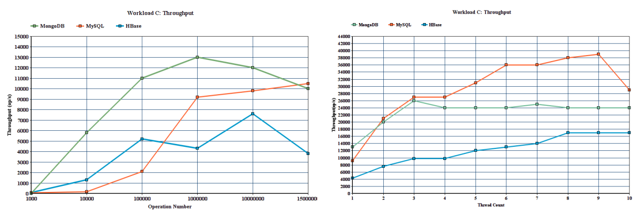

Figure 10: Workload C: Throughput(op/s) VS Operation/Thread Count

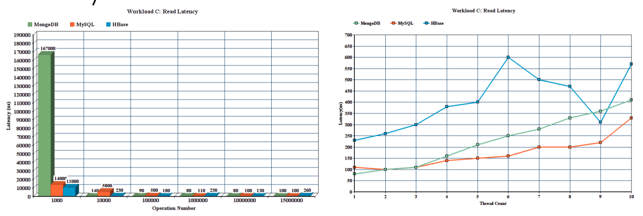

Figure 11: Workload C: Average Read Latency(us) VS Operation/Thread Count

Table 21: Workload C: Read Latency VS Operation count results

DBMS	Correlation	Significance	p-value
MySQL	negative	insignificant	>0.3
HBase	negative	insignificant	>0.5
MongoDB	negative	insignificant	>0.5

Table 22: Workload C: Read Latency VS Thread count results

DBMS	Correlation	Significance	p-value
MySQL	positive	significant	<0.0002
HBase	positive	significant	<0.00002
MongoDB	positive	significant	<0.00002

4.4.1 *Runtime:*

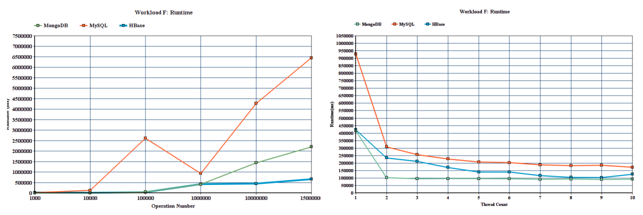

Figure 12: Workload F: Runtime(ms) VS Operation/Thread Count

As shown on the left of Fig.12, MySQL has the highest runtime rate and value, while MongoDB and HBase maintained a steadier ascend. HBase had the least runtime value and increase rate. As shown on the right of Fig. 12, All the database systems' runtime decreases in a near exponential manner, with MongoDB showing the shortest runtime, followed by HBase, leaving MySQL with the highest runtime value.

Table 23: Workload F: Runtime VS Operation count results

DBMS	Correlation	Significance	p-value
MySQL	positive	significant	<0.007
HBase	positive	significant	<0.03
MongoDB	positive	significant	<0.00007

Table 24: Workload F: Runtime VS Thread count results

DBMS	Correlation	Significance	p-value
MySQL	negative	significant	<0.05
HBase	negative	significant	<0.005
MongoDB	negative	insignificant	>0.1

4.4.2 *Throughput:*

MongoDB shows the highest throughput increase rate; initially, HBase and MongoDB showed a close increase rate. Nearing the end, HBase and MySQL showed nearly same throughput values, as demonstrated on the left of Fig.13. As illustrated on the right of Fig.13, MongoDB has the highest throughput, followed by HBase, leaving MySQL with the least throughput value.

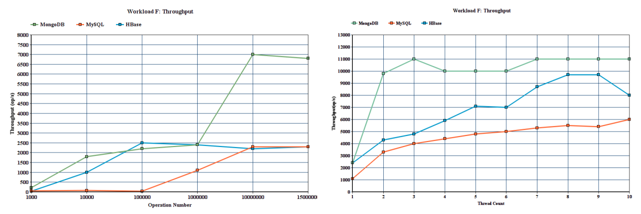

Figure 13: Workload F: Throughput(op/s) VS Operation/Thread Count

4.4.3 *Read Latency:*

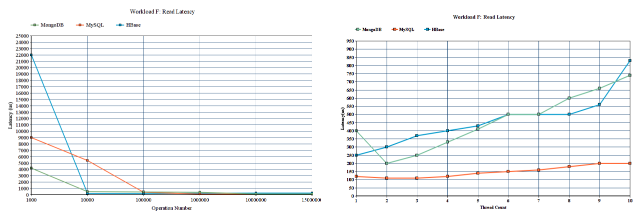

Figure 14: Workload F: Average Read Latency(us) VS Operation/Thread Count

As shown on the left of Fig.14, HBase had the highest the read latency initially; however, it displayed a very steep descend starting at 10,000 operations. MongoDB unexpectedly showed the lowest read latency descend rate, meanwhile MySQL showed a moderate read latency descend rate. MySQL has the lowest read latency, while HBase and MongoDB have a relatively high read latency, as shown on the right of Fig.14.

Table 25: Workload F: Throughput VS Operation count results

DBMS	Correlation	Significance	p-value
MySQL	positive	significant	<0.009
HBase	positive	insignificant	>0.4
MongoDB	positive	significant	<0.006

Table 26: Workload F: Throughput VS Thread count results

DBMS	Correlation	Significance	p-value
MySQL	positive	significant	<0.0007
HBase	positive	significant	<0.0002
MongoDB	positive	insignificant	>0.05

Table 27: Workload F: Read Latency VS Operation count results

DBMS	Correlation	Significance	p-value
MySQL	negative	insignificant	>0.3
HBase	negative	insignificant	>0.5
MongoDB	negative	insignificant	>0.4

Table 28: Workload F: Read Latency VS Thread count results

DBMS	Correlation	Significance	p-value
MySQL	positive	significant	<0.00002
HBase	positive	significant	<0.0002
MongoDB	positive	significant	<0.0004

4.4.4 *Read-Modify-Write Latency:*

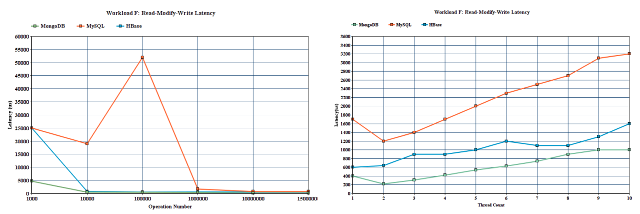

Figure 15: Workload F: Average Read-Modify-Write Latency(us) VS Operation/Thread Count

As shown on the left of Fig.15, MySQL exhibits the highest read-modify-write latency increase rate and value. Both HBase and MongoDB maintain a lower ascend rate, with MongoDB exhibiting the lowest latency rate and value. On the right of Fig.15, MongoDB has the lowest latency, followed by HBase, while MySQL has the highest latency.

Table 29: Workload F: Read-Modify-Write Latency VS Operation count results

DBMS	Correlation	Significance	p-value
MySQL	negative	insignificant	>0.1
HBase	negative	insignificant	>0.5
MongoDB	negative	insignificant	>0.4

Table 30: Workload F: Read-Modify-Write Latency VS Thread count results

DBMS	Correlation	Significance	p-value
MySQL	positive	significant	<0.00003
HBase	positive	significant	<0.00005
MongoDB	positive	significant	<0.00002

4.4.5 *Update Latency:*

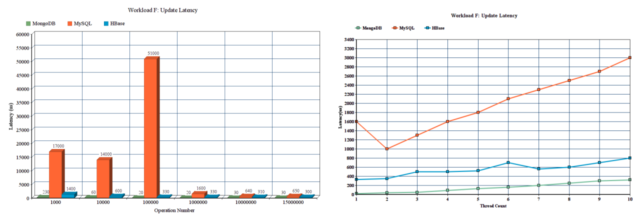

Figure 16: Workload F: Average Update Latency(us) VS Operation/Thread Count

MySQL showed the highest update latency values, meanwhile MongoDB showed the least update latency values. Both HBase and MongoDB maintained a steady slow decay, while MySQL had a nearly exponential decay starting from 100,000 operation, as illustrated on the left of Fig.16. As illustrated on the right of Fig.16, MongoDB shows the lowest update latency, followed by HBase, while MySQL has the highest update latency.

4.5 Analysis

The three database systems maintained a consistent behavior throughout the increase of operation and thread count. Consequently, the decision process is solely based on the applications' nature. However, it can be inferred from the

Table 31: Workload F: Update Latency VS Operation count results

DBMS	Correlation	Significance	p-value
MySQL	negative	insignificant	>0.2
HBase	negative	insignificant	>0.3
MongoDB	negative	insignificant	>0.4

Table 32: Workload F: Update Latency VS Thread count results

DBMS	Correlation	Significance	p-value
MySQL	positive	significant	<0.00004
HBase	positive	significant	<0.0002
MongoDB	positive	significant	<0.00002

figures that MongoDB shows the steadiest increase and decrease in runtime with respect to increasing operation and thread count respectively. Furthermore, MongoDB maintained unexpectedly the lowest update latency in most workloads. Moreover, HBase surprisingly showed the least throughput in 50% of the conducted workloads, while MongoDB maintained the highest throughput in 88% of the conducted workloads. Additionally, HBase exhibited the highest read latency through 50% of the workloads.

MySQL shows the lowest read latency amongst all database systems despite a heavy read load, while also exhibiting the highest update and write latency (100% of workloads) with relatively moderate throughput. Moreover, while loading and generating the data set, MySQL had the longest runtime. Thus, a RDBMS could be an ideal database system for application whose operations are mostly reads.

HBase showed the highest read latency, low update latency, short runtime, and relatively high throughput while overwhelmed by a large operation count. Its runtime during the data load phase was the shortest among MongoDB and MySQL. Hadoop mainly focuses on having high utilization rather than low latency. Hence, Hadoop processes monopolized the CPU time when running the workloads on HBase. Generally, Hadoop aims to maintain a low update latency mainly because it writes in memory (average of 4 GB of RAM). Consequently, Hadoop and HBase could be ideal for applications in which most of the operations are limited to write and/or update.

MongoDB shows a moderate read and low update latency, average throughput, and low runtime in most workloads. In addition, it maintains low update latency, perhaps due to high RAM allocation (average of 8 GB). Unlike Hadoop, it focuses on providing low latency rather than high CPU utilization. MongoDB showed a strong performance under large operation and thread count load. Thus, it would be an ideal database system for applications that have an equal number of read and write operations.

5. CONCLUSION

The exponential growth of data has overwhelmed the relational data model, soliciting the development of the non-relational databases including NoSQL and Hadoop. Since databases are the core of any application, choosing the ideal database system is a highly critical task. Thus, in this paper a benchmark was conducted using the YCSB benchmarking tool to observe the effects of changing both the operation and thread count with respect to runtime, latency, and throughput across three database systems: (1) MySQL, (2) MongoDB, and (3) HBase.

The benchmark concluded that HBase has a relatively lower update latency, shorter runtime, and higher throughput than MySQL, making it well suited for applications that require high update and insert operations. MySQL showed the lowest read latency, making it suitable for applications whose operations are mostly reads. MongoDB exhibited a low update latency (close to HBase's), relatively low read latency (approaching MySQL's), and moderate runtime and throughput, making it ideal for applications that require both adequate read and write performance. However, its high memory utilization might be a limitation.

6. REFERENCES

[1] M. V. Anand Loganathan, Ankur Sinha and S. Natarajan. A systematic approach to big data exploration of the hadoop framework. *International Journal of Information & Computation Technology*, 4(9):869–878, 2014.

[2] H. Balasubramanian. Performance analysis of scalable sql and nosql databases: A quantitative approach. Master's thesis, Wayne State University, 2014.

[3] K. Chodorow. *MongoDB: The Definitive Guide.* O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA, USA, 2nd edition, 2013.

[4] B. F. Cooper, E. T. Adam Silberstein, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. *Yahoo! Research*, June 2010.

[5] C. Hadjigeorgiou. Rdbms vs nosql: Performance and scaling comparison. Master's thesis, The University of Edinburgh, 2013.

[6] D. Hammes, H. Medero, and H. Mitchell. Comparison of nosql and sql databases in the cloud. 2014.

[7] M. Kofler. *The Definitive Guide of MySQL.* Springer-Verlag New York, Inc., 175 Fifth Avenue, NY, USA, 2nd edition, 2004.

[8] A. B. M. Moniruzzaman and S. A. Hossain. Nosql database: New era of databases for big data analytics - classification, characteristics and comparison. *International Journal of Database Theory and Application*, 6, 2013.

[9] C. Nance, T. Losser, R. Iype, and G. Harmon. Nosql vs rdbms - why is there a room for both. *Southern Association for Information Systems*, pages 111–116, May 2013.

[10] V. Sharma and M. Dave. Nosql and hadoop technologies on oracle cloud. *International Journal of Emerging Trends and Technology in Computer Science*, 2, Apr. 2013.

[11] B. Souley and D. Mohammed. Performance analysis of query optimizers under varying hardware components in rdbms. *Journal of Computer Engineering & Information Technology*, 2, 2013.

[12] T. White. *Hadoop: The Definitive Guide.* O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA, USA, 1st edition, 2009.