Advances in Science, Technology and Engineering Systems Journal Vol. 4, No. 2, 395-401 (2019)

www.astesj.com

ASTESJ

ISSN: 2415-6698

Relational Databases Versus HBase: An Experimental Evaluation

Zakaria Bousalem[*,1], Inssaf El Guabassi[2], Ilias Cherti[1]

[1]*Faculty of Sciences and Technologies, Hassan 1st University, Settat, Morocco*

[2]*Faculty of Sciences, Abdelmalek Essaadi University, Morocco*

A R T I C L E I N F O

Article history:
Received:21 February, 2019
Accepted:07 April, 2019
Online: 26 April, 2019

Keywords:
Relational database
HBase
NoSQL
Comparative study
Benchmarking
YCSB

A B S T R A C T

Relational database management systems (RDBMS) have been imposed for more than three decades as a facto standard for data storage, management, and analysis. They have a good reputation by supporting ACID properties (Atomicity, Consistency, Isolation, and Durability) and by adopting the SQL language which has become a standardized language. However, despite their power, RDBMS have failed to meet the modern application's requirements. That's why the need arises for new database management systems that support the manipulation of large amounts of data. NoSQL database systems allow a flexible schema, whereas RDBMSs require a strictly defined schema. They support horizontal scalability and prioritize data availability over consistency (BASE properties) and have performance that remains good with scalability. In this paper, we present an experimental comparison between a relational database (MySQL) and a NoSQL database (HBase) in terms of runtime and latency in different scenarios using the YCSB Framework.

1. Introduction

For more than three decades Relational databases has been the de-facto standard in the database management systems market thanks to its maturity[1], [2]. Nowadays, with a constant growth of data generated by modern web applications such as social networks, e-commerce sites, and mobile applications; the management, querying and analysis data have become a real challenge for relational database management systems (RDBMS). Besides, these data are recorded in several formats (structured, semi-structured and unstructured), whereas the traditional database management systems based on a rigid schema. These limitations of the relational model led the leaders of the internet such as Google, Amazon, eBay, Alibaba and Facebook to develop a new model named NoSQL databases[3], in order to overcome the weakness of relational database management systems towards the variety, the velocity and the large volume of new data captured. "NoSQL" databases are not usually a replacement, but rather a complementary complement to RDBMS and SQL. The NoSQL model is based on the CAP theorem (Consistency Availability Partition Tolerance) as opposed to RDBMS based on ACID properties (Atomicity, Coherence, Isolation, Durability). NoSQL databases management systems (DBMS) can be classified into four categories: Key-Value databases, Document Oriented databases, Column Oriented databases and Graph databases. This classification is due to the fact that each type of database arises in a specific context and based on different architectures [4]. Comparing different models provides a clear vision for choosing the most appropriate model for a given context. The purpose of this article is to compare the relational model (MySQL) and the NoSQL model (HBase)[5] in terms of runtime and latency in different scenarios using the YCSB Framework. We will measure the latency of three cases of operations: 100% read operations, 100% update operations, and a mix of 50% reads and 50% updates with two scenarios. The first is increasing the number of records however the total number of operations remains fixed at 10000. The second is increasing the number of operations while fixing the total number of records at 1 million records in order to reveal how the number of operations and number of records affect the performance in terms of the latency metric and runtime for data loading. In order to make an efficient approach for migration from Relational databases to HBase database, we have started by a feasibility assessment[6], and in this paper, we have made an experimental comparison between relational databases and HBase database. The goal of this comparison is clearly identifying which case is better to migrate from relational to HBase.

The rest of the paper is structured as follows: In section 2 we introduce the basic definitions starting with an introduction to the NoSQL databases after we present the HBase database, so we will see what Databases Benchmarking is, then we provide a brief presentation of YCSB Framework. In section 3 we introduce the experimental strategy used in our paper. In section 4 we describe

[*]Zakaria Bousalem, Faculty of Sciences and Technologies, Hassan 1st University, Settat, Morocco, zakaria.bousalem@gmail.com

the experimental setup for evaluation. In Section 5 we present the MySQL and HBase evaluation results. In Section 6, a summary and general observations about the results of this evaluation are provided. Finally, Section 7 concludes our paper.

2. Basic definitions

2.1. NoSQL Databases

NoSQL (Not Only SQL) is a broad category of next-generation database management systems, as they are typically non-relational, distributed, open source, and support horizontal scaling. Unlike relational databases, they can better respond to big data problems. These database systems do not rely on a rigid relational schema and the database can therefore grow without constraint.

There are various classes of NoSQL DBMS [7]:

- **Key / Value:** These databases function as a key/value associative array. This structure makes it a simple database to set up and allows quick access to information. The value can be a string or an object. It offers high scalability thanks to schema-less approach. E.g. Riak, Azure Table Storage, and Redis.
- **Document-Oriented databases:** These databases management systems are an extension of the key/value databases. Document-oriented engines do not associate a key with a value but with a schema-less document like JSON and XML. The flexibility of these databases makes them polyvalent. E.g. MongoDB, Couchbase Server, and OrientDB.
- **Column-oriented databases:** The data representation is done by columns contrary to traditional DBMS. This structure makes it easier to add a column to a table and manage millions of columns. These databases are known for their ability to scale and to store a large volume of data. These DBMSs are mainly used in environments where it is necessary to access many columns. They are especially useful for streaming data and Real-time analytics. E.g. HBase, Cassandra, and BigTable.
- **Graph databases:** Store data based on graph theory using graph structures (nodes, arcs, and properties). This storage model facilitates the representation of all highly connected data, which is particularly well adapted to the social networks data processing, fraud detection, and recommendation engine[8]. E.g. AllegroGraph, Neo4j, and FlockDB.

2.2. HBase

HBase is a distributed database management system, developed on top of the HDFS file system. It belongs to the column-oriented databases category. HBase is designed to provide real-time access to data stored on HDFS. It supports horizontal scalability which allows it to support extremely large database tables[9]. It was based on "BigTable" DBMS [10].

As shown in Figure 1, the HBase data model is based on six concepts [11]:

- **Table:** HBase was organizing data in tables.
- **Row:** Within tables, the data is organized in rows. RowKey is the identifier for each row.
- **Column Family:** In each row, data is grouped by "Column Families ". All rows have the same "Column Families". The "Column Family" is set when the table is created in HBase.
- **Column Qualifier:** Access to data within a "Column Family" is done via the "column qualifier". It's specified at the data insertion phase.
- **Cell:** Cell is identified by the combination of the "RowKey", the "Column Family" and the "Column Qualifier". It's Stores the values.
- **Version:** The values within a cell are versioned. The versions are identified by their timestamp.

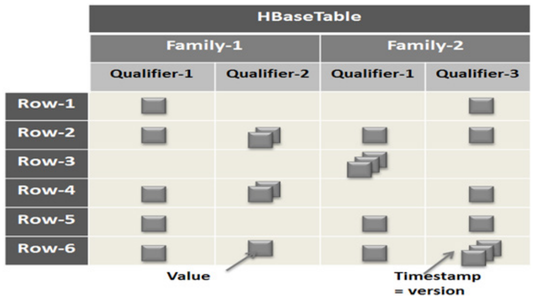


Figure 1: HBase model [12]

2.3. Databases Benchmarking

Database benchmarks (Performance evaluation by experimentation on a real system) [13] are an important tool for database researchers, designers, and users. Its role is to generate application-specific workloads and to test databases in order to assess the relative performance and ease the process of making comparisons between different database specifications. As mentioned by [14] the big data benchmarking process is composed of five steps: Planning, Generating data, Generating tests, Execution and Analysis, and evaluation Figure 2[14].

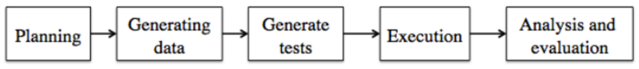


Figure 2: Benchmarking process for big data systems [14]

There are many existing tools for Big Data benchmark [15] like BigBench [16], TPC-C[17] , TPC-E[18], TPC-H[19], TPC-D[20], Bigdatabench [21] and YCSB [22]. In this paper, we are going to use Yahoo! Cloud Serving Benchmark (YCSB) because is currently the most popular choice for benchmarking performance of big data databases [23].

2.4. YCSB

YCSB benchmark is an extensible, modular benchmarking tool, it was developed by Yahoo teams for the aim of measuring the performance of various storage solutions, with adapters for a variety database systems such as Relational databases (with JDBC driver), Big data databases(HBase, Mongo, Cassandra, Redis, HyperTable, Couchbase, DynamoDB, Accumulo, etc) and others.

As shown in Figure 3 YCSB client is composed of four modules: the executor workload, client-threads module, Database interface module and statistics module. After generating the data to be loaded to the database by YCSB client, the executor workload will launch several client threads that execute a series of operations (client-threads module) by using the (Database Interface Layout). The statistics module will retrieve statistics

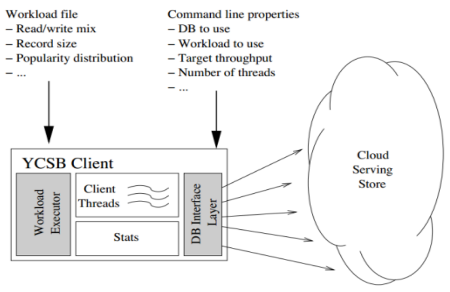


Figure 3: YCSB client architecture [22]

from each operation and analyze them.

YCSB benchmark varies the proportion of read, write, update, insertion, and scan operations in a series of queries named workloads. The YCSB distribution includes six workloads:

- **Workload A:** A mixed workload with 50% of reads and 50% of writes
- **Workload B:** A mixed workload with 95% of reads and 5% of writes.
- **Workload C:** A workload of 100% read
- **Workload D:** A mixed workload with 95% of reads and 5% of inserts.
- **Workload E:** A mixed workload with 95% of scans and 5% of inserts.
- **Workload F: Read-modify-write:** A mixed workload with 50% of reads and 50% of read-modify-writes

3. Experimental strategy

Much work on the potential of comparing database performances by YCSB has been carried out [24]–[26]. Abramova et al [24] compare five NoSQL databases (Redis, Cassandra, HBase, MongoDB, and OrientDB) in terms of their capabilities, based on read and update operations. They affirm that MongoDB, Redis, and OrientDB are better for reads, Cassandra and HBase are optimized for updates. Yassien and Desouky [26] compare MySQL, MongoDB, and HBase by using YCSB for the aim to study the effect of varying the operation and thread count with respect to runtime, throughput, and latency. The authors state that each database performs at its best in different circumstances. They recommend HBase to use for the applications that require the high update and insert operations, MySQL for the applications whose perform mostly reads operations and MongoDB for the applications that require both adequate read and write performance. Matallah et *al* [25] compare MongoDB and HBase in order to evaluate loading and running time of five workloads. According to their results, when performing reads, MongoDB showed good performance, unlike HBase which showed good performance for updates.

Latency means the time of response can get user when sending a request. It's one of the essential metrics to evaluate databases performance [27]. Consequently, we chose in our paper to compare MySQL and HBase in terms of runtime and response time (latency) based on read and update operations since they are the most used operations[24] while increasing the number of records however the total number of operations remains fixed at 10000. Then we will increase the number of operations while fixing the total number of records at 1 million records in order to reveal how the number of operations and number of records affect the performance in terms of latency and runtime for data loading.

4. Experimental setup

In order to perform our comparative study, we present the experimental setup for evaluation. The experiments were run using a single physical machine with Ubuntu operating system, YCSB benchmark, Cloudera Hadoop, Cloudera HBase, and, MySQL. All specifications are listed in Table1.

Table 1 Experimental specifications

CPU	Intel® Xeon(R) CPU E5504 @ 2.00GHz × 8
Memory	16 GB
Hard disk	237 GB SSD
Operating system	Ubuntu 14.04 (64-bit)
Java version	1.8.0_16
YCSB version	0.14
CDH version	5.14.1
Cloudera HBase version	1.2.0
Cloudera Hadoop version	2.6.0
MySQL	5.6.26

The main focus of this study is to evaluate read and update operations since they are the most used operations [24]. Therefore this comparison mainly consists of three workloads namely A and B included in the YCSB project and we create new workload G proposed by [24] to evaluate the Update Only case. Table 2 shows the tested workloads:

Table 2 Used workloads

Workload	Operations
Workload A	50% of reads and 50% of writes
Workload C	100% read: Read Only
Workload G	100% update : Update Only

The dataset used in this databases benchmarking is generated by YCSB data generator which is a part of YCSB client. The dataset records are composed of 10 fields. Each field is filled by a random string with 100 bytes which give 1 KB per record. The 'YCSB_KEY' is the primary key for each row[22]. Table 3 shows the YCSB dataset structure.

Table 3 YCSB Dataset structure

	YCSB_KEY	FIELD1	FIELD2	FIELD3	FIELD4	FIELD5	FIELD6	FIELD7	FIELD8	FIELD9
Row 1										
Row 2										
......										

Row N										

5. Experimental results

We performed three tests. The first is loading data, the second is running workloads while increasing the number of records and fix the number of operations at 10000 and the last is running workloads while increasing the number of operations and set the number of records to 1 million.

5.1. Loading data

- **Runtime (less is better):** As shown in Figure 4 as the size of data increases the runtime of loading data for MySQL and HBase increases, HBase exhibited an immense increase but MySQL shows starting from 100000 records a dramatic increase. Additionally, HBase has the lowest runtime. In the first test (Record number=1000) MySQL and HBase have almost identical runtime. As the record number increases, the runtime for MySQL to load data ranges from 2 times slower than HBase for the second test(Record number=10000), to more than 4 times slower for MySQL for the third test and more than 5 times for the fourth test.

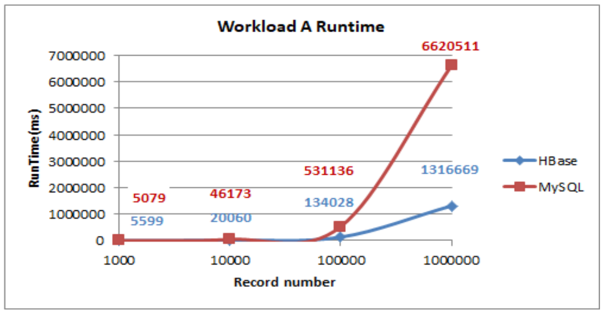


Figure 4: Loading data Runtime

- **Insert latency (less is better):** decreases as the size of data increases for HBase. Unlike MySQL that shows a steadiness initially and then it exhibited a decline and increase thereafter. As shown in Figure 5 HBase has the shortest insert latency.

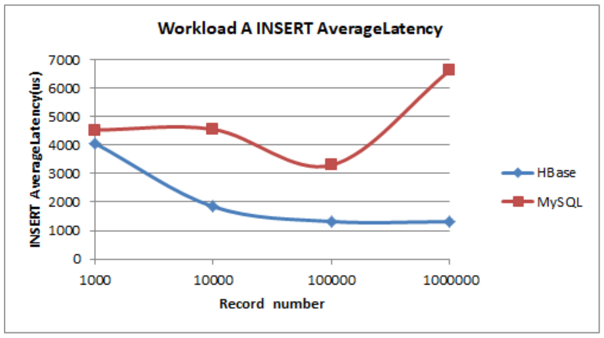


Figure 5: Loading data Insert latency

5.2. Increasing the number of records

5.2.1. Workload A

- **Runtime:** As illustrated in Figure 6 as the size of data increases the runtime of MySQL for data loading increases. MySQL exhibits a slight steady increase in runtime, unlike HBase that shows a slight decline and increase thereafter. HBase has the lowest runtime.
- **Read latency (less is better):** HBase shows a slight decline and increases thereafter, unlike MySQL that exhibits a slight steady increase as shown in Figure 7. MySQL has the shortest read latency.

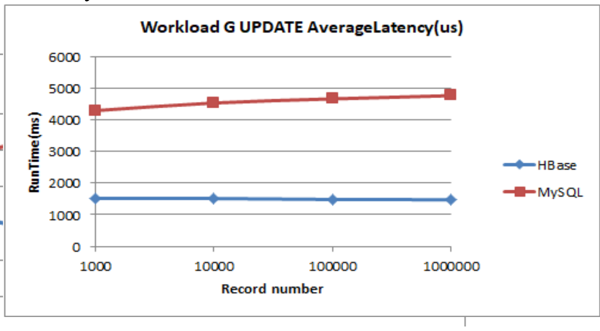


Figure 12: Workload G Latency

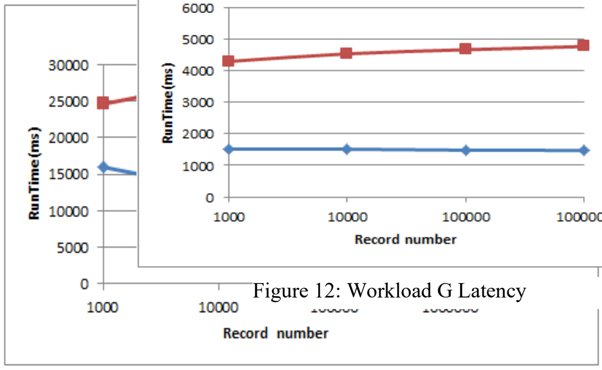


Figure 6: Workload A Runtime

- **Update latency (less is better):** As shown in Figure 8, like runtime in update latency MySQL exhibits a slight steady increase in runtime, unlike HBase that shows a slight decline and increase thereafter. HBase has the lowest update latency.

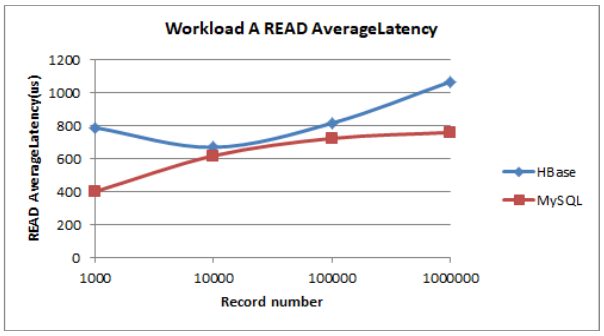


Figure 7: Workload A Read Latency

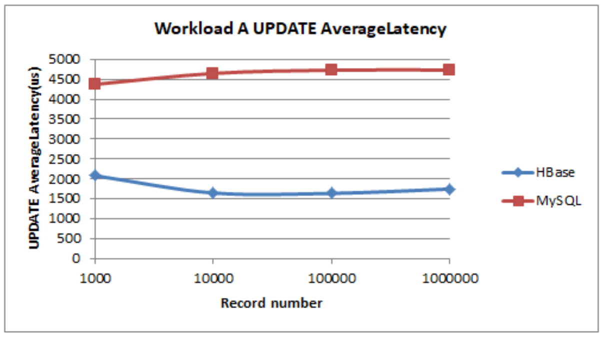


Figure 8: Workload A Update Latency

5.2.2. Workload C

As illustrated in Figure 9 and Figure 10 HBase exhibits a slightly decline initially, it shows an alternating increase and decline thereafter in terms of runtime and read latency, unlike MySQL that shows a steadiness initially, then it exhibited a slight increase after reaching 100000 records. MySQL has the shortest run time and read latency.

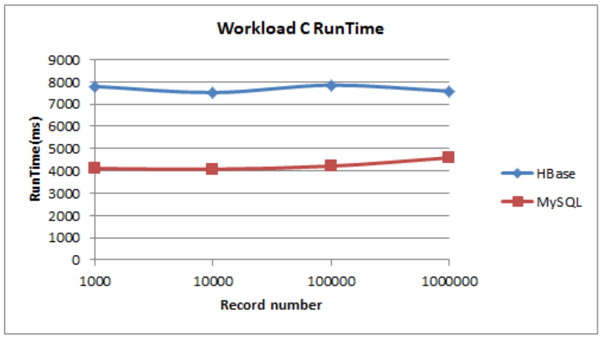


Figure 9: Workload C Runtime

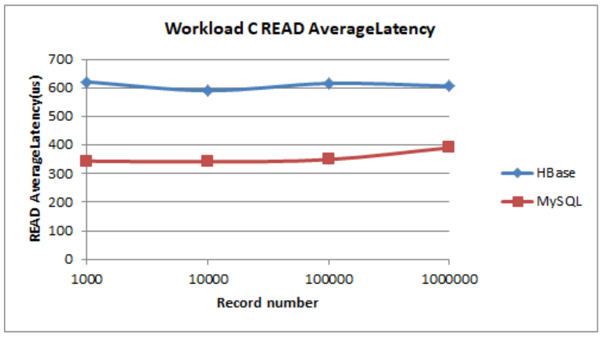


Figure 10: Workload C Read Latency

5.2.3. Workload G

As illustrated in Figure 11 and Figure 12 the HBase exhibits a steadiness both for runtime and update latency, unlike MySQL that shows a slight increase. HBase has the lowest value of runtime and read latency.

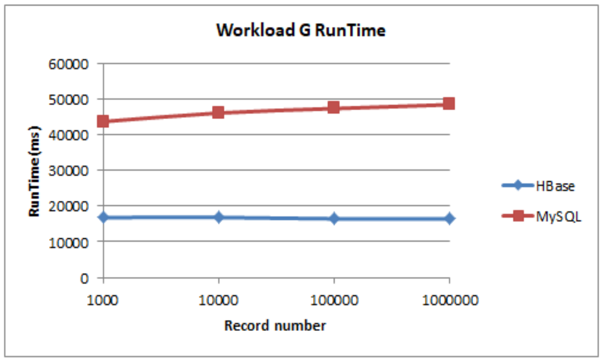


Figure 11: Workload G Runtime

5.3. Increasing the number of operations

5.3.1. Workload A

- **Runtime:** As shown in Figure 13 as the number of operations increases the runtime of MySQL and HBase increases, HBase and MySQL show an immense increase but MySQL exhibits starting from 100000 operations a dramatic increase. HBase has the lowest runtime.

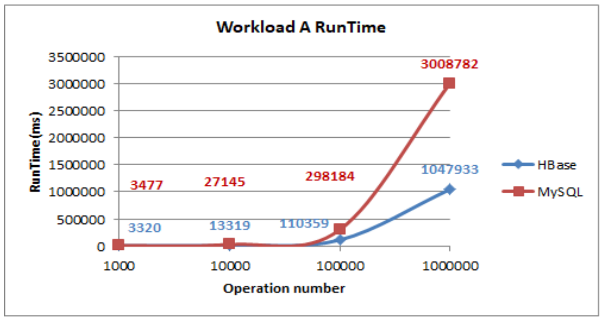
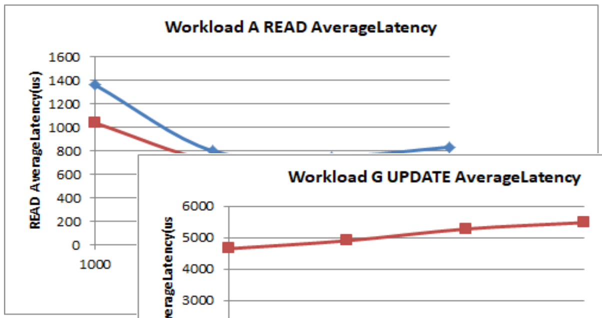


Figure 13: Workload A Runtime

- **Read latency (less is better):** HBase and MySQL show an immense decline until reaching 10000 records, then exhibit slightly decline as shown in Figure 14. MySQL has the shortest read latency.



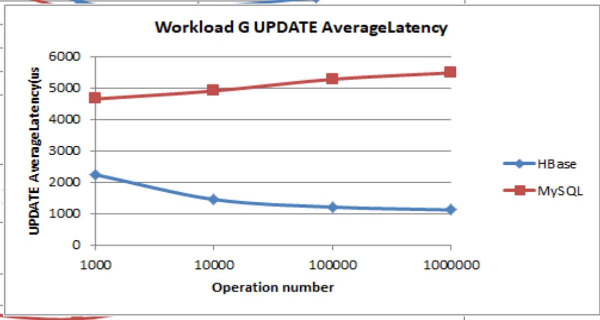


Figure 19: Workload G Update Latency

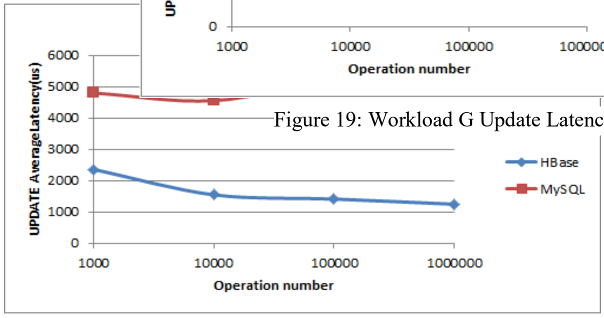


Figure 15: Workload A Update Latency

- **Update latency (less is better):** As illustrated in Figure 15 as the number of operations increases the update latency of HBase decline. MySQL exhibits a slightly decline initially, it shows an alternating increase and a steadiness thereafter in terms of update latency. HBase has the lowest update latency.

5.3.2. Workload C

- **Runtime:** As shown in Figure 16 as the number of operations increases the runtime of MySQL and HBase increases, HBase and MySQL show an immense growth but starting from 100000 operations they exhibit a dramatic increase. MySQL has the lowest runtime.

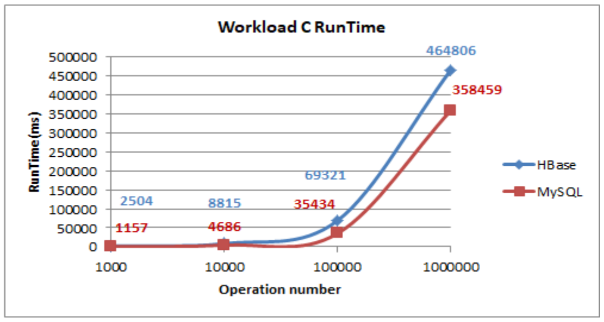


Figure 16: Workload C Runtime

- **Read latency(less is better):** HBase shows an immense decrease, unlike MySQL that exhibits a steady decline as shown in Figure 17. MySQL has the shortest read latency.

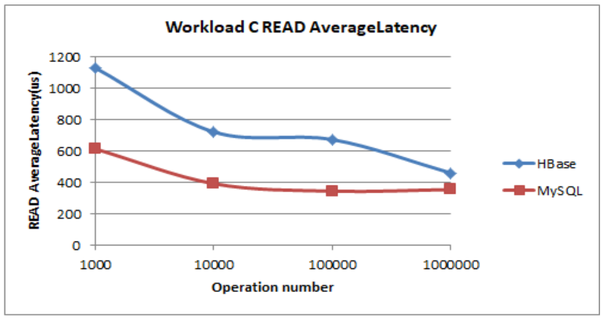


Figure 17: Workload C Read Latency

5.3.3. *Workload G*

- **Runtime:** As shown in Figure 18 as the number of operations increases the runtime of MySQL and HBase increases, HBase and MySQL show an immense increase but MySQL exhibits starting from 100000 operations a dramatic increase. HBase has the lowest runtime.

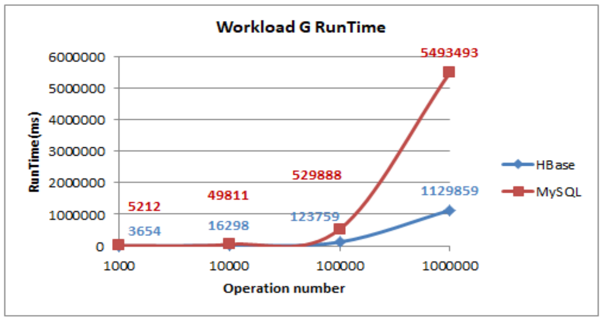


Figure 18: Workload G Runtime

- **Update latency (less is better):** HBase shows an immense decrease, unlike MySQL that exhibits a steady increase as shown in Figure 19. HBase has the shortest update latency.

6. General observations

According to our experimental results, it can be observed that MySQL runtime is higher in all scenarios for data loading and HBase performed far better compared to MySQL. Concerning read/write latencies, it can be stated that MySQL's latency is lower for read operations and HBase's latency is lower for write operations. In terms of the running workloads runtime; HBase beats the competition in all cases except for the read-only workload. Also from our experimental results it can be stated that on the one hand the increasing the number of records seems to have mediocre effects on read latency for MySQL and HBase, no consequences in term of read latency and great impact with respect to runtime data loading (Taken into account that number of records tested was not of a real large size). On the other hand, increasing the number of operations seems to have a significant impact on read and write latency for MySQL and HBase, and immense effects on the running workloads runtime. Consequently, we believe that HBase outperforms MySQL on I/O bound ('write') operations but lagged behind in bound ('read') operations with respect to runtime and latency metrics. HBase exhibits good performance in update operations thanks to using the log files and cache memories to store all transactions and then write only the log files on disk which reduce the input/output operations [25], contrary to MySQL that stores data directly on disk. Additionally, HBase lagged behind in reads capabilities due to comparing all copies by HBase before running a read operation in order to return the most recent copy, which affects database performance [25]. So, according to our experimental results, we can say that is better to migrate from Relational databases (MySQL) to HBase in case of the applications that require a heavy update, most update and high insert operations like session store in order to record recent actions.

7. Conclusion and future work

In this paper, we present an experimental comparison between a relational database (MySQL) and a NoSQL database (HBase) with respect to runtime and latency in different scenarios using the YCSB Framework. Based on the above results we can deduce that HBase performed far better compared to MySQL in data loading because MySQL runtime is higher in all scenarios for this kind of operation. Additionally, we have found that HBase outperforms MySQL on I/O bound ('write') operations but lagged behind in bound ('read') operations with respect to runtime and latency metrics. In perspective, we envisage to compare MySQL and HBase in terms of database performance of the aggregate functions and also pass to higher scales by using a very large database and performing the evaluation in a really distributed and parallel environment.

References

[1] Z. Bousalem and I. Cherti, "XMap: A Novel Approach to Store and Retrieve XML Document in Relational Databases.," *JSW*, vol. 10, no. 12, pp. 1389–1401, 2015.

[2] Z. Bousalem, I. El Guabassi, and I. Cherti, "Toward Adaptive and Reusable Learning Content Using XML Dynamic Labeling Schemes and Relational Databases," in *Advanced Intelligent Systems for Sustainable Development (AI2SD'2018)*, 2019, pp. 787–799.

[3] D. J. Abadi, "Data management in the cloud: Limitations and opportunities.," *IEEE Data Eng Bull*, vol. 32, no. 1, pp. 3–12, 2009.

[4] J. R. Lourenço, B. Cabral, P. Carreiro, M. Vieira, and J. Bernardino, "Choosing the right NoSQL database for the job: a quality attribute evaluation," *J. Big Data*, vol. 2, no. 1, p. 18, 2015.

[5] L. George, *HBase: the definitive guide: random access to your planet-size data.* O'Reilly Media, Inc., 2011.

[6] Z. Bousalem, I. Cherti, and G. Zhao, "Migration from Relational Databases to HBase: A Feasibility Assessment," in *International Conference on Advanced Information Technology, Services and Systems*, 2017, pp. 383–395.

[7] A. B. M. Moniruzzaman and S. A. Hossain, "NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison," *ArXiv13070191 Cs*, Jun. 2013.
[8] J. Webber and I. Robinson, "The Top 5 Use Cases of Graph Databases," *Neo Technol.*, 2015.
[9] N. Dimiduk, A. Khurana, M. H. Ryan, and M. Stack, *HBase in action*. Manning Shelter Island, 2013.
[10] F. Chang *et al.*, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst. TOCS*, vol. 26, no. 2, p. 4, 2008.
[11] A. Khurana, "Introduction to HBase schema design," *White Pap. Cloudera*, 2012.
[12] Blandine Larbret, "Hadoop Hbase - Introduction," 09:09:39 UTC.
[13] J. Darmont, "Data-Centric Benchmarking," in *Encyclopedia of Information Science and Technology, Fourth Edition*, IGI Global, 2018, pp. 1772–1782.
[14] R. Han, X. Lu, and J. Xu, "On big data benchmarking," in *Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware*, 2014, pp. 3–18.
[15] R. Han, L. K. John, and J. Zhan, "Benchmarking big data systems: A review," *IEEE Trans. Serv. Comput.*, vol. 11, no. 3, 2018.
[16] A. Ghazal *et al.*, "BigBench: towards an industry standard benchmark for big data analytics," in *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*, 2013, pp. 1197–1208.
[17] "Tpc-c." [Online]. Available: http://www.tpc.org/tpcc/.
[18] "Tpc-e." [Online]. Available: http://www.tpc.org/tpce/.
[19] "Tpc-h." [Online]. Available: http://www.tpc.org/tpch/.
[20] "Tpc-ds." [Online]. Available: http://www.tpc.org/tpcds/.
[21] L. Wang *et al.*, "Bigdatabench: A big data benchmark suite from internet services," in *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, 2014, pp. 488–499.
[22] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 143–154.
[23] D. Bermbach, E. Wittern, and S. Tai, "Getting Started in Cloud Service Benchmarking," in *Cloud Service Benchmarking*, Springer, 2017, pp. 151–153.
[24] V. Abramova, J. Bernardino, and P. Furtado, "Which nosql database? a performance overview," *Open J. Databases OJDB*, vol. 1, no. 2, pp. 17–24, 2014.
[25] H. Matallah, G. Belalem, and K. Bouamrane, "Experimental comparative study of NoSQL databases: HBASE versus MongoDB by YCSB," *Comput Syst Sci Eng*, vol. 32, no. 4, pp. 307–317, 2017.
[26] A. W. Yassien and A. F. Desouky, "RDBMS, NoSQL, Hadoop: A Performance-Based Empirical Analysis," in *Proceedings of the 2nd Africa and Middle East Conference on Software Engineering*, 2016, pp. 52–59.
[27] X. Tian, R. Han, L. Wang, G. Lu, and J. Zhan, "Latency critical big data computing in finance," *J. Finance Data Sci.*, vol. 1, no. 1, pp. 33–41, 2015.