National College of Ireland

Data Storage and Management

on

Comparing cassandra and hbase

Martin Mohan
18191339

MSc/PGDip Data Analytics – 2019

Submitted to: Noel Cosgrave

National College of Ireland
Project Submission Sheet – 2018/2019
School of Computing

Student Name:	Martin Mohan
Student ID:	18191339
Programme:	MSc Data Analytics
Year:	2018/9
Module:	Data Storage and Management
Lecturer:	Noel Cosgrave
Submission Due Date:	11/08/2019
Project Title:	Comparing cassandra and hbase

I hereby certify that the information contained in this (my submission) is information pertaining to my own individual work that I conducted for this project. All information other than my own contribution is fully and appropriately referenced and listed in the relevant bibliography section. I assert that I have not referred to any work(s) other than those listed. I also include my TurnItIn report with this submission.

ALL materials used must be referenced in the bibliography section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is an act of plagiarism and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature:	
Date:	August 13, 2019

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
3. Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Comparing cassandra and hbase

Martin Mohan
18191339

August 13, 2019

Abstract

This report describes the test strategy and results of a comparative analysis of the performance capabilities of two NoSQL databases cassandra and hbase. The Yahoo Cloud Serving Benchmark tool (YCSB) was used for the performance evaluation.
A latency and throughput were analysed for different YCSB workloads. In general it was found that hbase had a lower read latency than cassandra. cassandra had a lower write latency. This agreed with previous papers on the subject. It was also noted that at higher loads the performance of hbase degraded but cassandra did not. The key characteristics and main architectures of both databases were also described in this paper.

1 Introduction

Large corporations like google, Facebook and Amazon handle ever larger volumes of data. Handling the large amounts of data led large organisations to instigate new and better ways of tackling the problem of big data.
hbase and cassandra are both examples of NoSQL databases running on clusters which were influenced by Google's BigTable Chang et al. (2008).
hbase was a database created by google to run on top of a distributed file system hadoop. Google released their first paper on hbase in 2006. The initial hbase prototype was created as a Hadoop contribution in 2007 and hbase became Apache top-level project in May 2010 tutorialspoint.com (2019).
cassandra was created at Facebook and introduced in the paper Lakshman & Malik (2010). While cassandra implemented the BigTable data model, it uses a system inspired by Amazon's Dynamo for storing data (in fact much of the initial development work on cassandra was performed by two Dynamo engineers recruited to Facebook from Amazon).
This paper describes performance tests carried out to compare cassandra and hbase.The Yahoo Cloud Serving Benchmark tool (YCSB) was used for the performance evaluation. Several YCSB workloads were run and the results used to compare latency and throughput for the two databases.
The paper will also describe the key characteristics and architectures of the two databases. In addition two areas for comparison are also investigated namely ...

- Scalability, Availability and Reliability
- Security

2 Key Characteristics Sadalage & Fowler (2012)

cassandra and hbase both have many characteristics in common which is unsurprising given their common bigtable heritage. Databases with a bigtable-style data model are often referred to as column stores. Most databases have a row as unit of storage, which in particular, helps write performance. However there are many scenarios where writes are rare, but you often need to read a few columns of many rows at once. In this situation it is better to store groups of columns for all rows as the basic storage unit - which is why these databases are called column stores. Bigtable and its offspring (hbase, cassandra) follow this notion of storing groups of columns together and abandon the relational model. They are classed as column-family databases. See figures 1 and 2
The databases are also referred to as aggregate databases (as opposed to relational). The clinching reason for aggregate orientation is that it helps greatly with running on a cluster.If we're running on a cluster, we need to optimize how many nodes we need to query when we are gathering data. By explicitly including aggregates, we the give the database the important information about which bits of data will be manipulated together, and thus should live on the same node. Another key change from a centralized relational database to a NoSQL database is that relational databases try to exhibit strong consistency. In NOSQL databases the consistency is relaxed and the CAP theorem is applied. The basic statement of the CAP theorem is that given the three properties of Consistency, Availability, and Partition tolerance,you can only get two However more recently the validity of the CAP is under question Swaminathan, Surya Narayanan & Elmasri, Ramez (2016) section 2.4.3 Issues in CAP. Clustered databases also use batch processing. The map-reduce pattern is a way to organize processing in such a way as to take advantage of multiple machines on a cluster while keeping as much processing and the data it needs together on the same machine Sadalage & Fowler (2012) chapter 7.

2.1 Key characteristics of hbase

- Being schema-less, there is no concept of column with fixed schema.
- It has a faster lookup system even for large tables.
- As hbase uses Hash Tables, it stores data in indexed format, which results in faster reads.
- Hadoop integration for both source and destination.
- Automatic failure support is provided.
- It is scalable in a linear fashion.
- Data replication across clusters is provided.
- Easy API of Java is available for client.
- Low latency to access single rows from millions of records.
- Can be used to store an manage structured as well as semi-structured data.
- It has de-normalized data.

- Large volumes of data is processed in a parallel style using MapReduce technique.
- Due to built in replication, there is high availability of data.

2.2 Key characteristics of cassandra

- High Scalability: cassandra is horizontally scalable
- Rigid Architecture: cassandra does not have a single point of failure and is continuously available.
- Fast Linear-scale Performance: cassandra is linearly scalable: It increases your throughput because it facilitates you to increase the number of nodes in the cluster. Therefore it maintains a quick response time.
- Fault tolerant: Suppose there are 4 nodes in a cluster, where each node has a copy of same data. If one node is no longer serving then other three nodes can serve as per request.
- Flexible Data Storage: cassandra supports all possible data formats like structured, semi-structured, and unstructured.
- Easy Data Distribution: Data distribution in cassandra is very easy because it provides the flexibility to distribute data where you need by replicating data across multiple data centres
- Transaction Support: cassandra supports properties like Atomicity, Consistency, Isolation, and Durability (ACID).
- Fast writes: cassandra was designed to run on cheap commodity hardware. It performs fast writes and can store hundreds of terabytes of data, without sacrificing the read efficiency

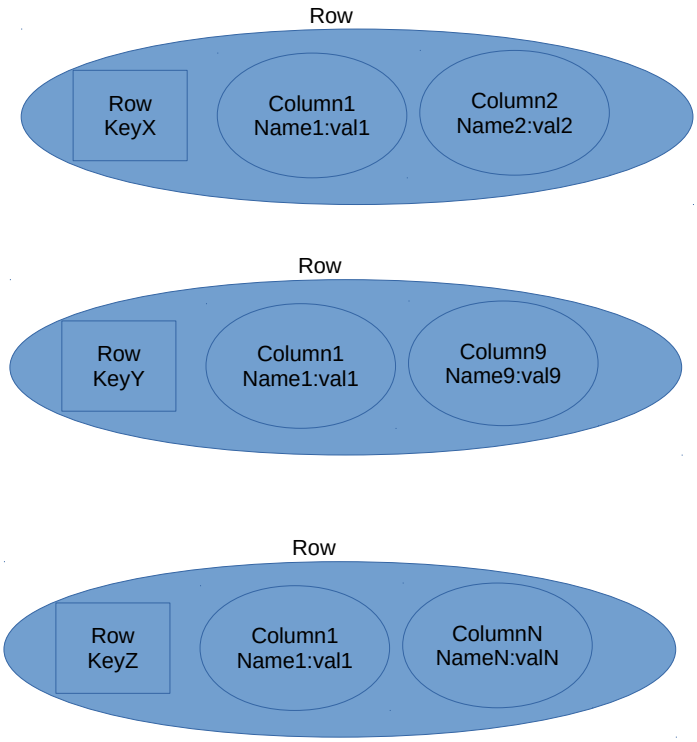
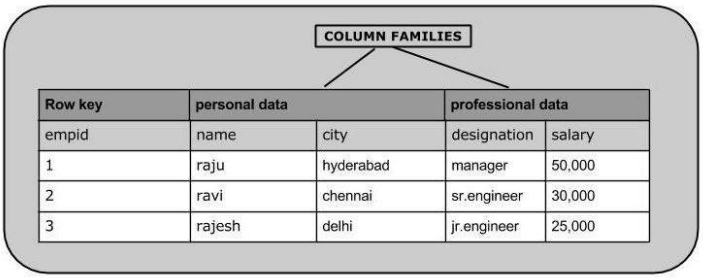


Figure 1: Column Storage



Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

Figure 2: Column Storage Example

3 Data Architecture

3.1 cassandra

cassandra was introduced in the paper Lakshman & Malik (2010) and is the most popular data store that can be categorized as a hybrid NoSQL system Swaminathan, Surya Narayanan & Elmasri, Ramez (2016). The system consists of nodes in a ring as illustrated in 3

cassandra was designed to address many architecture requirements. The most important requirement is to ensure there is no single point of failure. This means that if there are 100 nodes in a cluster and a node fails, the cluster should continue to operate. The main features of cassandra architecture are as follows:

- cassandra is designed such that it has no master or slave nodes.
- It has a ring-type architecture, that is, its nodes are logically distributed like a ring 3
- Data is automatically distributed across all the nodes.
- Similar to HDFS, data is replicated across the nodes for redundancy.
- Data is kept in memory and lazily written to the disk.

cassandra uses cassandra Query Language (CQL), an SQL-like language, for querying.

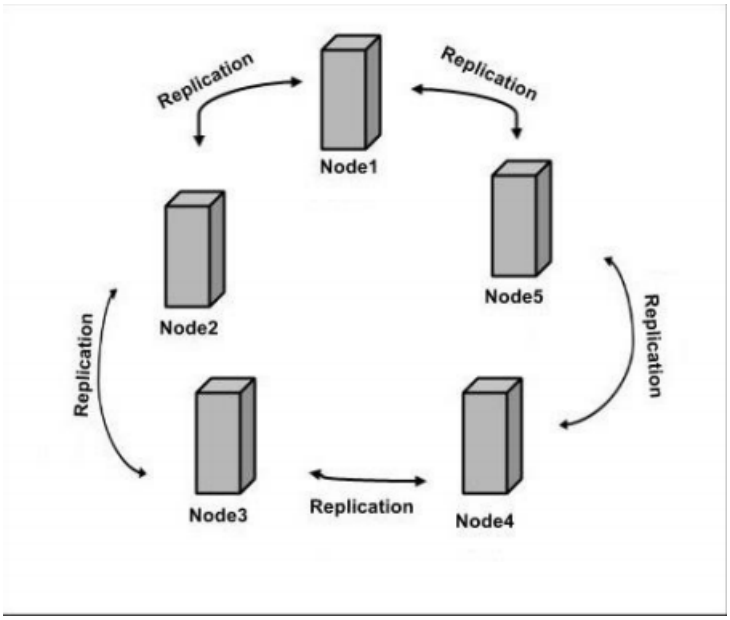


Figure 3: cassandra Architecture kumar Dey (2019)

3.2 hbase

Apache hbase is also patterned after Google's BigTable. hbase runs on top of Hadoop Distributed File System (HDFS). The urge to have a system that supports natural language processing on very large amounts of data is the motivation behind the development

of this Java based tabular store. A diagram of the architecture taken from dezyre (2016) illustrated in 4. hbase is being used in many applications for its fault tolerant distributed

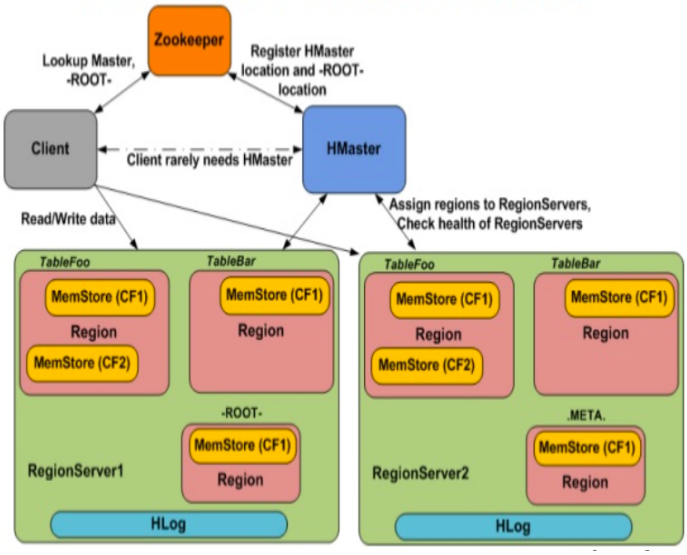


Figure 4: hbase Architecture

data storage (using HDFS), flexible data model and MapReduce functions Swaminathan, Surya Narayanan & Elmasri, Ramez (2016)

3.2.1 Components of Apache hbase Architecture dezyre (2016)

hbase architecture has 3 important components- HMaster, Region Server and ZooKeeper.

3.2.1.1 HMaster hbase HMaster is a lightweight process that assigns regions to region servers in the Hadoop cluster for load balancing. Responsibilities of HMaster –

- Manages and Monitors the Hadoop Cluster
- Performs Administration (Interface for creating, updating and deleting tables.)
- Controlling the failover
- DDL operations are handled by the HMaster
- Whenever a client wants to change the schema and change any of the metadata operations, HMaster is responsible for all these operations.

3.2.1.2 Region Server These are the worker nodes which handle read, write, update, and delete requests from clients. Region Server process, runs on every node in the Hadoop cluster. Region Server runs on HDFS DataNode and consists of the following components –

- Block Cache – This is the read cache. Most frequently read data is stored in the read cache and whenever the block cache is full, recently used data is evicted.

- MemStore- This is the write cache and stores new data that is not yet written to the disk. Every column family in a region has a MemStore.
- Write Ahead Log (WAL) is a file that stores new data that is not persisted to permanent storage.
- HFile is the actual storage file that stores the rows as sorted key values on a disk.

3.2.1.3 Zookeeper hbase uses ZooKeeper as a distributed coordination service for region assignments and to recover any region server crashes by loading them onto other region servers that are functioning. ZooKeeper is a centralized monitoring server that maintains configuration information and provides distributed synchronization. Whenever a client wants to communicate with regions, they have to approach Zookeeper first. HMaster and Region servers are registered with ZooKeeper service, client needs to access ZooKeeper quorum in order to connect with region servers and HMaster. In case of node failure within an hbase cluster, ZKquoram will trigger error messages and start repairing failed nodes. ZooKeeper service keeps track of all the region servers that are there in an hbase cluster- tracking information about how many region servers are there and which region servers are holding which DataNode. HMaster contacts ZooKeeper to get the details of region servers. Various services that Zookeeper provides include –

- Establishing client communication with region servers.
- Tracking server failure and network partitions.
- Maintain Configuration Information
- Provides ephemeral nodes, which represent different region servers.

4 Scaling, Availability and Reliability

A key feature of cassandra and hbase is they are horizontally scalable. Scalability can be defined as the extent to which the system can expand its capacity without interrupting the system. Scaling vertically means adding more power and scaling horizontally means adding more machines 5. Horizontally scalable systems are cost efficient. In order to compete with NoSQL systems, newer relational SQL systems also provide horizontal scalability. Swaminathan, Surya Narayanan & Elmasri, Ramez (2016).

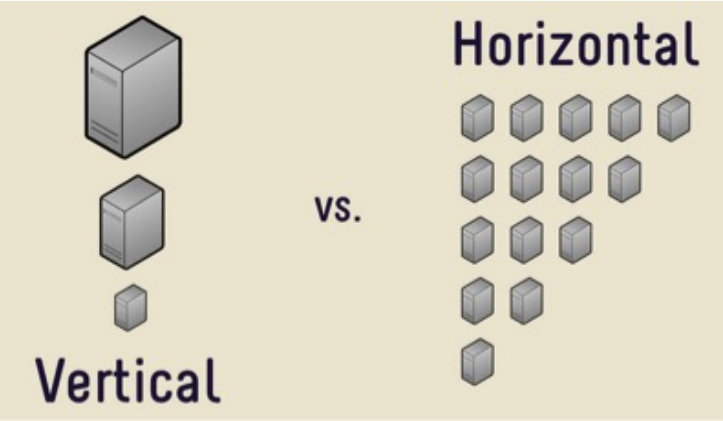


Figure 5: Vertical vs Horizontal Scaling

4.1 cassandra Scaling

Scaling in an existing cassandra system is a matter of adding more nodes. As no single node is a master, when we add more nodes we are improving the capacity of the cluster to support more writes and reads. This type of horizontal scaling allows you to have maximum uptime, as the cluster keeps serving requests from the clients while new nodes are being added to the cluster Sadalage & Fowler 2012

4.2 cassandra Availability

cassandra is by design highly available, since there is no master in the cluster and every node is a peer in cluster. The availability of a cluster can increased by reducing the consistency level of the requests Sadalage & Fowler 2012. cassandra position in relation to the CAP theorem is show in 6.

4.3 cassandra Reliability

cassandra's node replication and eventual consistency features are core to the functioning of this distributed system. These features were designed from the outset and have been improved and battle tested throughout its lifetime and are now considered highly reliable. In fact, cassandra has the reputation of having the most robust, reliable multi-datacenter replication of any datastore in the industry Team (2018)

4.4 hbase Scaling

The basic unit of horizontal scalability in hbase is called a Region. Regions are a subset of the table's data and they are essentially a contiguous, sorted range of rows that are stored together.
Initially, there is only one region for a table. When regions become too large after adding more rows, the region is split into two at the middle key, creating two roughly equal halves.

4.5 hbase Availability

hbase chooses Consistency and Partition Tolerance and compromises on Availability part. hbase position in relation to the CAP theorem is show in 6.

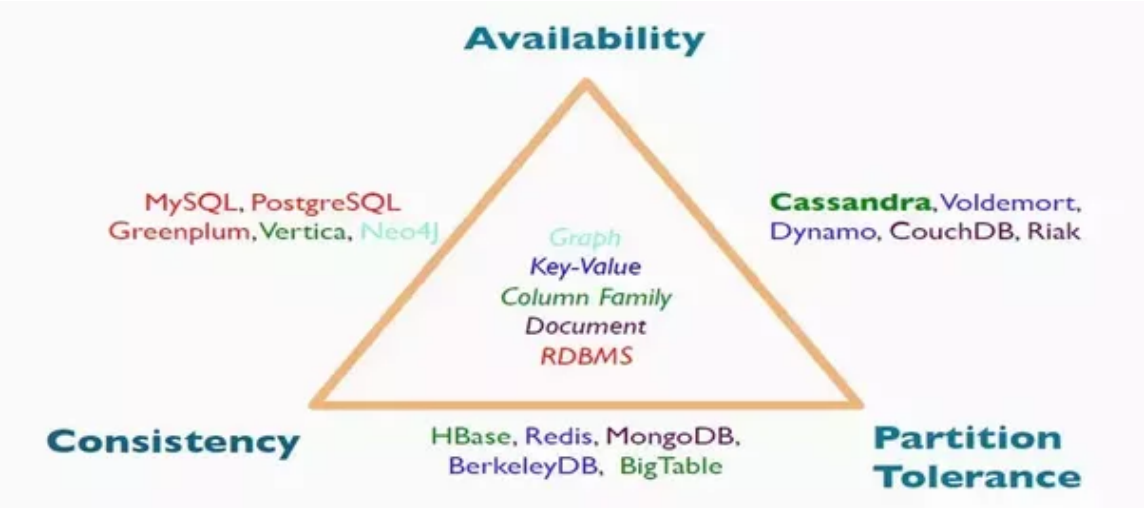


Figure 6: Database positions relative to CAP - courtesy Datalex

4.6 hbase Reliability

HDFS provides high-reliability low-level storage support for hbase.

5 Security

5.1 cassandra

cassandra provides these security features to the open source community DataStax (2019)

5.1.1 Authentication based on internally controlled rolename/passwords

cassandra authentication is roles-based and stored internally in cassandra system tables. Administrators can create, alter, drop, or list roles using CQL commands, with an associated password. Roles can be created with superuser, non-superuser, and login privileges. The internal authentication is used to access cassandra keyspaces and tables, and by cqlsh and DevCenter to authenticate connections to cassandra clusters and sstableloader to load SSTables.

5.1.2 Authorization based on object permission management

Authorization grants access privileges to cassandra cluster operations based on role authentication. Authorization can grant permission to access the entire database or restrict a role to individual table access. Roles can grant authorization to authorize other roles. Roles can be granted to roles. CQL commands GRANT and REVOKE are used to manage authorization.

5.1.3 Authentication and authorization based on JMX username/passwords

JMX (Java Management Extensions) technology provides a simple and standard way of managing and monitoring resources related to an instance of a Java Virtual Machine (JVM). This is achieved by instrumenting resources with Java objects known as Managed Beans (MBeans) that are registered with an MBean server. JMX authentication stores username and associated passwords in two files, one for passwords and one for access. JMX authentication is used by nodetool and external monitoring tools such as jconsole.In cassandra 3.6 and later, JMX authentication and authorization can be accomplished using cassandra's internal authentication and authorization capabilities.

5.1.4 SSL encryption

cassandra provides secure communication between a client and a database cluster, and between nodes in a cluster. Enabling SSL encryption ensures that data in flight is not compromised and is transferred securely. Client-to-node and node-to-node encryption are independently configured. cassandra tools (cqlsh, nodetool, DevCenter) can be configured to use SSL encryption. The DataStax drivers can be configured to secure traffic between the driver and cassandra.

5.1.5 General security measures

Typically, production cassandra clusters will have all non-essential firewall ports closed. Some ports must be open in order for nodes to communicate in the cluster. These ports are detailed.

5.2 hbase

hbase supports both map-reduce task for token based authentication as well as user authentication tutorialspoint.com (2019).
Authentication of user is performed by making use of SASL i.e Simple Authentication and Security Layer with the help of Kerberos which is basically done on an individual connection criteria.
Furthermore, additional security to hbase are managed by Access COntrol List (ACL).Logging support up to fata node level is also provided by hbase.
High level auditing and monitoring which are are absent in hbase, as is security configurations and encryption of data at rest.Vishwakarma (2018).

6 Literature Review

In 2008 Google published big table Chang et al. (2008) which described databases running on clusters. This led the way for databases like cassandra and hbase which are NoSQL databases running on top of a cluster of machines. One big advantage of these databases is that they scale horizontally rather than vertically. This means that when more data storage is required the user can buy more machines rather than having to buy a bigger machine, which is more economical.
Both cassandra and hbase are classed as NoSQL databases and the description of key characteristics of these type of databases are well described in the book NoSQL distilled Sadalage & Fowler (2012).
The differences in cassandra and hbase can be found in the paper Swaminathan, Surya Narayanan & Elmasri, Ramez (2016). This explained how hbase and cassandra were descended from bigpaper Chang et al. (2008).
Benchmarking was carried out by downloading the widely used YCSB tool which is available at Cooper (2019) and described in the paper Cooper et al. (2010)
Engineers at the University of Toronto who did a benchmarking using YCSB declared cassandra the best performing database when compared against several other databases including hbase. This is described in the paper Tilmann Rabl Mohammad Sadoghi Hans-Arno Jacobsen et al. (2012) and summarised at of Toronto (2012).
A more recent comparison which compared MongoDB, cassandra and hbase also evaluated using YCSB Swaminathan, Surya Narayanan & Elmasri, Ramez (2016) gave a more subtle conclusion. The report concludes that "different NoSQL databases have different design features, which are advantageous for specific types of operations and it is necessary to examine the behaviour of different NoSQL databases in order to make an informed decision on the choice of database suitable for an application" . With respect to cassandra and hbase results it can be seen that hbase performs better overall for blind writes however in general cassandra performs better in most other categories for larger databases. The results are more nuanced for smaller databases. The results of this experiment agree with the conclusions in the papers above. The difference in cassandra and hbase architecture was researched in several papers. Much information was available about hbase in the papers Parab (2018) and Kailas (2018). Information on cassandra was available Parab (2018) and on the DataStax website which supports the open source cassandra DataStax (2019).

7 Performance Test Plan

The latency 7.1 and throughput 7.2 of the 2 databases were measured using the YCSB tool. YCSB measured the throughput, READ latency, INSERT latency and UPDATE latency while gradually increasing the number of records per second.

7.1 Latency

Latency is the time between making a request and beginning to see a result. Some define latency as the time between making a request and the completion of the response, but this definition does not clearly distinguish the psychologically significant time spent waiting, not knowing whether a request has been accepted or understood. You will also see latency defined as the inverse of throughput, but this is not useful because latency would then give you the same information as throughput. Latency is measured in units of time, such as seconds Killelea (2002).

7.2 Throughput

Throughput is the number of items processed per unit time, such as bits transmitted per second, HTTP operations per day, or millions of instructions per second (MIPS). It is conventional to use the term "bandwidth" when referring to throughput in bits per second. Throughput is found by adding up the number of items and dividing by the sample interval. This calculation may produce correct but misleading results because it ignores variations in processing speed within the sample interval Killelea (2002)

7.3 Why Use YCSB to measure performance?

There are several options available to evaluate the performance of databases, so why use YCSB? For example, the TPC-H benchmark suite is commonly used to benchmark relational databases, and some NoSQL systems include native tools for measuring throughput and latency. (hbase comes packaged with the PerformanceEvaluation utility.) However, most of these approaches don't work with other systems and often focus on scenarios in which the data store excels. From the perspective of a generic, database-neutral, performance evaluation utility, YCSB is currently the de-facto comparative benchmark for NoSQL stores. It includes support for a wide range of database bindings and is commonly used to compare their performance for a set of desired workloads. Being open source and extensible, support for additional databases is regularly added Kamat (2015).

7.4 Experimental Setup

hbase and cassandra was setup on a server running Ubuntu 18.04. The system specifications are...

- ASUS-NotebookSKU (UX305FA)
- Intel(R) Processor 5Y10 CPU @ 0.80GHz
- ATA Disk
- 8GB RAM

- Ubuntu 18.04
- Hadoop-2.8.5
- hbase-2.2.0 [1]
- YCSB-0.14.0
- cassandra version 3.9
- openjdk version "1.8.0_212"

7.5 YCSB Setup

The Yahoo Cloud Serving Benchmark (YCSB) Cooper (2019) is a benchmarking tool developed by yahoo and presented in 2010 to compare different databases. The architecture of YCSB is shown in illustration 7.1 which is from the original YCSB paper Cooper et al. (2010).
After the cassandra and hbase databases are configured as desired the YCSB client must be configured to decide what tests are to be run.
YCSB provides a command line to run tests and testharness which is a bash shell script which generates the appropriate command lines provided by YCSB. It was necessary to configure the testharness for appropriate workloads and records counts.
Although YCSB can be used in two phases "load" and "run" only the run data was analysed as this was sufficient to cover all measurements necessary i.e Throughput, READ latency, UPDATE latency and INSERT latency.
The subsequent results were evaluated in section 8

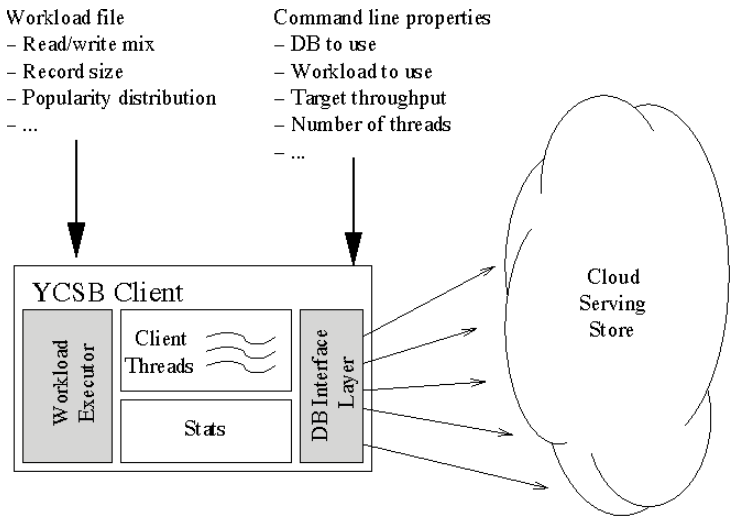


Figure 7: YCSB Architecture

[1] hbase in Pseudo-distributed mode

7.5.1 Workloads considered for testing hbase and cassandra Database performance

YCSB provides 6 workloads. Two workloads A and D were tested used in the tests...

- Workload A
- Workload D

7.5.2 Record Counts used for Workloads

Loading is adjusted by modifying the "record count" property in YCSB. These record counts are the number of records loaded or run on the database per second.
Performance was tested twice. Once with records up to 300 thousand and the second up to 2.5 million 7.5.2

Low Workload	High Workload
100,000	500,000
150,000	1,000,000
200,000	1,500,000
250,000	2,000,000
300,000	2,500,000

Table 1: record counts per second

8 Evalutation and Results

Performance of two database systems i.e hbase and cassandra are compared by making use of YCSB. Two different YCSB workloads (A and D) were applied Cooper et al. (2010) as these covered operations of interest (latency, throughput) .
Two sets of records were applied 7.5.2 A lower value of records were measured 8.1.1, 8.1.2 and then loads were then increased by a factor of 10 7.5.2 to see how the databases would react. A python script was written to extract the data into csv files 9. A second script was then written to present the graphs using the python library matlabplot 9.

The measurement units are...
Throughput measured in the number of operations per second.
Latencies are all measured in microseconds.
The results show that when reading hbase is generally faster than cassandra except with workload A "Update heavy workload" 8 where the latency is slightly higher than cassandra.
cassandra always has better latency when writing (update and insert).

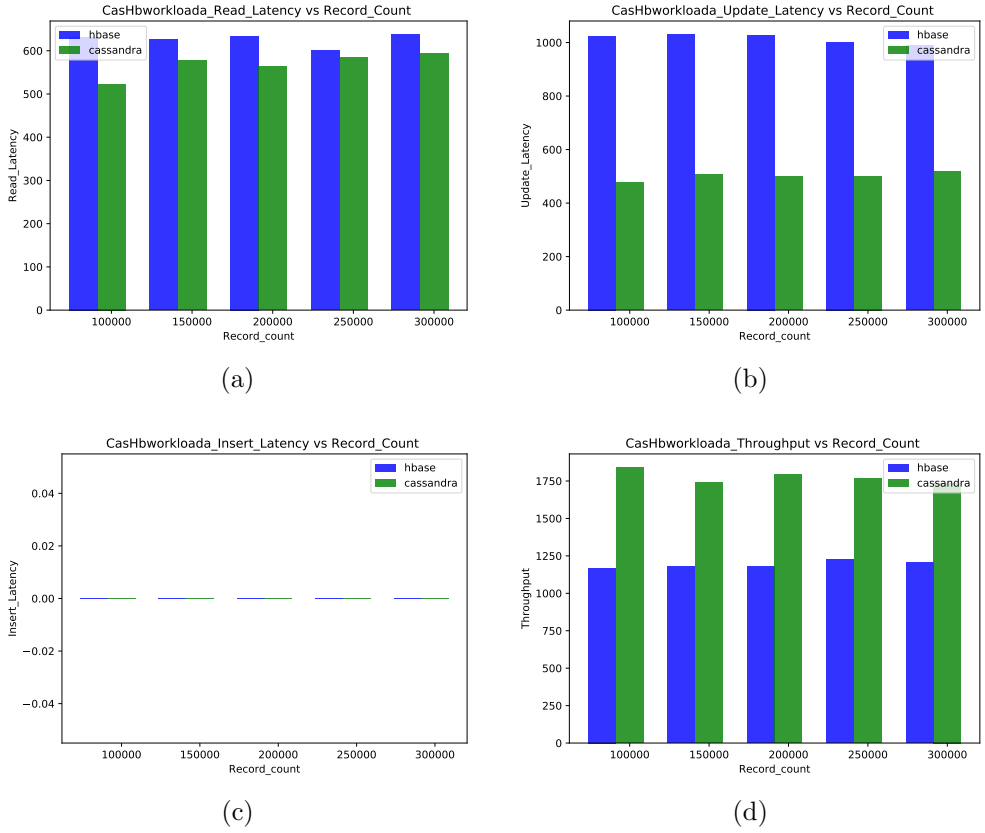
8.1 Workload Results : Low Record Count

Workloads were run with an increasing **low** number of record counts per seconds 7.5.2

8.1.1 Workload A: Update heavy workload

This workload has a mix of 50/50 reads and writes. An application example is a session store recording recent actions.

8.1.1.1 Result In this hbase has worse latency than cassandra for reading and updating hbase also has a slower throughput. Insertion latency is not recorded.



(a) (b)

(c) (d)

Figure 8: workloada Read Latency(a), Update Latency (b), Insert Latency(c), Throughput(d)

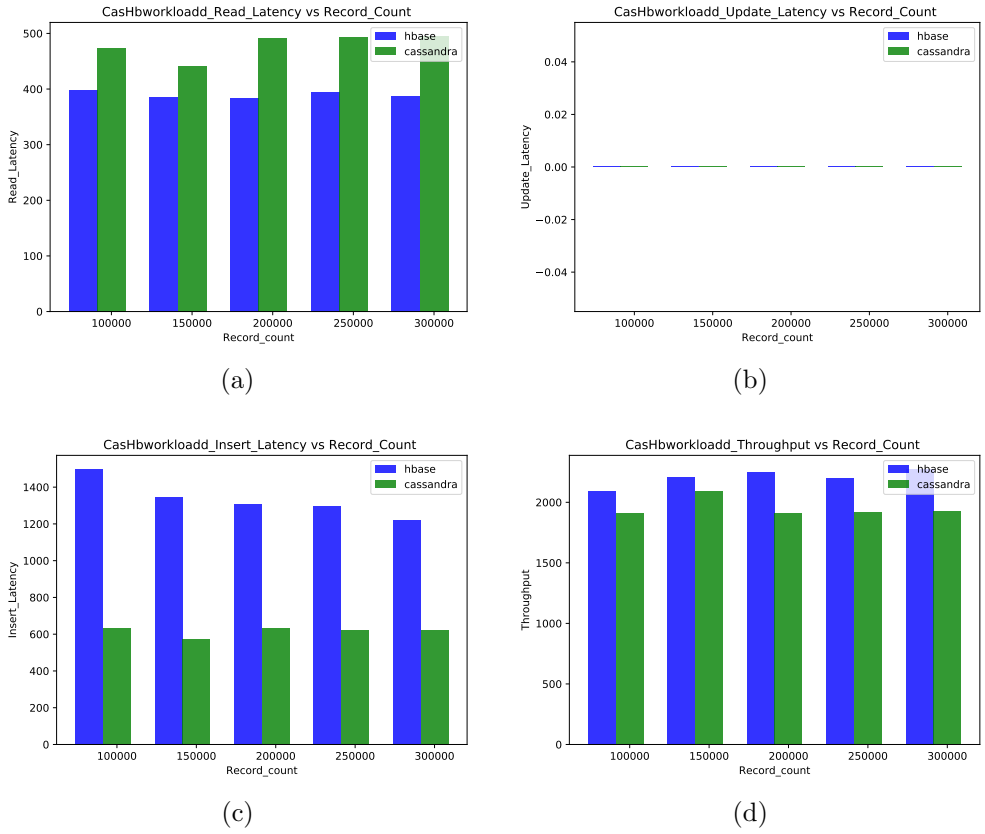
Database	Record_count	Throughput	Read_Latency	Update_Latency	Insert_Latency
cassandra	100000	1839	523	478	0
hbase10	100000	1169	632	1024	0
cassandra	150000	1743	577	507	0
hbase10	150000	1178	626	1029	0
cassandra	200000	1793	563	500	0
hbase10	200000	1178	634	1028	0
cassandra	250000	1769	584	501	0
hbase10	250000	1225	600	1002	0
cassandra	300000	1737	594	517	0
hbase10	300000	1206	637	989	0

Table 2: CasHbworkloada

8.1.2 Workload D: Read latest workload

In this workload, new records are inserted, and the most recently inserted records are the most popular. Application example: user status updates; people want to read the latest.

8.1.2.1 Result In this hbase has worse latency than cassandra for insertion but better latency for reading. hbase also has a faster throughput than cassandra. Update latency is not recorded.



(a) (b)

(c) (d)

Figure 9: workloada Read Latency(a), Update Latency (b), Insert Latency(c), Throughput(d)

Database	Record_count	Throughput	Read_Latency	Update_Latency	Insert_Latency
cassandra	100000	1908	474	0	632
hbase10	100000	2091	398	0	1498
cassandra	150000	2090	442	0	575
hbase10	150000	2206	386	0	1349
cassandra	200000	1915	491	0	631
hbase10	200000	2247	383	0	1308
cassandra	250000	1919	494	0	621
hbase10	250000	2201	395	0	1299
cassandra	300000	1924	495	0	623
hbase10	300000	2274	387	0	1222

Table 3: CasHbworkloadd

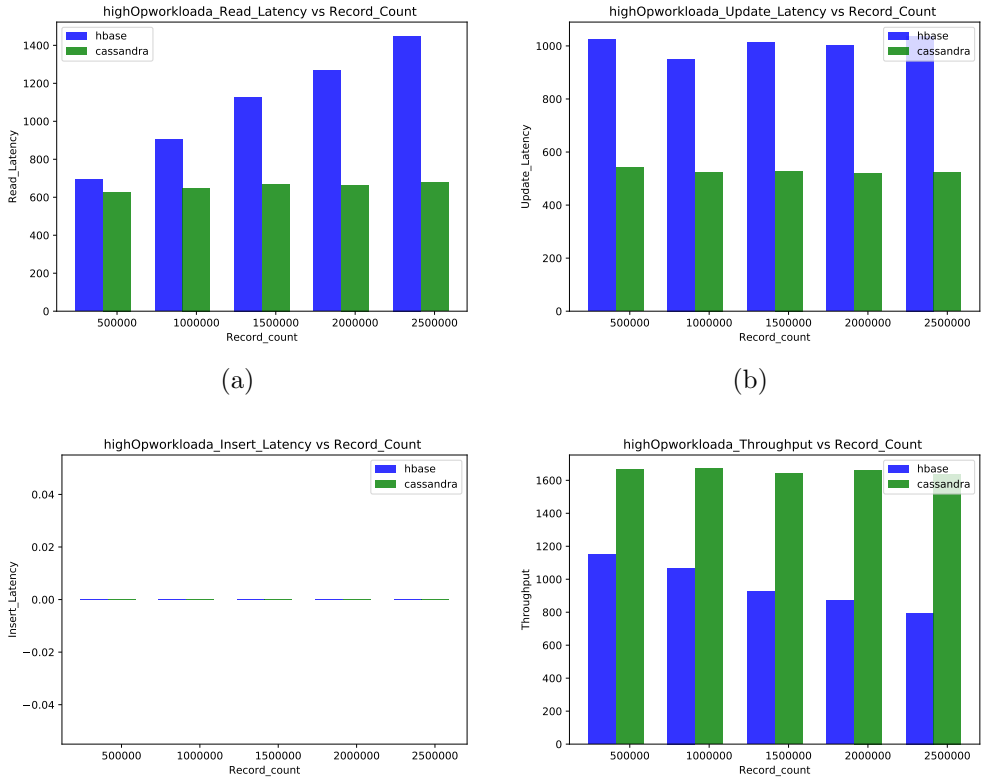
8.2 xWorkload Results : High Record Count

The Workloads were run with an increasing **high** volume of record counts 7.5.2

8.2.1 xWorkload A: Update heavy workload

This workload has a mix of 50/50 reads and writes. An application example is a session store recording recent actions.

8.2.1.1 Result In this case hbase has worse latency than cassandra for reading and uprating hbase also has a slower throughput. Insertion latency is not recorded.As the load increases read latency and throughput degrade. cassandra's performance is better for latency and throughput and does not degrade.



(a) (b)

(c) (d)

Figure 10: workloada Read Latency(a), Update Latency (b), Insert Latency(c), Throughput(d)

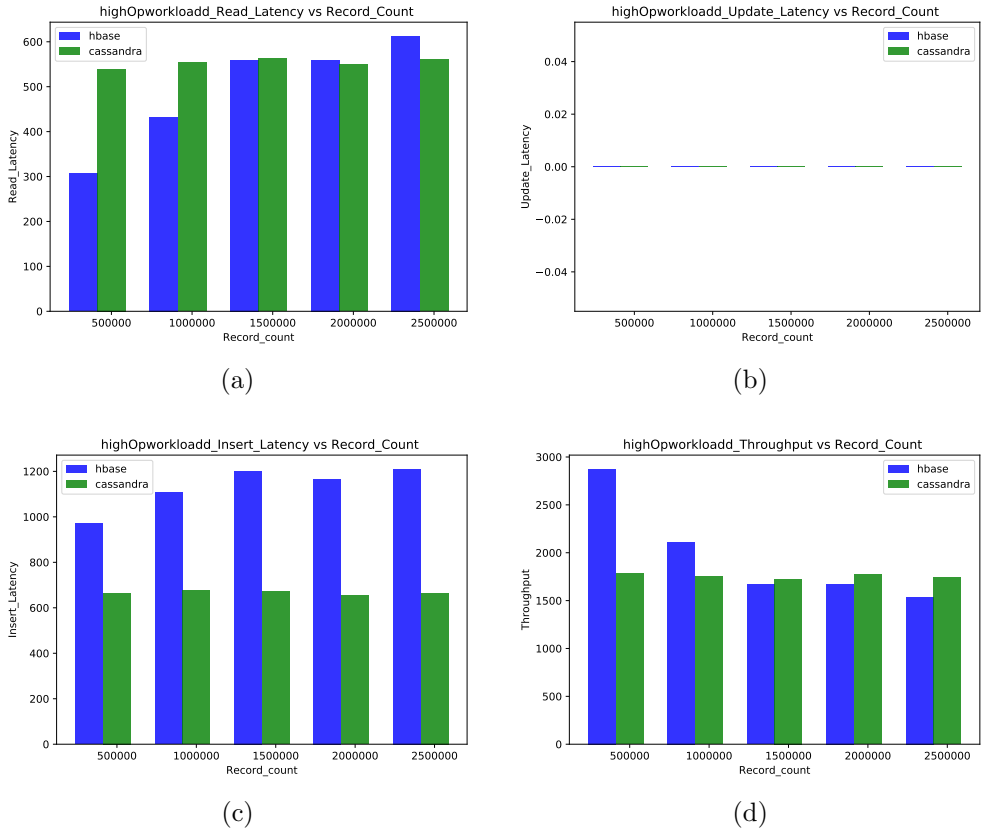
Database	Record_count	Throughput	Read_Latency	Update_Latency	Insert_Latency
cassandra	1000000	1670	648	525	0
hbase10	1000000	1068	903	950	0
cassandra	1500000	1640	670	528	0
hbase10	1500000	926	1128	1013	0
cassandra	2000000	1663	664	518	0
hbase10	2000000	873	1269	1001	0
cassandra	2500000	1639	679	522	0
hbase10	2500000	796	1450	1038	0
cassandra	500000	1666	628	541	0
hbase10	500000	1149	694	1024	0

Table 4: highOpworkloada

8.2.2 xWorkload D: Update heavy workload

This workload has a mix of 50/50 reads and writes. An application example is a session store recording recent actions.

8.2.2.1 Result In this case hbase has better latency than cassandra for reading and throughput is higher at the start but this degrades as the number of records increase until it is slightly worse than cassandra. Insertion latency is always worse for hbase.cassandra's performance stays constant for both latency and throughput.



(a) (b)

(c) (d)

Figure 11: workloadd Read Latency(a), Update Latency (b), Insert Latency(c), Throughput(d)

Database	Record_count	Throughput	Read_Latency	Update_Latency	Insert_Latency
cassandra	1000000	1751	554	0	676
hbase10	1000000	2113	433	0	1106
cassandra	1500000	1728	563	0	672
hbase10	1500000	1670	559	0	1201
cassandra	2000000	1774	549	0	655
hbase10	2000000	1675	560	0	1166
cassandra	2500000	1741	561	0	665
hbase10	2500000	1538	613	0	1211
cassandra	500000	1785	540	0	665
hbase10	500000	2876	307	0	971

Table 5: highOpworkloadd

9 Conclusion and Discussion

In this paper, cassandra and hbase were compared with the use of YCSB. The performance of both databases was compared at **low load** (10,000-300,000) records second and **high load** (500,000-2,500,000) records per second. Four main performance measures were compared ...

- Read Latency
- Update Latency
- Insert Latency
- Throughput

At **low load** results hbase generally performed worse than cassandra with latency although with read type workloads it performed better with read latency. hbase also had slightly faster throughput for workloads with more read operations but had worse throughput for workloads with write operations.
At **high load** the latency for hbase could be seen to degrade as the load increased. The throughput also degraded in hbase. cassandra's performance did not degrade at these same higher loads.
In general hbase seems to perform worse with writes than cassandra and better with reads. This agrees with many of the previous papers cited in the literary review. At the higher loads we tested hbase performance degraded but cassandra did not. In future it may be useful to add even higher loads to see at what point cassandra starts to degrade.

References

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A. & Gruber, R. E. (2008), 'Bigtable: a distributed storage system for structured data', *ACM Transactions on Computer Systems* (2), 4.

Cooper, B. (2019), 'Yahoo! Cloud Serving Benchmark. Contribute to brianfrankcooper/YCSB development by creating an account on GitHub'. original-date: 2010-04-19T20:52:11Z.
URL: *https://github.com/brianfrankcooper/YCSB*

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R. & Sears, R. (2010), Benchmarking Cloud Serving Systems with YCSB, *in* 'Proceedings of the 1st ACM Symposium on Cloud Computing', SoCC '10, ACM, New York, NY, USA, pp. 143–154. event-place: Indianapolis, Indiana, USA.
URL: *http://doi.acm.org/10.1145/1807128.1807152*

DataStax (2019), 'Securing Cassandra'.
URL: *https://docs.datastax.com/en/archived/cassandra/3.0/cassandra/configuration/secureIntro.ht*

dezyre (2016), 'Overview of HBase Architecture and its Components'.
URL: *https://www.dezyre.com/article/overview-of-hbase-architecture-and-its-components/295*

Kailas, G. (2018), '(PDF) A Comparison Of NoSQL Database Systems: A Study On MongoDB and Apache HBase'.
URL: *https://www.researchgate.net/publication/330937764_A_Comparison_Of_NoSQL_Database_Sys*

Kamat, G. (2015), 'YCSB, the Open Standard for NoSQL Benchmarking, Joins Cloudera Labs'.
URL: *https://blog.cloudera.com/blog/2015/08/ycsb-the-open-standard-for-nosql-benchmarking-joins-cloudera-labs/*

Killelea, P. (2002), 'Web Performance Tuning, 2nd Edition [Book]'.
URL: *https://www.oreilly.com/library/view/web-performance-tuning/059600172X/*

kumar Dey, T. (2019), '(PDF) Comparative Study between Hbase and Cassandra'.
URL: *https://www.researchgate.net/publication/333296441_Comparative_Study_between_Hbase_and_C*

Lakshman, A. & Malik, P. (2010), 'Cassandra: A Decentralized Structured Storage System', *SIGOPS Oper. Syst. Rev.* **44**(2), 35–40.
URL: *http://doi.acm.org/10.1145/1773912.1773922*

of Toronto, U. (2012), 'Apache Cassandra NoSQL Performance Benchmarks'.
URL: *https://academy.datastax.com/planet-cassandra/nosql-performance-benchmarks*

Parab, N. N. (2018), '(PDF) HBase v/s MongoDB'.
URL: *https://www.researchgate.net/publication/330675909_HBase_vs_MongoDB*

Sadalage, P. J. & Fowler, M. (2012), *NoSQL distilled : a brief guide to the emerging world of polyglot persistence*, Boston, Mass. : Addison-Wesley, 2012.

Swaminathan, Surya Narayanan & Elmasri, Ramez (2016), 'Quantitative Analysis of Scalable NoSQL Databases', *2016 IEEE International Congress on Big Data (BigData Congress), Big Data (BigData Congress), 2016 IEEE International Congress on, bigdatacongress* p. 323.

Team, V. W. (2018), 'Why we switched to Cassandra'.
URL: *https://victorops.com/blog/cassandra*

Tilmann Rabl Mohammad Sadoghi Hans-Arno Jacobsen, Middleware Systems, Research Group, University of Toronto, Canada Middleware Systems, Research Group, University of Toronto, Canada Middleware Systems, Research Group, University of Toronto, Canada, tilmann@msrg.utoronto.ca, Sergio Gómez-Villamor mo@msrg.utoronto.ca, Victor Muntés-Mulero arno@msrg.utoronto.ca, Serge Mankovskii, DAMA-UPC, Universitat Politècnica de, Catalunya, Spain CA Labs Europe, Barcelona, Spain CA Labs, San Francisco, USA, sgomez@ac.upc.edu, victor.muntes@ca.com & serge.mankovskii@ca.com (2012), 'Solving Big Data Challenges for Enterprise Application Performance Management'.

tutorialspoint.com (2019), 'HBase Overview'.
URL: *https://www.tutorialspoint.com/hbase/hbase_overview*

Vishwakarma, P. (2018), 'Comparitive Performance Analysis of MongoDB and HBase on YCSB'.
URL: *https://www.researchgate.net/publication/330823015_Comparitive_Performance_Analysis_of_M*

Appendix

Python code to extract data into csv file

```
#!/usr/bin/python3
# Author: Martin Mohan
# Date: 10/08/2019
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import csv,sys,os,re
import matplotlib.pyplot as plt; plt.rcdefaults()

# Return a list of files from directory
def get_filelist(dirname):
    files=[]
    for r, d, f in os.walk(dirname):
        for file in f:
            if '.txt' in file:
                files.append(os.path.join(r, file))
    return files

#  if list l string else -1
def find_strings(l,s):
    if all(x in s for x in l):
        return s
    else:
        return -1

# find float in string
def get_float(s):
    matches=re.findall("[+-]?\d+\.\d+",s)
    if(len(matches)>1):
        print('get_float: Error more than one matches: {0}'.format(matches))
    return(int(float(matches[0])))

# find fname (e.g. workloada) in files and extract avalues
def get_array(files,fname):
    i=0
    for f in files:
        reccount=[int(s) for s in f.split("_") if s.isdigit()][0]
        # found *db_worloadx* e.g. *hbase10_worloada*
        if all(x in f for x in [fname,"run"]):
            i=i+1
    #         print('found: f1={0} f={1}'.format(f1,f))
            f2 = open(f, 'r')
            lines = f2.read().splitlines()
            f2.close()
            for s in lines:
    #             print('reccount: {0}'.format(reccount))
```

```
                if all(x in f for x in ["cassandra"]):
                    arr[i][0]="cassandra"
                else:
                    arr[i][0]="hbase10"
#                 arr[i][0]=fname
                arr[i][1]=reccount
                if all(x in s for x in ["Throughput"]):
                    arr[i][2]=get_float(s)
                if all(x in s for x in ["AverageLatency","READ"]):
                    arr[i][3]=get_float(s)
                if all(x in s for x in ["AverageLatency","UPDATE"]):
                    arr[i][4]=get_float(s)
                if all(x in s for x in ["AverageLatency","INSERT"]):
                    arr[i][5]=get_float(s)
    return(arr)


argc=len(sys.argv)
dirname="ycsb_dir"

if argc < 2:
    print('Usage␣{0}␣{1}'.format(sys.argv[0],dirname))
    print('Extract␣ycsb␣performance␣files␣into␣csv␣files')
    sys.exit()
else:
    dirname=sys.argv[1]

if(os.path.isdir(dirname)==False):
    print('{0}␣should␣be␣a␣valid␣directory'.format(dirname))
    sys.exit()


files=get_filelist(dirname)

def init_array():
    rows, cols = (11, 6)
    arr = [[0 for i in range(cols)] for j in range(rows)]
    arr[0][0]="Database"
    arr[0][1]="Record_count"
    arr[0][2]="Throughput"
    arr[0][3]="Read_Latency"
    arr[0][4]="Update_Latency"
    arr[0][5]="Insert_Latency"
    return arr

#tables=["cql_workloada","cql_workloadd","hbase10_workloada","hbase10_worklo

#tables=["workloada","workloadd"]
tables=["workloada","workloadb","workloadc","workloadd","workloade","workloa
#tables=["workloadd"]
for fname in tables:
```

```
    arr=init_array()
    myarr=get_array(files,fname)
#    df=np.array(myarr)
#    print(df)
    myfile=dirname+"_"+fname+'.csv'
    wtr = csv.writer(open (myfile, 'w'), delimiter=',', lineterminator='\n')
    for x in myarr : wtr.writerow (x)
    print(myfile)
```

Python code plot graph using matplot

```
#!/usr/bin/python3
# Author: Martin Mohan
# Date: 10/08/2019
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sys,os

fname="xxxxxx"
ytitle="xxxxxxxx"
colnames = ["Database","Record_count","Throughput","Read_Latency","Update_La
workloads = ["workloada","workloadb","workloadc","workloadd","workloade","wo
if len(sys.argv) < 3:
    print('Usage␣{0}␣{1}␣{2}'.format(sys.argv[0],fname,ytitle))
    print('Usage␣{0}\n␣{1}\n␣{2}'.format(sys.argv[0],workloads,colnames))
    sys.exit()
else:
    fname=sys.argv[1]
    ytitle=sys.argv[2]

if(os.path.isfile(fname)==False):
    print('{0}␣should␣be␣a␣valid␣file'.format(fname))
    sys.exit()


#colnames = ["Database","Record_count","Throughput","Read_Latency","Update_L
df = pd.read_csv(fname, names=colnames)

#ytitle="Read_Latency"
title=os.path.splitext(fname)[0]+"_"+ytitle
print(ytitle)

#cols = df.columns.drop('Database')
#df[cols] = df[cols].apply(pd.to_numeric, errors='coerce')
cols = ["Record_count","Throughput","Read_Latency","Update_Latency","Insert_
df[cols] = df[cols].apply(pd.to_numeric, errors='coerce', axis=1)
# Sort
df = df.sort_values(["Record_count","Database"])
#print(df)
```

```
# Seperate hbase10 and cassandra
hb1 = df[df['Database'] == "hbase10"]
yc1 = df[df['Database'] == "cassandra"]
rc = yc1.Record_count.tolist()
# See 'title'

hbx="hb1."+ytitle+".tolist()"
casx="yc1."+ytitle+".tolist()"
#print(hbx)
#print(casx)
hb=eval(hbx)
cas=eval(casx)
#hb = hb1.Read_Latency.tolist()
#cas = yc1.Read_Latency.tolist()

yh = list(map(int,hb))
yc = list(map(int,cas))
n_groups = len(yh)
x1 = list(map(int,rc))

# create plot
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.35
opacity = 0.8

rects1 = plt.bar(index, yh, bar_width,
alpha=opacity,
color='b',
label='hbase')

rects2 = plt.bar(index + bar_width, yc, bar_width,
alpha=opacity,
color='g',
label='cassandra')

plt.xlabel('Record_count')
plt.ylabel(ytitle)
plt.title(title+' vs Record_Count ')
plt.xticks(index + bar_width, x1)
plt.legend()

plt.tight_layout()
plt.savefig(title+'.pdf')
#plt.show()
```