**Data Storage & Management**

**project on**

**Analysis of HBase & Cassandra**

**Smit Jain**

**x18135340**

**MSc Data Analytics – 2019/20**

**Submitted to: Noel Cosgrave**

Table of Contents

[**Abstract** 3](#_Toc8246850)

[**Introduction** 3](#_Toc8246851)

[**Key characteristics of Cassandra & HBase** 4](#_Toc8246852)

[**Cassandra: Overview** 4](#_Toc8246853)

[**HBase: Overview** 6](#_Toc8246854)

[**Database Architecture** 7](#_Toc8246855)

[**Cassandra** 7](#_Toc8246856)

[**HBase** 9](#_Toc8246857)

[Scalability, Availability & Reliability of Cassandra & HBase 10](#_Toc8246858)

[Security in Cassandra & HBase 11](#_Toc8246859)

[Literature surveys 13](#_Toc8246860)

[Performance test plan 15](#_Toc8246861)

[Evaluation & results 17](#_Toc8246862)

[Conclusion & Discussion 25](#_Toc8246863)

[References 26](#_Toc8246864)

# **Abstract**

In the modern world and the present time, most of the applications are using NonSQL or NoSQL databases because of the salient features and the properties of the database (non-relational, horizontally scalable and distributed). The factors which makes these databases stand out of the rest other databases are - non-relational, schema-free, simple API, no join, easy replication support and are eventually very much consistent. These databases are highly scalable w.r.t their performances. As there are many options available in NoSQL databases, we are going to compare a few factors which are relatable to the performance of these NoSQL databases. In this project, HBase and Cassandra databases will be compared and analysed and later, performance of each of the two databases will be evaluated. Both the databases will be compared and analysed on various factors like scalability, availability, reliability, security, storage and retrieval process of the databases of different types. The environment where the tests have been done is Ubuntu version – 18.04, where Hadoop, HBase and Cassandra have been installed. Hence, for comparing the performance of both the databases, I have installed YCSB also known as Yahoo! Cloud Serving Benchmark. I’ll be using YCSB for comparing and analysing the relative-performance of the NoSQL databases on the points discussed earlier in this abstract.

# **Introduction**

NoSQL generally refers to a type of storage engines which use non-relational format to store data in them. Traditional RDBMS, on the contrary, uses tables to store data in the tables format where the data can be related in the form of primary or foreign keys. The NoSQL databases are primarily of 4 different types depending upon the type of requirement and hence the job of each of the NoSQL databases is different. The types of databases are: Column oriented, Key-Value stores, Document stores and Graph DB. The high-level overview of all these types are as follows:

1. **Column oriented**: These types of databases use columns to store the data which allows the user to store data which is very large in size of rows. Example: **HBase**.
2. **Key-Value stores**: In these types of databases, for all the binary objects, a key is defined which is used to store serialized objects of large size. Example: **Cassandra**
3. **Document stores**: These types of databases store data as documents in the structured format which are similarly based upon the key value system. Example: **CouchDB** & **MongoDB**.
4. **Graph DB**: These types of databases stores Graphs in the data format and stores relationships of different objects. Example: **Neo4j**.

In NoSQL databases, data manipulation of any kind is dependent on the application, which makes the NoSQL database to be a perfect storage engine. Here, the question arises - what is NoSQL used for? So, to clearly answer this question, we can think about NoSQL as a database which can perform read & write operations on a very large amount of data very quickly. To improve the performance of these databases, we can add a few more servers and see the performance uplifted with just a little effort, because of the fact that NoSQL databases scale horizontally. The NoSQL databases comes with schema-less data-model.

The main characteristics to consider when choosing a NoSQL database are as follows: no schema to consider, no unused cell, no datatype, application layer has most of the considerations carried out in and all the items are taken as an aggregate document. The consistency models of both RDBMS and NoSQL act as the major difference between the two. To allow the horizontal scalability, NoSQL has given relaxation on ACID guarantees which are given by the relational transactions.

# **Key characteristics of Cassandra & HBase**

## **Cassandra: Overview**

Cassandra is a system with distributed storage which offers high availability of service with a no point of failure and at the same time it manages large volume data which is structured data and is distributed amongst various commodity servers.

To offer high level of availability, scalability and fault tolerance, Cassandra is developed to run simultaneously on a number of nodes. Cassandra works on a multi-master mode where the writes are distributed in between all the nodes with the help of hash function while reads are directed to specific nodes.

In Cassandra, the writes are channelled towards other available nodes in case one of the nodes goes down and the system as a whole continues to operate. Since Cassandra is designed as a Multi-master architecture, it is linearly scalable, and it can handle twice number of writes just by doubling the nodes number. It also stores incoming write operations directly into RAM in order to provide better and speedy performance.

The Key features of Cassandra are as follows:

1. **Open source**

Cassandra is an open source project developed by Apache. It is having a very big community of developers who discuss their views and queries on a common forum. Cassandra can also be integrated with other open-source projects by Apache for example Hadoop, Hive etc.

1. **Peer-to-peer architecture**

Apache developed Cassandra which allows peer-to-peer communication between the nodes, also known as peer-to-peer architecture.

1. **Elastic scalability**

Cassandra provides the greatest advantage of elastic scalability by which one can scale-up / scale-down the available clusters. The number of nodes can be added or deleted from the cluster easily without any disturbances and can be done without restarting the cluster which results in a very high throughput in Cassandra.

1. **High availability & Fault tolerance**

Cassandra is fault tolerant and shows high availability due to the data replication. In case of failure at one node, the data can be found available readily at other nodes from where the data can be retrieved easily.

1. **High performance**

The base reason for the development of Cassandra was to use multi-core machines and maximize the capabilities. When a large set of data is used, Cassandra proves to be highly reliable.

1. **Tuneable consistency**

Strong consistency and Eventual consistency are the types of consistencies in Cassandra. As per the requirement, the developer can make a choice. The eventual consistency gets the required client approval as soon as the write is accepted by the cluster. On the contrary, Strong consistency broadcasts all the updates to all the available nodes or machines, suitable to any particular data.

## **HBase: Overview**

Apache HBase is Hadoop's and HDFS (Hadoop Distributed File System's) most famous non-relational databases. It is also recognized as the database of Hadoop. HBase is also an open-source NoSQL DB written in the Java language.

HBase is built on the Bigtable concepts of Google. It is suitable for use in cases where access to large volumes of data (big data) in real time and random read/write is needed. The performance is deeply dependent on hardware support as HBase is based on top of HDFS. To achieve a better performance, we need to have sufficient number of available nodes (minimum 5).

HBase being a NoSQL database, works on HDFS, which sometimes leads people to think that HBase is a replacement or a replacement for HDFS. They're fundamentally different, though. HDFS is a distributed storage that spans multiple hardware of commodities. It is the Hadoop file system and for any type of Hadoop application it works as a generic storage. But HBase is a non-relational database that stores its data using HDFS. It can be compared in normal / local file system with any relational database and its storage. We can conclude, therefore, that HBase is not a substitute, but they work together and complement each other.

HBase scales linearly, so there should be a primary key for all tables. All the key spaces are distributed in sequential blocks contains and regions are allocated to these blocks. Now, in a clustered environment, these regions are controlled by Region Servers to distribute the load evenly. HBase supports automatic data sharding, so there is no need for manual intervention.

The key characteristics of HBase are listed below:

*Scalability*

*Sharding*

*Consistency*

*Real time processing*

*Distributed Storage*

*Failover support*

*API support*

*MapReduce support*

*Backup support*

# **Database Architecture**

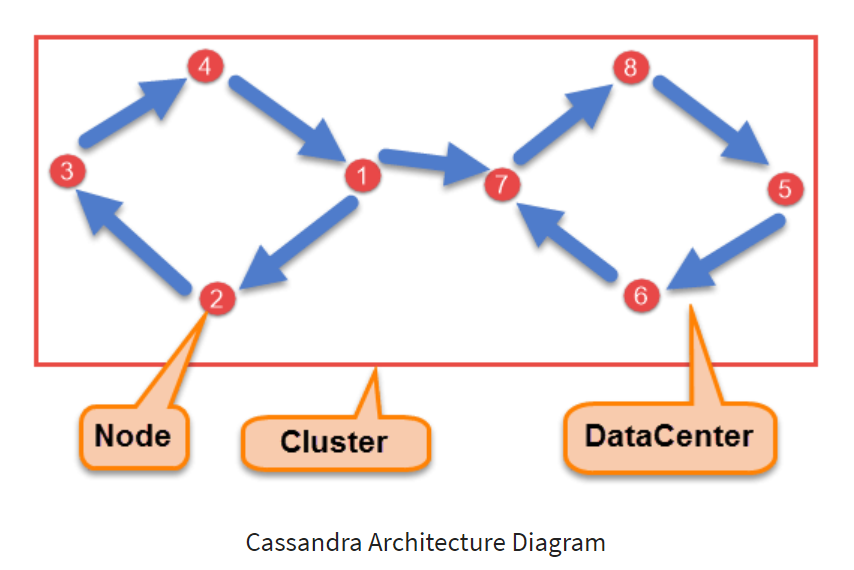
## **Cassandra**

The storage system architecture that operates in the production system setting is complex. The system needs to have following qualities in addition to actual component of data persistence: scalable & robust, failure recovery, membership & failure detection, replica sync, state transfer, overload handling, job assigning, routing the received requests and marshalling, system done by the monitoring and alarming.

All of these modules work synchronously to handle requests for reading / writing. A request for a key to read / write is typically channelized to run time selected nodes in the cluster. Thereafter nodes calculate the count of replica for the specific key. The system assigns the requests to the aforementioned created replicas for writes and other CRUD operations for a minimum of replicas to get done the completion of the writings. For readings, the system either routes the requests to the nearest replica or routes the requests to all replicas and waits for a quorum of replies.

Cassandra has been designed to handle large data workloads with no single point of failure across multiple nodes. Its architecture has been centred on the understanding that failures in the system and hardware can and do happen. Cassandra addresses the issue of failures by using a peer-to-peer distributed system throughout the homogeneous nodes where data is distributed across all cluster nodes.

In Cassandra, the rows are organized in the form of tables where primary key is required since it is a partitioned row database. The architecture of Cassandra serves the authorization functionality using CQL language to connect the user to the system and grant them the access of data using the to any node in any datacenter. CQL is using the same syntax to SQL to make it easy to use and works with table data. Developers can use cqlsh, DevCenter, and application language drivers to access CQL. A cluster typically has one key space per application consisting of many different tables. Requests for the client to read or write can be sent to any cluster node. When a client connects with a request to a node, that node acts as the coordinator for the particular operation of the client. The coordinator functions as a proxy terminal between the application of the client and the nodes which own the requesting data.



***Node***

A junction where data gets stored is generally called as a node. It's Cassandra's basic infrastructure component.

***datacenter***

A collection of nodes associated with it. A datacenter can be a virtual or physical datacenter. Data can be stored on several datacenters depending on the replication factor.

***Commit log***

For robustness, commit log is initially updated with the data. It can be restored, removed, or reprocessed at a later stage after its data gets propagated to SSTables.

***Cluster***

There is one or more datacenters in a cluster. It can span physical sites.

***SSTable***

Also known as Sorting string tables (SSTable) are unchanging data repositories that Cassandra writes on a scheduled basis to memtables. For each Cassandra table, SSTables are appended only and stored sequentially on the disk and maintained.

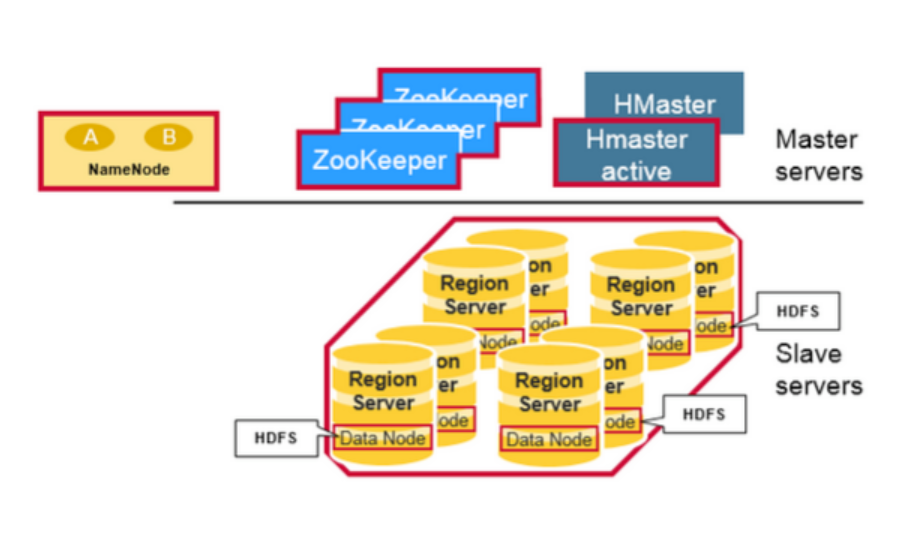
***CQL Table***

A series of columns that are retrieved using table row. A table has a structure where columns are mapped using a primary key.

## **HBase**

Physically, in a master slave type of architecture, HBase consists of three server types. Regional servers serve read and write data. Customers communicate directly with HBase RegionServers when data has to be used. The HBase Master process handles regional assignment, DDL (create, delete tables) operations. Zookeeper, part of HDFS, maintains the state of a live cluster.

The Hadoop DataNode stores the data being managed by the Region Server. All data from HBase are stored in HDFS files. Region Servers are collocated with HDFS DataNodes that allow data locality (setting data near where it is needed) for data served by RegionServers. When it is written, HBase data is local, but when a region is moved, compaction is not local.



***Namenode***

The NameNode maintains metadata information for all the physical data blocks that comprise the files.

***Regions***

HBase Tables are horizontally divided into ‘Regions’ by row key range. A region consists of all rows throughout the table between the start and end key of the region. All the nodes in the cluster are assigned to particular regions which are called region servers which serve the read and write operations on the data.

***HBase HMaster***

The operations of DDL (delete and create data) operations and region assignment are taken care of by HBase Master. The responsibilities of Master are as follows:

* Coordination of the region servers:
* Assignment of the available parts on initiation, and after that re-assignment of the regions for assigning the load with efficacy or recovery.
* Monitoring of all cluster RegionServer instances (listening for zookeeper notifications)
* Admin functions
* Interface for deletion, creation and updation of the tables.

***ZooKeeper: The Coordinator***

Zookeeper is used to uphold HBase’s server side state in the available cluster as a spreaded well-coordination service. Zookeeper keeps the servers responsive and broadcast message on server failure. Zookeeper uses consensus to ensure a shared common state. Note that for consensus, there should be three or five machines.

# Scalability, Availability & Reliability of Cassandra & HBase

**Cassandra:**

***Scalability:*** Cassandra can be scaled linearly. It means that by addition and removal of nodes, the capacity or scalability of the database can be changed. Cassandra can scale either horizontally (more data centers added) or vertically (more nodes added).

***Availability:*** Cassandra is supporting a model of "multiple master." Even minor loss in node does not impacts the cluster's capacity to commit writes, so you can get uptime for writes at 100 percent. By allowing multiple seed nodes in a cluster, Cassandra guarantees high availability.

***Reliability:*** Because of Cassandra’s high write performance, it is reliable.

**HBase:**

***Scalability:*** A Region is called the basic unit of HBase horizontal scalability. Regions can be explained as a subset of data and are generally a continous, arranged range of rows that are warehoused together. There are two main services in the HBase architecture: HMaster, which is responsible for coordinating the cluster and carrying out administrative operations, and HRegionServer, which is responsible for handling a table's data subset.

***Availability:*** In a standard configuration, across all the aspects of HBase are usually significantly available. Typically, a cluster comprises of a Master server and three or more RegionServers that have all the data kept in HDFS. Configure one or more than one Masters backups to ensure that each of the components is highly available. The Masters backup is a host instead of the active one.

***Reliability:*** HBase assures a high degree of reliability. HBase is sometimes considered tolerant of fault when configured with the correct redundancy, which means that any failure can be tolerated by HBase and still it will function properly. HBase is a distributed system, when it comes to fault tolerance. Distributed systems when compared with the high-availability configuration of the High-end Scalable data base server, shows that the failure modes are seemingly different.

# Security in Cassandra & HBase

***Cassandra***

Primarily, there are 3 components of the security features introduced by Cassandra:

* Authorization
* Client authentication
* TLS/SSL encryption based on client & inter-node communication

These features are disabled by default as other members of a cluster are configured to find and find Cassandra easily. In other words, for a bad actor, an out - of-the-box installation by Cassandra presents a large attack surface. It is not enough to allow clients to authenticate using the binary protocol to protect a cluster.

These vectors should be negated by the correct configuration of all the 3 security components. Understanding the security features in Cassandra is therefore crucial for setting up your cluster and meet your security needs.

Authentication process in Cassandra is totally roles based which is stored in the Cassandra system tables internally. With the help of an associated password, the administrators can alter, create, list roles or drop using the CQL commands. For super-user and non-superuser and the accessible login rights, roles can be formed. For accessing Cassandra's key-spaces and tables, the internal verification is to be taken, while CQLSH and the devcenter validates the contacts to SSTableloader and Cassandra clusters for loading SSTables.

Cassandra provides following security features for the open source community:

* Verification based on role-name or passwords checked internally
* Authorization based on the management of object permissions
* JMX username / password authentication and authorization
* SSL encryption
* General security measures

***HBase***

HBase should authenticate itself in the HDFS services to operate HBase on a secured HDFS cluster. HBase behaves as the principal of Kerberos and needs credentials from Kerberos to interact with HDFS daemons enabled by Kerberos. It is possible to authenticate a service using a keytab file.

RBAC has a control over the number of users or available groups which can read and also write over a certain HBase source or on the other hand execute a co-processor endpoint using familiar role paradigm. HBase is not having the functionality to keep a track of a mapping of a isolated group, but instead depend on a Hadoop group mapper which maps amongst entities of the directory for example - LDAP or the Active Directory and the active users of the HBase. All the present group mapper which are supported by Hadoop usually work.

All the available Users are then given specific approvals (Admin, Read, Write, Create, Execute) against the resources (global, namespaces, tables, cells, or endpoints). The Visibility labels limit the read and write processes to subsets of the data and allows to tag cells and offer control access to the aforementioned labelled cells.

Transparent data encryption which is underlying the file-system of the HBase, both in the WAL and the HFiles. This defends the remaining data from an attacker who has access to the underlying file system without changing the client's implementation.

Following are the types of security features in HBase:

**Authentication**

* Authentication of Client
* Authentication of Server
* Store Credential (needed for login)

**Role based security**

* Security for each role
* Options of Security role
* Scope (roles)

**Database security and logging**

* Database encryption
* Logging

# Literature surveys

The database communities and the file systems have performed research work and studies on the performance, durability and the availability for the data distribution.

Storage and computational requirements of the applications like BI and social networking because of the datasets ranging in petabyte, have pushed centralized databases like SQL to their limits [1].This resulted in the development of a horizontally scalable and non-relational distributed data stores which are also called NoSQL databases e.g., Facebook's Cassandra, Google's Bigtable and HBase.

The distributed file systems have supported hierarchical namespaces as often compared to the P2P storage systems which supports flat namespaces only. For the high Availability at the cost of consistency, replication of files is done by systems present like Coda [2] and Ficus [3].

We have another distributed file system which is known as Google File System for Google’s internal application. The Architecture of this file system is single master for storing the metadata and we have chunk servers for hosting the original data which is split into many chunks. Conflicts with updates are typically managed with the help of specialized procedures for conflict resolution. Farsite [4] which is a file system that is logically centralized but it doesn't actually use any centralized server. With the help of replication, Farsite is able to achieve high scalability and availability.

Other distributed file system like Google File System (GFS) [5] which is built specially for hosting internal applications for Google's state. The GFS has been made error tolerant with the help of Chubby [6] abstraction.

As per the article, Bayou [7] is another RDBMS data which is distributed and it facilitates eventual data consistency and allows disconnected operations. Dynamo [8] allows write and read operations which are continued and solve update conflicts by various resolution mechanisms. Amazon uses Dynamo as a storage system for their users shipping carts. Dynamo following algorithm which is Dynamo Gossip which helps each node to keep information of all others nodes.

Now a days each systems have to handle large amount of write output, this can be very limited. Bigtable [9] offers both structure and unstructured of data, but depend on for its durability on a distributed file system.

Other approaches to quantify the impact of performance using transit encryption data focus on various protocols as well as application domains like web servers, [10], [11] or web services, e.g., [12]. In turn, most of the NoSQL databases do not take security as a concern and focus on performance of the operations e.g., [13], [14], novel cloud-specific benchmarks, e.g., [15], or consistency, e.g., [16], [17]. Lastly, approaches like [18], [19] can balance our way to give insight into the cloud infrastructure.

Being a representative database of NoSQL, HBase is warehoused with the key value sets and because of HBase's high availability and scalability, it turns out to be a good scheme. There are a lot of research to examine performance and change in key-value stores [20], however others are trying to find resolutions to progress performance or improve I / O.

As per the studies done by Wang et al., n.d. [21] the operations rate are not even slightly increased by the higher replica in the Hbase and at a few instances, Cassandra impacts the read operation negatively.

All of such studies analyse HBase on various angles, primarily to improve the current HBase's Input and Output performances. However, their main research is built on the condition where the HBase RegionServer with the HDFS datanode server is deployed to gather. On Contrary, however the focus seems to be unlike from them, they give a common application situation in companies, that is, OSD server pools are owned by many HBase users, hence the RegionServers mostly are deployed from the OSD servers on different machines.

# Performance test plan

1. **YCSB platform:**

To evaluate the performance of the NoSQL databases (Cassandra & HBase), we have used Yahoo! Cloud Service Benchmark (YCSB) version 0.15.0. The performance has been tested using different workloads

* Workload A: Update heavy workload (50% Read,50% Update)
* Workload D: Read Latest Workload (Insert Records)

1. **Test Harness:**

A Test-Harness comprises of a test script repository and test execution engine. Using test harness we can achieve automation of tests. A Test Harness can call functions when supplied with parameters and then compare the results.

1. **Openstack:**

Openstack is an open-source software platform which is used for cloud computing, and is deployed mostly as infrastructure-as-a-service (IaaS), wherever virtual servers are provided to the customers. I have created an instance on NCI’s cloud present on openstack. Following are the details:

* Instance Name: x18135340
* Flavour: m1.medium
* RAM: 4GB
* Disk: 40GB
* IP Address: 192.168.101.7

1. **Virtual Machine:**

Using the proxy SSH command in Ubuntu, I have connected to the instance created above.

1. **Operation Counts:**

For testing the performance of the databases, we have taken 5 operation counts:

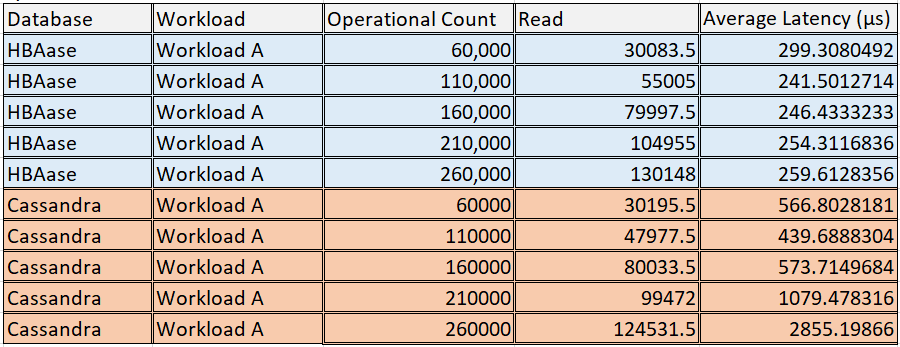
* 60,000
* 110,000
* 160,000
* 210,000
* 260,000

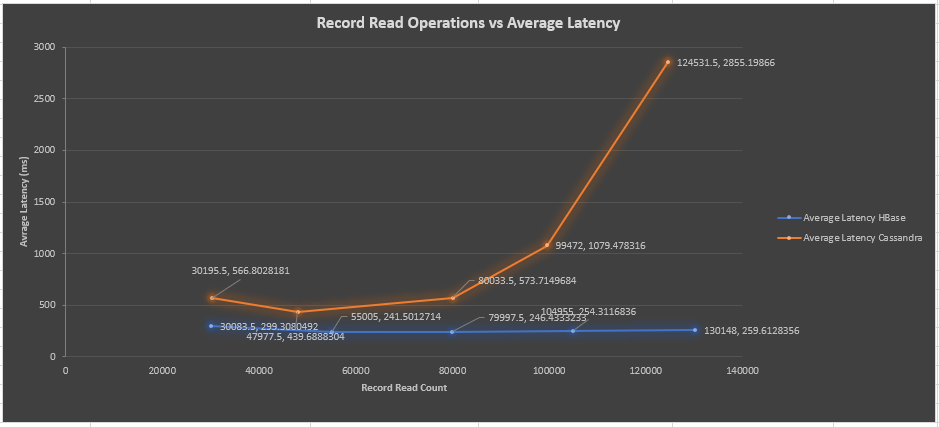
The tests for the above mentioned counts were performed two times and the average of both the outputs were taken for analysis since the output for each test run was different.

# Evaluation & results

The performance of both the databases – HBase and Cassandra were analysed with different operational counts (5 in each case). The results of all the tests are given in the below screenshots:

**Record Read operations vs average latency for workload A.**

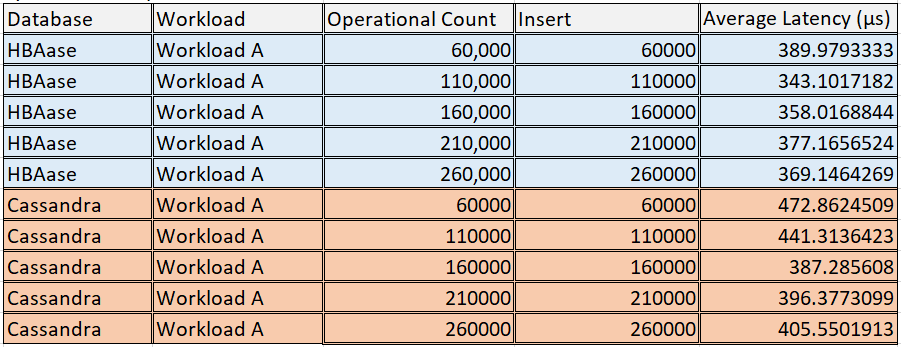


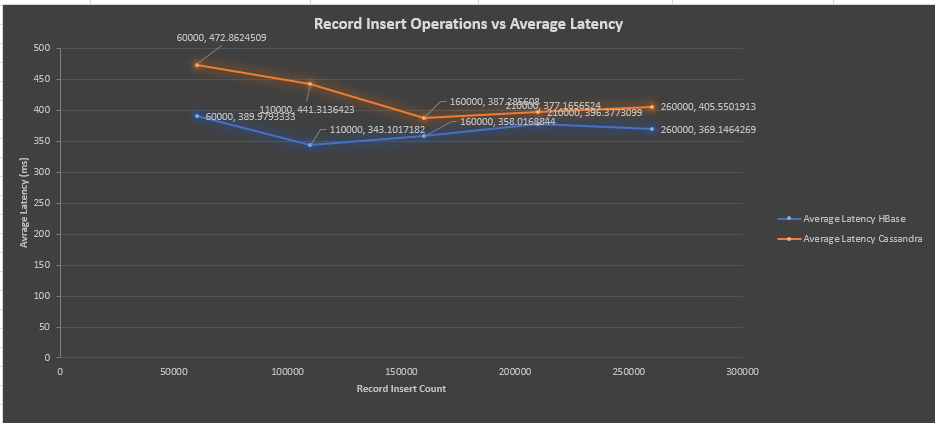


In the above graph, the Y-axis displays the Average latency and the X-axis displays the record read count. The graph shows the average latency for HBase remains consistent for all the values of operational counts whereas for Cassandra the average latency increases gradually with the increase in operational counts.

The graph above is evident that the performance of HBase is better than of Cassandra’s as in the case of Cassandra, the Average Latency goes up drastically at higher Operational Counts.

**Record Insert operations vs average latency for workload A.**

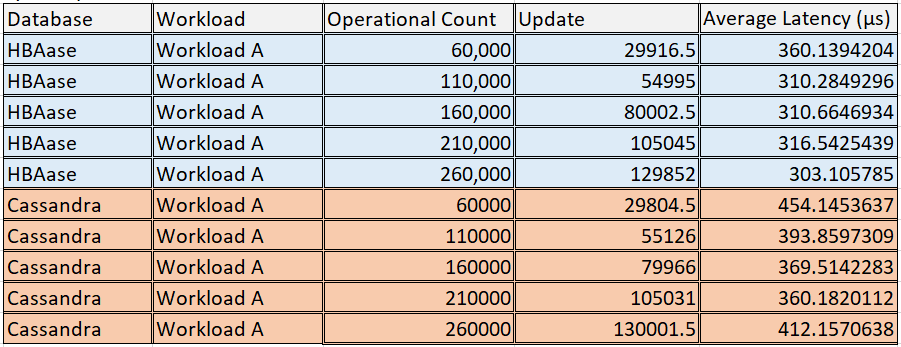


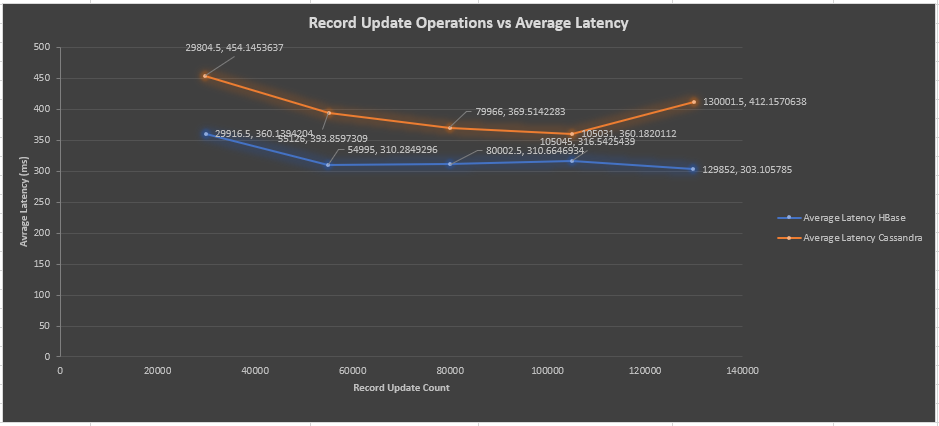


In the above graph, the Y-axis displays the Average latency and the X-axis displays the record Insert count. It can be seen from the graph that the average latency for HBase drops down at 110,000 operational count first, but then it again increases whereas for Cassandra the average latency decreases gradually till three operational counts and then increases in the last two operational counts.

The graph above is evident that the performance of HBase is better than of Cassandra’s as in the case of Cassandra.

**Record Update operations vs average latency for workload A.**

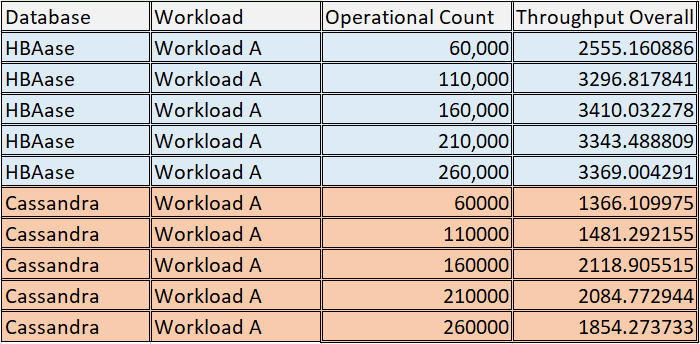


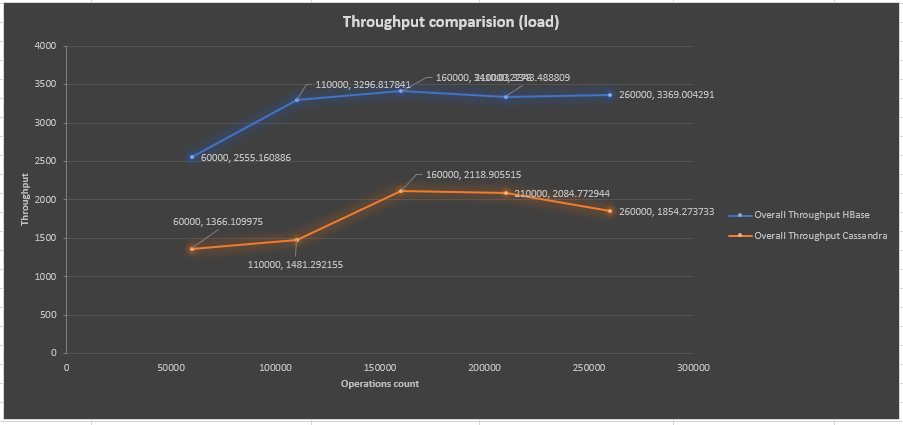


In the above graph, the Y-axis displays the Average latency and the X-axis displays the record Update count. It can be seen from the graph that the average latency for HBase drops down at 110,000 operational count first, but it becomes almost consistent with a slight drop in the end; whereas, for Cassandra the average latency decreases gradually till 210,000 operational counts and then increases in the last operational count.

The graph above is evident that the performance of HBase is better than of Cassandra’s as in the case of Cassandra.

**Throughput comparison for workload A.**

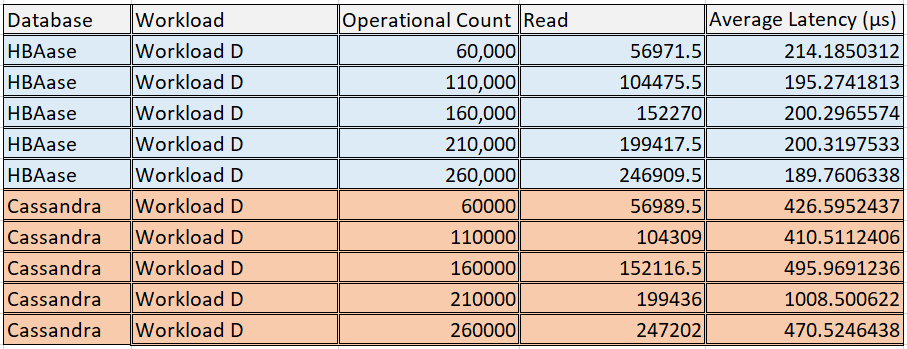


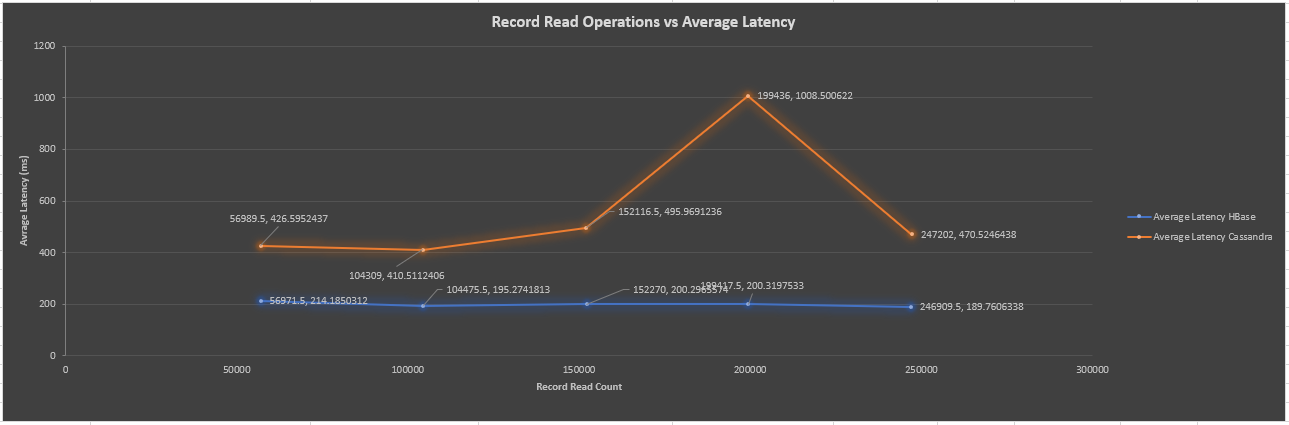


In the above graph, the readings for 5 operational counts have been taken. The X-axis displays the Throughput and the X-axis displays the Operations count. It can be seen from the graph that for both HBase and Cassandra, the throughput increases at first and then becomes almost consistent.

The graph above is evident that the performance of HBase is better than of Cassandra’s as in the case of Cassandra because the throughput values are much higher for HBase.

**Record read operations vs average latency for workload D.**

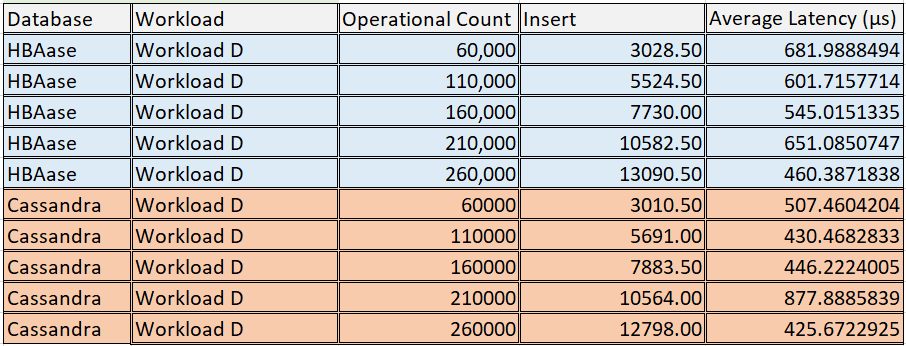


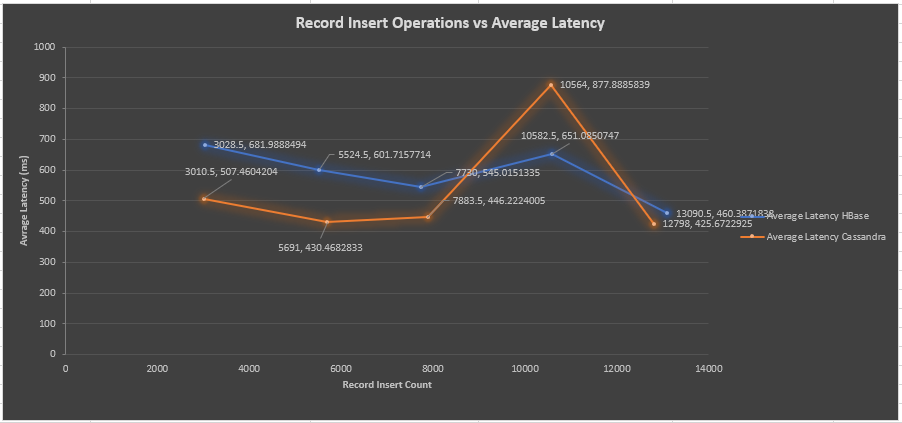


In the above graph, the readings for 5 operational counts have been taken. The X-axis displays the Average latency and the Y-axis displays the record Read count. It can be seen from the graph that the average latency for HBase drops down at 110,000 operational count first, but it increases till 210,000 operational count and then sees a sudden drop at 260,000. In the case of Cassandra, the average latency remains almost consistent throughout.

The graph above is evident that the performance of HBase is better than of Cassandra’s as in the case of Cassandra because the HBase the Average Latency is consistent and much lesser than for Cassandra.

**Record Insert operations vs average latency for workload D.**

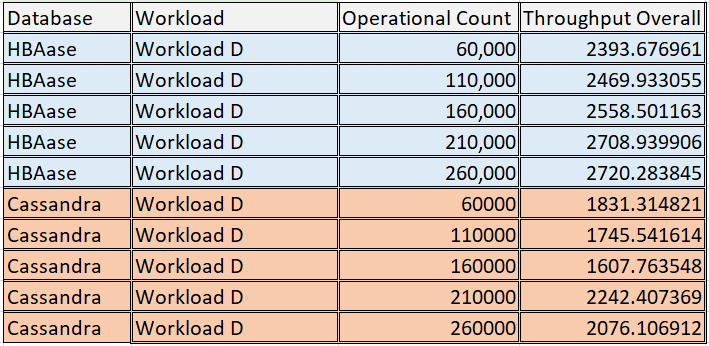


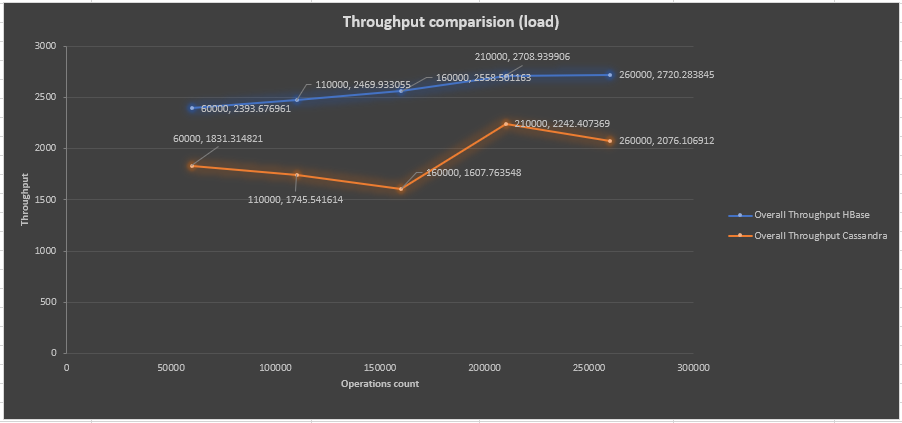


In the above graph, the readings for 5 operational counts have been taken. The Y-axis displays the Average latency and the X-axis displays the record Insert count. It can be seen from the graph that the average latency for HBase drops down till 160,000 operational count first, then experiences a sudden increase and then drops again at 260,000 operational count. In the case of Cassandra, the average latency first decreases and then rises suddenly at 210,000 count and then again drops at 260,000 count.

In this case, the performance of Cassandra is better than of HBase’s since the Average Latency is higher for Insert in HBase.

**Throughput comparison (load) for workload D.**

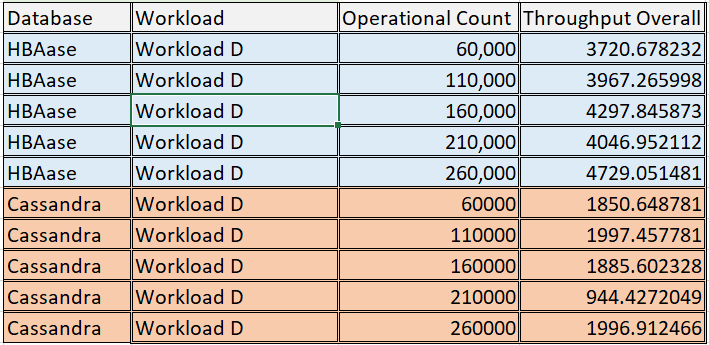


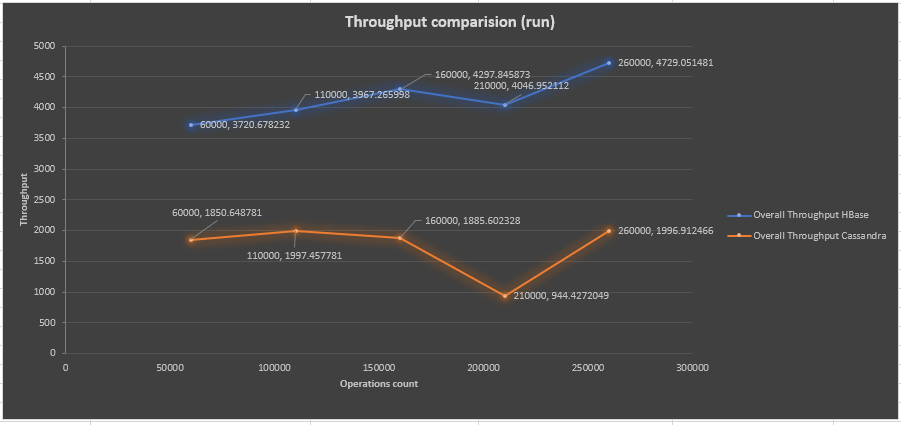


In the above graph, the readings for 5 operational counts have been taken. The X-axis displays the Operations count and the Y-axis displays the throughput. It can be seen from the graph that the throughput for HBase has been consistently increasing throughout. In the case of Cassandra, the throughput first decreases till 160,000 and then rises suddenly at 210,000 count and then again drops at 260,000 count.

The graph above is evident that the performance of HBase is better than of Cassandra’s as in the case of Cassandra because the throughput values are much higher for HBase.

**Throughput comparison (run) for workload D.**





In the above graph, the readings for 5 operational counts have been taken. The X-axis displays the Operations count and the Y-axis displays the throughput. It can be seen from the graph that the throughput first increases till 160,000 and then drops and rises again. In the case of Cassandra, similar pattern can be observed, throughput first increases till 160,000 and then drops and rises again.

The graph above is evident that the performance of HBase is better than of Cassandra’s as in the case of Cassandra because the throughput values are much higher for HBase.

# Conclusion & Discussion

In this project, an Analysis and comparison of the results on the tests performed for both the databases – HBase and Cassandra, on different workloads (workload A and workload D in this case), has been done. Both HBase and Cassandra are highly preferred by most of the reputed organizations around the globe. As per the results of the tests done for two workloads A & D, performed on YCSB platform, HBase has performed better in most of the cases. In an article by *Muthukkaruppan, 2010* [22], HBase has a very good scalability at run time and also the performance with a much simpler consistency model when compared to Cassandra. HBase has displayed lower average latency when compared to Cassandra for most of the cases. Cassandra may have performed better if the tests would have been carried out related to availability or scalability. This can be some work dedicated towards future work.

On analysing the test results on graph, it is evident that overall throughput for HBase is much better than that of Cassandra. However, in the real-world applications, Cassandra performs better than HBase and is also preferred more because of the high volume of the data that has to be managed and it’s better write performance. HBase on the other hand is good at high read performance. The only drawback of Cassandra is consistency of data, while for HBase, the availability of data is the issue, although both try to lower down the negative effects of these problems. The graphical representations of the average latency and throughput shows that there is a little higher level of fluctuations when the operational counts changes. By knowing the interests and the requirements, a company can choose to perform the tests on YCSB benchmarking platform.

# References

1. J. Abadi. Data Management in the Cloud: Limitations and Opportunities. IEEE Data Eng. Bull, 32(1):3–12, 2009.
2. Peter Reiher, John Heidemann, David Ratner, Greg Skinner, and Gerald Popek. Resolving file conflicts in the ficus file system. In USTC’94: Proceedings of the USENIX Summer 1994 Technical Conference on USENIX Summer 1994 Technical Conference, pages 12–12, Berkeley, CA, USA, 1994. USENIX Association
3. Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere. Coda: A highly available file system for a distributed workstation environment. IEEE Trans. Comput., 39(4):447–459, 1990.
4. Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. In In Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI, pages 1–14, 2002.
5. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In SOSP ’03:Proceedings of the nineteenth ACM symposium on Operating systems principles, pages 29–43, New York, NY, USA, 2003. ACM.
6. Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In OSDI ’06:Proceedings of the 7th symposium on Operating systems design and implementation, pages 335–350, Berkeley, CA, USA, 2006. USENIX Association.
7. D. B. Terry, M. M. Theimer, Karin Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In SOSP ’95: Proceedings of the fifteenth ACM symposium on Operating systems principles, pages 172–182, New York, NY, USA, 1995. ACM
8. Giuseppe de Candia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazonOs highly available ˜ key-value store. In Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, pages 205–220. ACM, 2007.
9. Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. In In Proceedings of the 7th Conference on USENIX Symposium on Operating Systems Design and Implementation - Volume 7, pages 205–218, 2006.
10. G. Apostolopoulos, V. Peris, and D. Saha, “Transport layer security: how much does it really cost?” in Proc. of INFOCOM 1999, 1999, pp. 717–725.
11. L. Zhao, R. Iyer, S. Makineni, and L. Bhuyan, “Anatomy and performance of ssl processing,” in Proc. of ISPASS 2005, march 2005, pp. 197–206.
12. S. Shirasuna, A. Slominski, L. Fang, and D. Gannon, “Performance comparison of security mechanisms for grid services,” in Proc. of GRID 2004, 2004, pp. 360–364.
13. B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, “Benchmarking Cloud Serving Systems with YCSB,” in Proc. of SOCC. ACM, 2010, pp. 143–154.]
14. [M. Klems, D. Bermbach, and R. Weinert, “A Runtime Quality Measurement Framework for Cloud Database Service Systems,” in Proc. of QUATIC, 2012, pp. 38–46.
15. C. Binnig, D. Kossmann, T. Kraska, and S. Loesing, “How is the Weather Tomorrow?: Towards a Benchmark for the Cloud,” in Proc. of DBTEST. ACM, 2009, pp. 1–6.
16. D. Bermbach and S. Tai, “Benchmarking Eventual Consistency: Lessons Learned from Long-Term Experimental Studies,” in Proc. of IC2E. IEEE, 2014, pp. 47–56.]
17. [“Eventual Consistency: How Soon is Eventual? An Evaluation of Amazon S3’s Consistency Behavior,” in Proc. of MW4SOC. ACM, 2011, pp. 1–6.]
18. [Kuhlenkamp, K. Rudolph, and D. Bermbach, “AISLE: Assessment of Provisioned Service Levels in Public IaaS- based Database Systems,” in Proc. of ICSOC. Springer, 2015, pp. 154–168.]
19. A. H. Borhani, P. Leitner, B.-S. Lee, X. Li, and T. Hung, “WPress: An Application-Driven Performance Benchmark for Cloud-Based Virtual Machines,” in Proc. of EDOC. IEEE, 2014, pp. 101–109.
20. GANDINI, A., GRIBAUDO, M., KNOTTENBELT, W. J., OSMAN, R., AND PIAZZOLLA, P. Performance evaluation of nosqldatabases. In European Workshop on Performance Engineering (2014), Springer, pp. 16–29.|AHAMED, M. A. U. R. benchmarking top nosql databases.| PATIL, S., POLTE, M., REN, K., TANTISIRIROJ, W., XIAO, L., LOPEZ ́ , J., GIBSON, G., FUCHS, A., AND RINALDI, B. Ycsb++: benchmarking and performance debugging advanced features in scalable table stores. In Proceedings of the 2nd ACM Symposium on Cloud Computing (2011), ACM, p. 9.
21. Wang, H., Li, J., Zhang, H. and Zhou, Y. (n.d.). Benchmarking Replication and Consistency Strategies in Cloud Serving Databases: HBase and Cassandra. (Wang et al., n.d.)
22. Muthukkaruppan, K. (2010). The Underlying Technology of Messages | Facebook. [online] Facebook.com. Available at: https://www.facebook.com/notes/facebook-engineering/the-underlying-technology-of-messages/454991608919/ [Accessed 7 May 2019].