HBase or Cassandra? A Comparative study of NoSQL Database Performance

Prashanth Jakkula*

* National College of Ireland

DOI: 10.29322/IJSRP.10.03.2020.p9999
http://dx.doi.org/10.29322/IJSRP.10.03.2020.p9999

Abstract- A significant growth in data has been observed with the growth in technology and population. This data is non-relational and unstructured and often referred to as NoSQL data. It is growing in complexity for the traditional database management systems to manage such vast databases. Present day cloud services are offering numerous NoSQL databases to manage such non-relational databases ad- dressing different user specific requirements such as performance, availability, security etc. Hence there is a need to evaluate and find the behavior of different NoSQL databases in virtual environments. This study aims to evaluate two popular NoSQL databases and in support to the study, a benchmarking tool is used to compare the performance difference between HBase and Cassandra on a virtual instance deployed on OpenStack.

Index Terms- NoSQL Databases, Performance Analysis, Cassandra, HBase, YCSB

I. INTRODUCTION

Data is growing in complexity with the rise in data. Large amount of data is being generated every day from different sources and corners of the internet. This exponential data growth is represented by big data. It is serving different use cases in the present day data driven environment and there is a need to manage it with respect to velocity, volume and variety. The traditional way of managing the databases using relational database management systems could not handle because of the volume and they are capable of storing the data which is schema based and only in certain predefined formats. Big Data paradigm is gradually changing the present data storing techniques, processing, administration and the methods of analysis [1]. This lead to new developments in design and architecture of database management systems to handle the big data. As the data also is non-relational it is often known as NoSQL data. The NoSQL data do not have a fixed structure or schema and it is enormous in volume. They allow storing of multiple forms of data which is structured, unstructured or even semi-structured [1]. They store data in the form of column families, key value data stores, document data stores etc. Hence NoSQL database management systems are designed to replace the traditional SQL DBMS. Since these databases are non-relational, the query language support is subjective.

Different types of NoSQL data bases are being used in the present day applications as they are not dependent entirely on queries for data management. These databases are designed to provide flexible storage requirements. These databases are extensively used in environments where data do not rely on a relational model. There are different NoSQL databases, categorized depending on the type of data store. They are categorized into document based, key value based, column based etc. Each type of database serves user specific data storage requirements. Two such databases are Apache HBase and Apache Cassandra. Both the databases are NoSQL databases and are popularly used for present day non-relational database management. With the rise in cloud technologies, virtualization has become one of the widely adapted technologies. Open source offerings such as OpenStack are providing different platforms to execute the workloads on virtual machines. Though the virtual environments are scalable and highly performing, there are certain challenges when it comes to the latency and bandwidth allocation. As a solution, offerings such as Amazons CloudFront are providing edge locations to replicate and store the data to the closest possible availability zones. However, the performance of the databases in virtual environments and clouds has remained a question to the research community. The types of NoSQL databases are given in figure 1.

This paper aims to evaluate the performance of NoSQL Databases, HBase and Cassandra that are deployed over a single virtual machine in OpenStack. The later sections of the paper, gives an understanding of the key characteristics and the architectures of both the databases and the differences. As a part of the study, the later sections of the paper also covers the performance evaluation techniques implemented in previous research concluding with the evaluation and the results.

Type of Database	Example
Key-Value Store	Azure Table Storage, Amazon DynamoDB, Redis, Riak, Berkely DB, etc.
Column-Family Database	HBase, Cassandra, Hypertable, Amazon SimpleDB, etc.
Document-Oriented Database	MongoDB, RavenDB, CouchDB, OrientDB, etc.
Graph Databases	Neo4J, Infinite Graph, etc.

Figure 1 Examples of NoSQL Databases

Identify the constructs of a Journal – Essentially a journal consists of five major sections. The number of pages may vary depending upon the topic of research work but generally comprises up to 5 to 7 pages. These are:

1) Abstract
2) Introduction
3) Research Elaborations
4) Results or Finding
5) Conclusions

In Introduction you can mention the introduction about your research.

II. Literature Review

There has been a vivid research that was carried out in the field of performance evaluation of database management systems. Several techniques and methodologies were proposed to benchmark the performance of NoSQL databases. [11] has emphasized on the frameworks that are capable of performance evaluation of databases. They proposed a framework that can monitor, analyze and predict the behavior of a database. This architecture helped in forming the challenges that are faced in evaluating NoSQL databases. And it was observed that the behavior of the evaluation framework depends on the database characterization and the testing system. However, their model is inclined toward machine learning and prediction of the database behavioral patterns. [12] gave insights for evaluation of in memory databases. The study emphasized on the available variations of NoSQL databases and the need to determine the best performing databases management system. Their study drew evaluations between the MongoDB, Memcached, Redis and Cassandra. A software based on Java has been used to draw the evaluations over metrics such as the execution time per operation. And their tool was based on the studies conducted by [13]. And [5] emphasized on the advantages of HBase over other NoSQL databases, similarities and the differences between HBase and Googles BigTable.

[14] suggests that NoSQL databases are generally characterized by the properties such as no-schema data models, horizontal scalability, and simple cloud deployment. The study also suggests that there is a need to identify the correct system requirements before deployment to avoid overprovisioning. The benchmarking was done between MongoDB, HBase and Cassandra databases deployed on Amazon EC2. YCSB was used as a benchmarking tool. They have testes each of the database with specific workload deployed on different virtual instances offered by Amazon EC2. The proposed modelling approach suggested complex modelling of replication to accurately depict the performance of a replica. [15] has proposed an approach to benchmark the similar databases such as the column family databases. Brian Cooper emphasized on two tier benchmarking in which one focuses on the performance of the database while the other focuses on the impact on performance due to the scalable feature of database. Their benchmarking system measures the metrics such as the inserts, updates, reads and scans. And they have defined certain workloads to choose from depending on the targeting metric.

[16] compared the performance and the working of SQL databases and NoSQL databases. The comparison is done on a specific dataset. Their study suggested the implementation of transactions in the both types of databases. The transactions were tested over storing the digital media with respect to social media platforms and simulated the social network environments to test the workloads. Their experiment results suggested that NoSQL databases surpass the SQL operations when it comes to the transactions for storing the digital media. Similar evaluations were conducted by [17] between MySQL, Cassandra and Hbase on the write heavy operations. Their experimental implementation was executed with the help of a web-based REST application. The study also emphasized on the CAP properties of the databases and suggests the trade-offs between each database management system. They made use of a java application that in executed with the help of Representational State Transfer. It puts the data in the database using HTTP POST requests. The standard metric for the throughput selected is transactions per second (TPS) and the application was hosted on a Tomcat server. Their test results suggest that HBase has write speeds twice as fast as MySQL database which is a relational database. It is also observed that Cassandra gives significantly fast writes even in a write heavy application.

In summary, [5] gave insights on the differences between the conventional relational database management systems and the Non-relational DBMS. While [11], [12], [13] suggested other evaluation techniques available to evaluate the performance of different types of databases. The implementation that was carried out in the work done by [14] suggested a stable methodology to evaluate the performance of NoSQL databases over the cloud instances.

It's the foremost preliminary step for proceeding with any research work writing. While doing this go through a complete thought process of your Journal subject and research for its viability by following means:

1) Read already published work in the same field.
2) Goggling on the topic of your research work.
3) Attend conferences, workshops and symposiums on the same fields or on related counterparts.
4) Understand the scientific terms and jargon related to your research work.

III. Key Characteristics

There are various NoSQL databases available to store different forms of data. Apache HBase and Apache Cassandra were selected for this performance evaluation. These databases offer wide range of functionalities starting from the type of data store. And each database offers different features to manage the data. The key features that differentiate HBase and Cassandra are in the following sections.

A. HBase

Apache HBase is a part of Apache Hadoop. It offers a scalable and distributed big data store in Hadoop. It can be used to achieve real time and random read/write access to the data. It helps in storing very large tables. As it is a column family database, it is capable of storing tables with billions rows X million columns. And this can be deployed in commodity hardware. Similar to BigTable by Chang et al., HBase is a distributed, Open Source non-relational database model. It provides the capabilities of a BigTable over Hadoop file system similar to BigTable for Google File System. The following are the key features offered by HBase;

- HBase provides linear scalability and modularity to the database
- It offers consistent read/write operations
- It offers automatic sharding of tables and can also be configured as per user requirements
- In case of region servers, HBase supports automatic failover
- Hadoop MapReduce tasks are supported by Convenient base classes with Apache HBase tables
- Java API can be used with less complexity for client access
- Real-time queries can be implemented with the help of Bloom Filters and Block caches
- Server side Filters can be used in HBase for Query predicate push down
- HBase offers REST-ful service and Thrift gateways that supports Protobuf, XML, and binary data encoding
- Jruby based shell (JRIB) is included with HBase
- Exporting metrics to files or JMX or Ganglia is supported by metric subsystem of Hadoop

Since HBase is a NoSQL database it does not support SQL. However, SQL support is under development which can be used with the help of Hive. And as it uses MapReduce, requests with low latency cannot be implemented [2]. HBase when compared with a traditional RDBMS, it does not work as a column family database but makes use of the storage format on disk [3]. This is one of the factors that differentiate HBase from the traditional RDBMS. Real time access to analytical data can be seen in the traditional column based databases whereas Hbase provides support to key based access to a single cell of the data or a specific range of cells.

B. Cassandra

Cassandra is a scalable and highly available NoSQL database. It is an open source column store designed to accommodate enormous amount of data over commodity hardware. It is highly available without a single point of failure. This database is also a widely known Column family database and it also works as a Key value store. Similar to HBase it also shares features with Big table [4]. Cassandra is licensed by Apache and is designed to manage large structured datasets. One of the main features of Cassandra is that it allows low latency operations with the help of asynchronous replication without a master node. This can be achieved over the clusters if multiple datacenters. Other than this, Cassandra offers a lot of features which make it stand out. Some of the key characteristics of Cassandra are as follows;

- Cassandra is highly scalable and it can be scaled linearly and elastically
- Increase in the size of a cluster can contribute to the performance of Cassandra database
- It offers continuous availability to the operation critical environments as it has no single point of failure
- Cassandra is highly tolerant
- It allows easy distribution of data by data replication over multiple datacenters

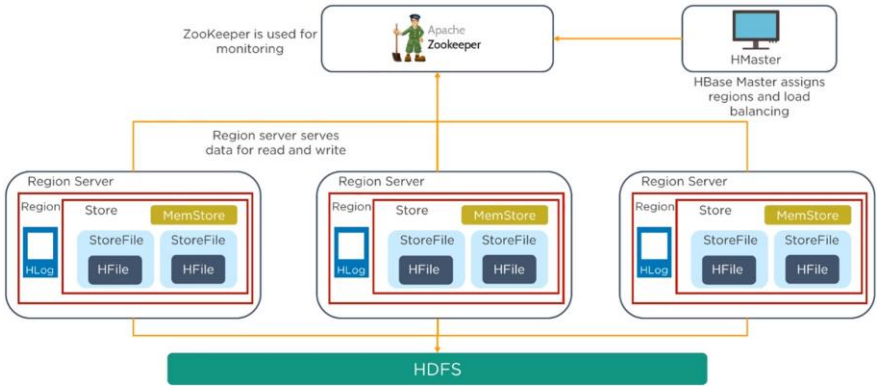

Figure 2 HBase Architectural Components

- It also supports ACID properties which ensures the quality of service and supports high speed writes
- It supports all kinds of data such as unstructured, structured and semi-structured data and also allows user specific changes to the data structures

When compared to a relational database management system, Cassandra is highly dissimilar due to its data model. Efficient storage, associations between concerns, relational look ups are few of the best known characteristics of relational data models whereas the data model for Cassandra is built for large amounts of data storage and performance. Similar to HBase, the data model for Cassandra is query dependent.

IV. NoSQL Database Architectures: Apache HBase

As mentioned in the previous sections, HBase is a Database designed for Hadoops Distributed File System and it is built on the Map Reduce framework. The difference between HBase and Hadoop HDFS is that HDFS is a file system for storing large datasets or files and unlike a normal file system, it fails to provide instant record lookups and updates [5]. But HBase stores the data in the form of indexed storefiles that are stored in HDFS. This allows to achieve fast lookups of the files. The architecture of HBase has a master node and several slave nodes. Figure 2 Explains the architectural components of HBase. It consists of Master and slave nodes. A Single master node is present in the HBase architecture which assigns the regions and load balancing called HMaster. The slave nodes are Known as the Region Servers. These region servers are the computers in a Hadoop cluster that serves different regions. Each region can only be handled by one region server. When a write request is sent by the client, it is received by the HBase Master and it is further sent to the specified region server. Zookeeper is used to monitor the system. Hence the three major components of HBase are;

- HMaster
- Region Server
- ZooKeeper

A. HBase HMaster

HBase H Master is the Master node or the master server in the cluster. It is responsible for the monitoring of the region servers. It also acts as an interface for any changes that goes into the metadata. In case of distributed clusters, the HMaster node of HBase runs on the NameNode of the Hadoop Map Reduce framework. The startup behaviors and the runtime impacts of the master vary depending on the multi-master setup. When a Hmaster is released by ZooKeper, the rest of the masters present in the cluster will compete for the role of HMaster. But in case of a Master Shut down, the cluster can be functional as the HBase client directly communicates with the Regional servers. However, the critical functions are controlled by Master such as failover of Region Server. And hence it is required to have another master allocated instantly. According to Jahar Mohamed, an HMaster consists of the following components:

i. ZooKeeper System Trackers

ZooKeeper is used by the HBase HMaster and the Region Server to maintain a track of events that are happening in a defined cluster. ZooKeeperWatcher is a centralized class that is defined in a Master which acts as a proxy for the event tracker targeting ZooKeeper. All node management jobs such as exception, connection handling are handled by the ZooKeeperWatcher. Registered trackers with this class can get notified of the defined event.

ii. External Interfaces

These interfaces are used to establish communication between HBase and the external entities such as HMasters, Region Servers and other utilities. It contains Info server. It is an instance of a jetty server that is initiated by HMaster to follow http requests. It also contains RPC server to maintain the protocols and a Master MXBean to view the monitoring metrics in HBase.

iii. Executor Services

Different types of events posted in an event queue can be abstracted with the help of Executor Services. Each of these events are handled by a specific event handler which are capable of picking the threads from the thread pool.

iv. Chore

Chores are the tasks that are performed regularly in HBase. These tasks are executed in their own threads. Hence this is a repeated function, this provides a while loop and sleep to pause the iterations. The basic function of a Chore in HBase is to keep checking for any unhandled work. Balancer Chore, Catalog Janitor Chore, Log Cleaner Chore and HFile Cleaner Chore are the chore tasks present in HBase.

v. File System Interfaces

The services that interact with Hadoop Distributed File System to manage and store the data with respect to the HMaster code control are categorized as File system interfaces. It contains MasterFileSystem abstraction class that abstracts the operations such as delete region, delete table etc. It also has Log Cleaner and HFile Cleaner chores which are responsible for performing cleaning tasks in the file system.

B. RegionServers

HBase tables are horizontally divided by a row and each vertical unit is the basic available unit of data distribution for tables. These units are called as regions. Each column family is assigned a data store. Regions are assigned to the nodes in the cluster called Region Servers. A region server implementation in HBase can be executed by HRegionServer. They are responsible for serving data to the regions for read and write operations. Region Servers typically run over a DataNode in a distributed cluster. Each Region Server consists of components such as, BlockCache, MemStore, Write Ahead Log, HFile [6].

C. ZooKeeper

ZooKeeper acts as a monitoring system for HBase for assigning regions to the Region Servers and to recover failed Region Servers by replacing them with others. It is a central server which is capable of maintaining the configurations and providing synchronization over distributed systems. The communication link between the client and the regions is managed by the Zoo- Keeper. The Region Servers and the HMaster are to be registered with the ZooKeeper Service. To establish a connection with such HMaster and the Region Servers, a client has to access the ZooKeeper quorum. It quorum is responsible for generating the error messages and recovery in the event of node failure.

The ZooKeeper Service is also responsible for tracking and maintaining information about the Region Servers present in HBase. It maintains information such as the No. of working region servers and the Region Server allocation to data nodes. Different services that are offered by ZooKeeper are as follows:

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

V. NoSQL Database Architecture: Cassandra

Handling Workloads in Big Data without a single point of failure is one of the main motivations behind the design and development of Apache Cassandra. In a cluster running with Cassandra, nodes are interconnected in a peer to peer fashion and the distribution of data is achieved over all the nodes of the cluster. Each node in the cluster is independent by itself and also communicates with other nodes and all the nodes are given the similar role. Regardless of the data location, these nodes in the cluster are capable of accepting read and write requests. During the node failure, the data which is replicated into another node is served in the network.

A. Cassandra Ring:

The scalability, performance and the continuous availability are the three key features of Cassandra database and its architecture contributes to it. It has a Ring type architecture in which the master node is absent, often referred to as masterless ring. This ring type architecture makes Cassandra less complex to install and maintain. Due to the scalable property of the architecture, Cassandra is capable of managing large data sets. It also allows several no. of operations that can be performed in a second over multiple datacenters. For example, when a workload is specified to a particular node, that node is not entirely responsible for the operations but the workload is spread across all the nodes in the cluster and hence all the nodes contribute to the operations and the behavior of the cluster [7].

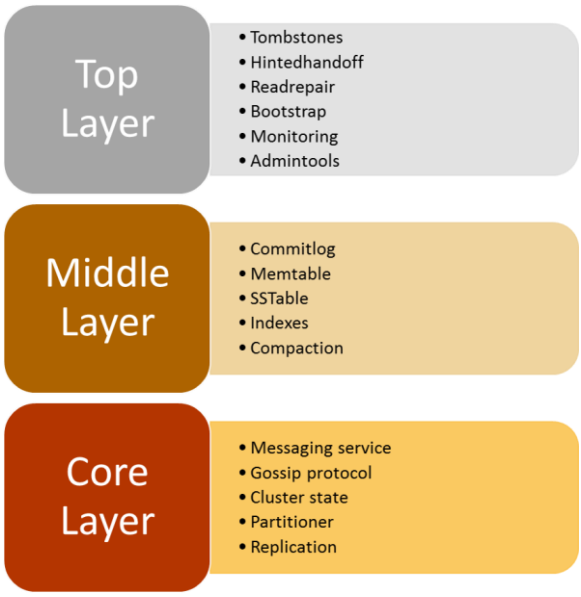

Figure 3 Three layers of Cassandra Architecture [8]

Since the Cassandra cluster implements a continuous hashing algorithm for the distribution of data it is often depicted as ring. Token ranges are defined for each node during the start- up of a cluster. This helps in determining the data range that is stored in the node and the position of a node in the cluster. Each node in the cluster is defined a specific token range to ensure the even distribution of data in the ring. A partitioner determines the set of data that has to be assigned to a node and the node takes the responsibility to manage that data in the cluster. A partitioner acts as a hash function that is capable of computing the token result for a specific row key. This resultant token determines the node to store the data replication. Cassandra includes data partitioners such as RandomPartitioner, Murmur3 and ByteOrdered Partitioners. The architecture of Cassandra mainly contains three layers classified depending on the functions. The three layers are the Core, Middle and the Top Layers. Figure 3 Explains the responsibilities of the three layers in the architecture of Cassandra. Each layer handles different tasks. The responsibilities of each layer are as follows:

i. Core Layer
All the basic operational features of Cassandra are handled by the core layer. Responsibilities such as partitioning, communication across multiple loads, the gossip protocols, messaging and interaction through Thrift or Binary protocol, Clusters, cluster behavior and the replication of data are maintained by the Core layer.

ii. Middle Layer
The middle layer of Cassandra architecture acts as the communicator between the nodes and the clusters. It is responsible for maintaining the communication and logs the communication requests. It maintains Commitlogs, Mem tables and the SStables to track the two phase commit in the database. This helps to maintain the consistency of the data. It also maintains the data indexes to know in which node the data is getting stored and the compactions to leverage the optimal utilization of the memory.

iii. Top Layer
This layer acts as the communicator between the external entities and the cluster. It provides an extraction layer responsible for communicating with the core layer services. The top layer has services such as Tombstones, Hintedhandoff, readrepair which are responsible for maintaining backups and restores of the data in a particular node. Other services include system Cluster bootstrapping, Monitoring the cluster ecosystem and admintools to give maintain security and integrity of the data on the node.

B. Gossip Protocol

Gossip protocol is one of the vital component of Cassandra cluster. It is used to know the state of all the nodes that are available in a cluster. Nodes in the cluster implement gossip protocol to exchange state information about other nodes they were notified about and themselves. This can be achieved with other three nodes in the cluster. To reduce the network load, the nodes do not send messages to all the other nodes in the cluster but can only send messages over a time period and the node information will reach all the nodes of the cluster. Gossip protocol is helpful in case of failure detection. The advantages of this protocol are that the execution is simple, it facilitates the cluster scaling up or scaling down at any point of time as there is no single entity that takes up the responsibility of all the nodes in the cluster. Every node in the cluster has mutual and equally shared responsibility. This differentiates Cassandra from conventional

Master-Slave implementation. Reliability is also one of the best features of Gossip Protocol as it can help discover any node failures in the cluster and can immediately replace the node.

C. Cluster Bootstrapping

Each cluster in Cassandra database must be identified by a name tag. All the nodes inside the cluster will have the same name. Some nodes are used to help the startup processes of a cluster and these are called as seed nodes. The main purpose of these seed nodes is to assist in bootstrapping a cluster with the help of Gossip Protocol. Seeds provide the list of active nodes in a cluster which can be used by the newly launched nodes to find the nodes in its cluster. As mentioned above each node can communicate about the state information only with other three nodes. The state information of the nodes is continuously shared and updated in the cluster which contains the information about the message sending node and the other nodes that previously sent the message in the cluster. This helps to disclose the information of all the nodes.

VI. Differences Between HBase and Cassandra

HBase and Cassandra are two different NoSQL databases licensed by Apache. Since both the databases are non-relational databases they share identical features. Similarities such as being wide-column NoSQL database stores based on BigTable are prominent. As HBase runs on top of Hadoop, it does not support query languages but it works with HBase shell which is based on JRuby and can also include features such as Hive and Drill. Cassandra on the other hand supports its own Cassandra query language. Both of these databases offer different security policies and also differ in transaction management.

A. Security

Similar to any other NoSQL database, Cassandra and HBase have their own security challenges and workarounds. The first issue with high security in these databases can be loss of performance. But both of them offer unique features to address these security issues. They ensure security of data by authentication and authorization. But, Unlike HBase Cassandra has more rigid security features such as inter-node and client to node data encryption. But HBase makes use of other technologies to secure the communication between the client and the cluster.

HBase offers cell level security features. It offers the following options;

- Authentication,
- Role-based security
- Data Security, Logging

HBase offers authentication both from server side and the client side. As it maintains the user credentials it also provides secure storage of these credentials. It uses different kinds of protocols to authenticate the traffic into the database. As HBase also works in distributed mode, the database servers can authenticate themselves with each other to secure the communication in the cluster. A credential store is provided to securely store the data about the user credentials. It is often stored in an external file. HBase also ensures the security by providing roles. It is a secure approach to authorizing the user access into the contents of database. Implementation of role-based security can simplify the operations and administration of security in the database. It allows users to create their own roles called custom roles and also provides default roles. Moreover, it is essential to define the scope of each role to ensure finer granularity especially for highly sensitive data.

Logging is also one of the security features of HBase which is helpful in maintaining the database security. HBase achieves database security by encrypting the database partitions. Logging the events also helps in ensuring the security of the environment. Security logs can record the information such as the no. of users that are logged into the cluster. HBase defines a administrator to select the events to be recorded in the log. HBase supports Configurable event logging and Fixed Event logging

B. Security: Cassandra

Cassandra provides three different security functionalities to the database. They are

- Client Authentication
- Authorization and
- Inter-node and client communication encryption with TLS/SSL

However, these features come disabled initially as Cassandra is known to be easily found by the members of cluster. Proper configuration of these given security features can help in ensuring the security of the cluster. Communication security is achieved by enabling TLS/SSL encryption for inter node communication and between client machine and database. It ensures that the data in transit is not compromised. FIPS settings can also be configured at the runtime level to not change the encryption settings in Cassandra.yaml. Cassandra also implements roles for the database which can be applied to a single user or to a user group. These roles can define permissions and authenticate the users. Role management is also facilitated by Cassandra which enables to configure the default role configurations.

Authentication and authorization are essential to maintain security in the database. The authentication is provided by an authenticator

class in Cassandra configurations. It allows two options, allow authenticator which is responsible check the authentication without any credentials required and Password Authenticator which takes the user credentials such as passwords and stores them in a table. Authorization is handled by an authorizer configuration. There are Allow authorizer which does not require any checking and grants permissions to the user roles. It also comes with default Cassandra authorizer which handles permission management and stores the user data in a system table.

In Detail, not only both the databases provide access control over the whole system but also they allow a level of flexibility. HBase offers access control to the deepest level such as the cell level. whereas, Cassandra provides access control to Row level. User roles are other security feature offered by Cassandra which sets access control to the users and other nodes and can be used later to determine the user access over a particular data set. Unlike Cassandra, HBase offers an inverse approach to user roles by providing visibility label that can be assigned by the administrator to a particular data set. This can give the user access form the administrators side.

C. Transaction Management: HBase

HBase supports ACID properties and hence it provides atomicity to any changes on the basis of Rows, even in case of changes to different column families. However, it is not assured that a transaction is consistent for changes across rows. Even though HBase uses HDFS to store the data, the write operations often go through region servers and they are responsible for diverting the traffic to multiple regions. In case of a write operation to a particular row, that is present in a region, the region server locks that row across all the other columns and does not allow any writes to that particular row. It means that it can only allow a single row transation. And further the region server records the transaction in the Write ahead log to store it on HDFS. After this transaction, put operations are performed over all the column families that are defined in the put operation. Since the writes are recorded in the Write Ahead Log, new region servers that replace the failed region servers can perform all the write operations for a region and start serving traffic.

However, HBase also offers a concurrency control system called Multiversion Concurrency Control that multiple concurrent reads and writes across multiple rows in a region ensuring the semantics of row-level transactions. Briefly, HBase tables consists Key-value pairs. The first column can be considered as a Key and the rest as the values for the defined key. As it is defined similar to a SQL record, transactions within the columns can be possible. As it is a NoSQL database, transactions between the column families are not supported. However, external support can be available from other engines such as Apache Tephra [9] which is a transaction engine. It comes with Transaction Server, transaction client and a transaction processor.

D. Transaction Management: Cassandra

Similar to HBase, Cassandra also supports ACID properties but unlike a conventional RDBMS, it does not use locking and rollbacks. It offers transactions that are atomic and isolated with user defined consistency to ensure the durability. Which means it can only support atomicity, Isolation and durability of the ACID properties. Being a non-relational database, Cassandra does not foreign key joins. It offers isolation and atomicity at row level and also provides high availability and high speed write by losing transactional atomicity and isolation. It treats an operation of write over multiple rows of a partition as a single operation, an atomic operation and it is similar for a delete operation. Client side timestamps are used by the database to define the latest transactions to a column. On a data request, the latest timestamp is viewed first as it is the last updated record in the database.

Other operations such as writes and deletes are fully isolated at the row level. A write to a row in a column or a single partition over a node can only be accessed by the user that is performing the operation. This operation is isolated from the other nodes and the users until the transaction is deemed complete. But performing isolated batch operations is not supported by Cassandra when it comprises multiple partition changes. The transactions in the database are durable, as all the write operations are recorded in a commit log and memory. In the event of a node failure before the flushing of memtables to the disk, the commitlog helps in recovery of lost writes. Replication in Cassandra contributes to the durability of the transactions. Another feature of Cassandra is that it offers light w eight transactions often referred as compare and set transactions [10]

VII. Performance Test Plan

For testing the performance of HBase and Cassandra, a java based tool YCSB has been taken into consideration. The test follows several iterations to test both the databases in different workloads. A test harness file is used to bind the databases with YCSB. The configurations and the specifications required for the test environment are as follows:

A. System Specifications

Native Machine:

- Processor: AMD A8-6410 APU with AMD Radeon R5 Graphics, 4CPUs 2.0 GHz
- Memory: 16 GB
- Operating System: Windows 8.1

```
sidewinder@dsm1:~$ lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              2
On-line CPU(s) list: 0,1
Thread(s) per core:  2
Core(s) per socket:  1
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               79
Model name:          Intel(R) Xeon(R) CPU E5-2673 v4 @ 2.30GHz
Stepping:            1
CPU MHz:             2294.686
BogoMIPS:            4589.37
Virtualization:      VT-x
Hypervisor vendor:   Microsoft
```

Figure 4 lscpu console of Virtual instance

Virtual Instance: OpenStack

- Processor: Intel Xeon CPU E5-2673 v4 2.30 GHz
- VCPUs: 2
- Memory: 8 GB
- Hadoop 3.1.2 for HBase
- Apache HBase 1.4.0
- Apache Cassandra 3.11.4
- YCSB-0.14.0 Yahoo! Cloud Servicing Benchmark

Figure 4 shows the other system specifications of the virtual instance launched on OpenStack. This is achieved by executing lscpu command.

B. Workloads Selected:

YCSB[15] is an open source benchmarking tool that offers different workloads to test different read and writes in a distributed environment. It offers five different workloads; these workloads vary in nature. Following are the selected workloads;

- Workload A: Update heavy with 50-50 reads and updates
- Workload D: Read latest with 95-05 read-inserts

C. No. of Opcounts Defined:

The performance tests are executed with different number of operational counts.

- 100000
- 125000
- 175000
- 200000
- 250000

The performance test strategy is to install and run standalone HBase and Cassandra data- bases on a virtual machine deployed on OpenStack. To achieve this an OpenStack VM was launched with minimal configuration. Hadoop was installed to support the underlying HDFS system of HBase. Later HBase and Cassandra were installed and tested. YCSB was installed and configured to test both the databases. A set of configuration files called testharness were used to define the no. of opcounts, the test database and the workloads.

VIII. Evaluation

To evaluate the performance of HBase and Cassandra, YCSB-0.14.0 was used. As suggested by the previous evaluation studies over NoSQL databases and the performance evaluation of distributed cloud environments, Yahoo Cloud Servicing Benchmark is chosen do draw the analysis. Form different workloads offered by YCSB, workload A and workload D are selected. Both of these workloads have different specifications. The evaluation of both databases is executed on a single virtual instance with 2 VCPUs and 8GB of memory. After each test run, YCSB generates data over different metrics of the operations involved in the workload. Metrics such as throughput, runtime, scavenge time, sweep time etc. Since the operations involved in the workloads present the metrics for evaluation, average latency is taken over the operations performed.

A. Workload A:

The YCSB client's workload A is an update heavy workload. It uses read() and update() methods to evaluate. These methods represent the standard create, read, update, delete operations in a database.

i. Average Read Latency over Read operations:

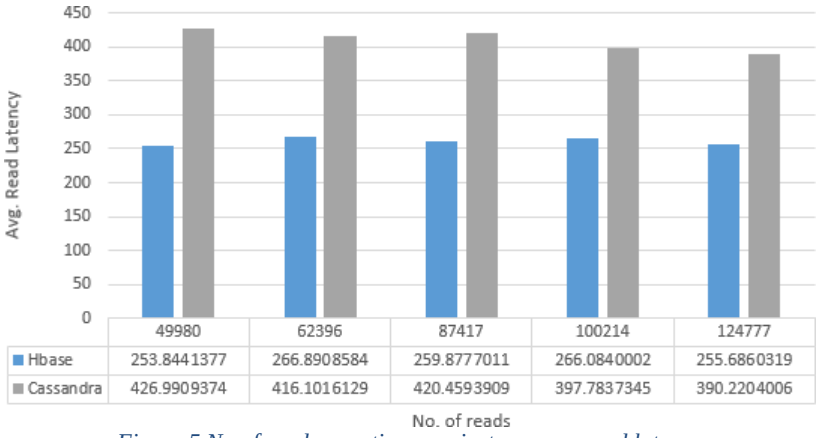

Figure 5 No.of read operations against average read latency

The above given data is the representation of the performance variations in workload A. The graph in Figure 5 is plotted between the average read latency of the operations. The test is executed in five iterations with different operational counts. It can be observed that HBase has less latency in over the read operations and the latency stayed almost the same except for the operations more than 100000. and the latency tends to decrease with further increase in reads.

ii. Average Update Latency over Update operations

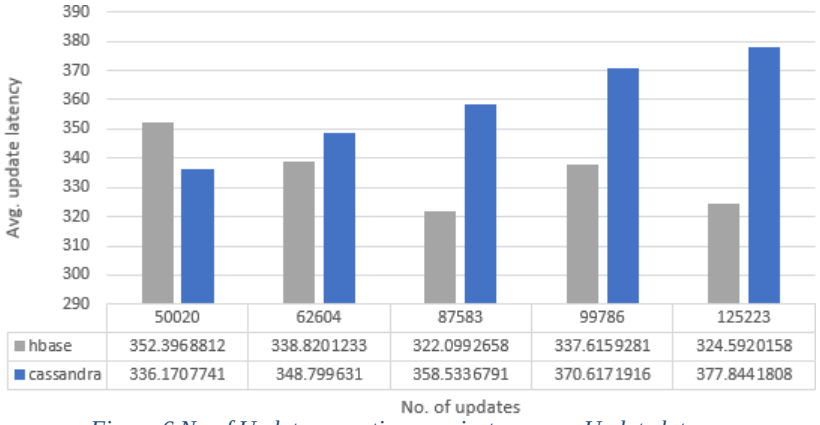

Figure 6 No.of Update operations against average Update latency

From Figure 6, it can be observed that from the given operational counts, the Update latency in Cassandra increased with the increase in no. of updates performed. But HBase has fluctuating latency over different no.of update operations.

Reads	HBase	Cassandra
49980	253.8441	426.9909
62396	266.8909	416.1016
87417	259.8777	420.4594
100214	266.084	397.7837
124777	255.686	390.2204

updates	HBase	Cassandra
50020	352.3969	336.1708
62604	338.8201	348.7996
87583	322.0993	358.5337
99786	337.6159	370.6172
125223	324.592	377.8442

Table 1 Average update and read latency in HBase and Cassandra

The above given data represents the no. of Read and Update operations performed and the recorded average latency for both the operations. As YCSB presented different metrics for performance evaluation, the throughput performance is also considered.

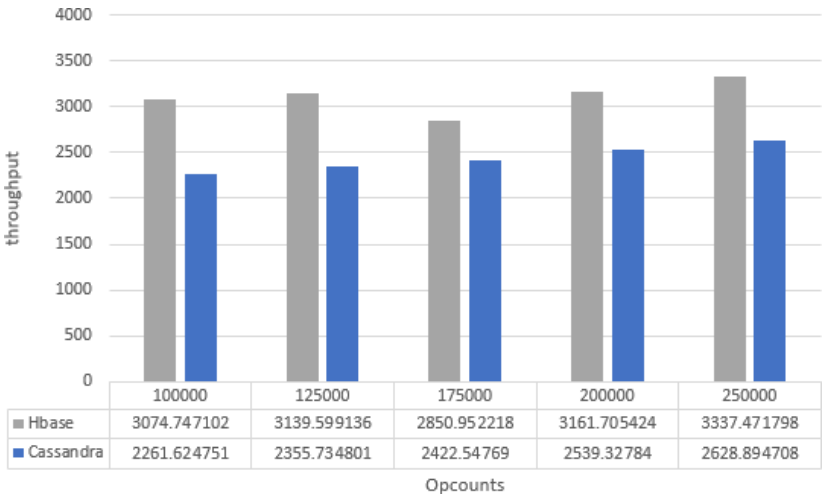

Figure 7 Throughput over no.of operations in HBase and Cassandra

The throughput performance of both HBase and Cassandra are presented in Figure 7. It is observed that Cassandra has a rise in throughput with increase in workload. whereas HBase showed equivalent throughput but higher throughput is recorded at the highest operational count i.e 250000.

B. Workload D

Workload D offered in YCSB is a read latest workload which has 95 percentiles of read operations and 05 percentile of insert operations. These operations are executed in several iterations similar to workload A.

i. Average Read Latency against No. of Read operations:

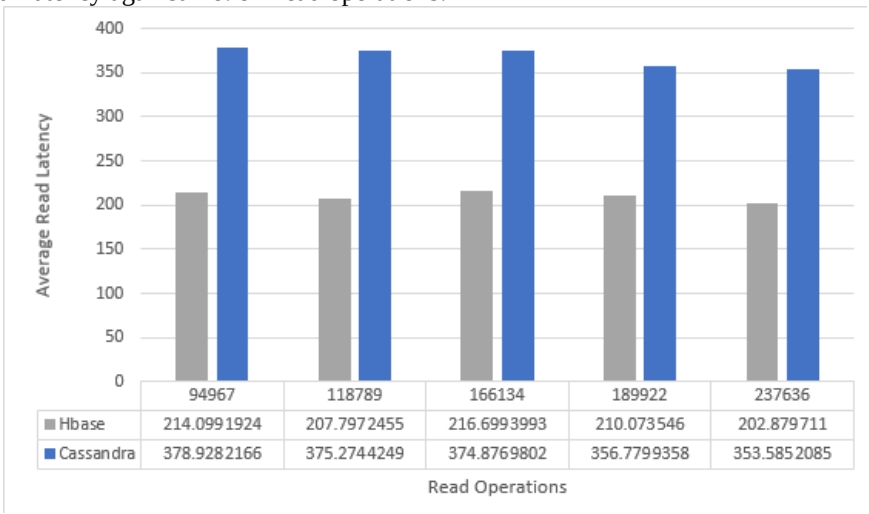

Figure 8 Average Read Latency over no. of reads

Figure 8 shows the differences in average latency in read against the number of read operations in the database. It can be seen that Cassandra has higher average read latency than HBase but all the operation counts are relatively equivalent.

ii. Average Insert latency against No.of Insert operations:

It can be seen in Figure 9 that the average insert operations in Cassandra are equivalent to all the different counts of inserts but Hbase showed significant peaks in workload D at 8866 and 10078 inserts. As workload D is a read latest workload, less inserts are observed. Higher insert operations are performed with the higher number of operational counts. The graph in Figure 10 represents the throughput of each operation for different number of operations. From the recorded observations, it can be seen that Cassandra has slight increase in the throughput with increase in operational counts. On HBase, fluctuations in throughput are observed. at the beginning of the workload, the throughput was high and the highest was recorded at 250000 opcounts.

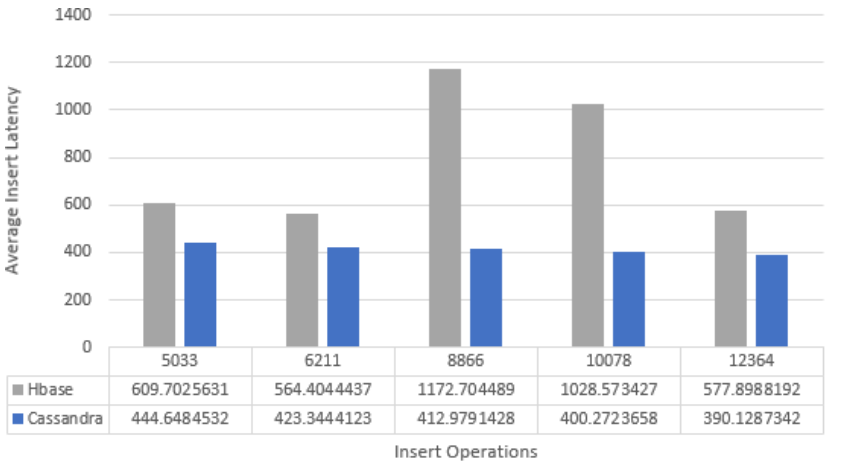

Figure 9 Average Insert latency over Insert operations

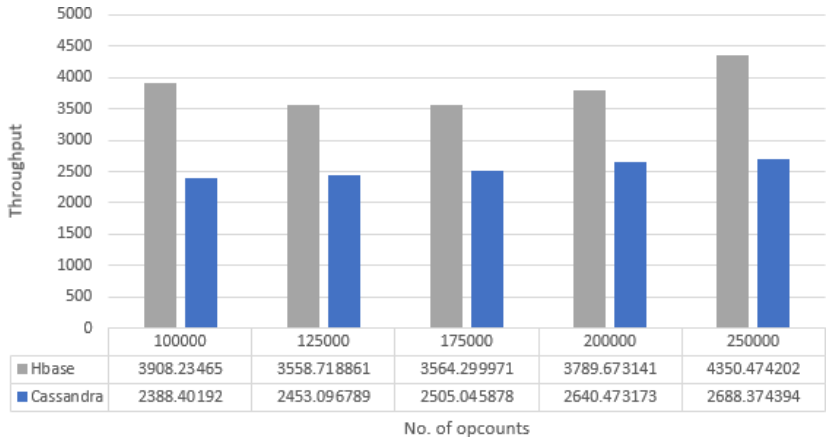

Figure 10 Throughput of databases over opcounts

Table 2 represents the data that is recorded in workload D. As the workload is read latest both read operations and the insert operations are considered for both the databases. Throughput is also given to determine the best performing database among the two.

Read Latency			Thtoughput			Insert Latency		
Read	Hbase	Cassandra	Opcounts	Hbase	Cassandra	Insert	Hbase	Cassandra
94967	214.0992	378.92822	100000	3908.235	2388.402	5033	609.70256	444.6485
118789	207.7972	375.27442	125000	3558.719	2453.097	6211	564.40444	423.3444
166134	216.6994	374.87698	175000	3564.3	2505.046	8866	1172.7045	412.9791
189922	210.0735	356.77994	200000	3789.673	2640.473	10078	1028.5734	400.2724
237636	202.8797	353.58521	250000	4350.474	2688.374	12364	577.89882	390.1287

Table 2 Workload D Data

IX. Conclusion and Discussion

NoSQL databases are gaining attention with the rise in distributed technologies, digital media etc. Unlike the conventional relational DBMS, NoSQL are non-relational and they offer more functionalities to store different kinds of data. From the performance evaluation conducted over HBase and Cassandra, several variations in the database behavior has been observed and presented in the evaluations. HBase being backed up with HDFS, it has a relatively larger ecosystem that Cassandra. Both the databases have performance exceptions at higher workloads. At workload A, Cassandra has fluctuating latency during read operations but HBase presented significantly less latency during reads. with the rest of the operations as updates, HBase showed the fluctuating records but Cassandra has increased latency. Runtime for each operation can also be considered but to know the performance of the virtual machine running both databases other metrics had to be considered. however, form the observations, Cassandra has higher runtimes than HBase.it can be assumed that it is because of the Hadoop map reduce framework.

Though YCSB helped in the evaluation of performance with different metrics, multiple databases cannot be evaluated in a single execution of the tool. However, features such as high scalability and the fast writes speeds of Cassandra can help when executing higher workload as observed. The properties of HBase such as SQL type execution and properties shared with BigTable can make the database suitable for write heavy applications. And significant performance increase can be observed on a native deployment as the network

latency issues can be mitigated. Though YCSB is successful in presenting the metrics, new metrics can be can also be included and tested. There is a scope in research towards forming new metrics and simplified performance evaluation tools.

REFERENCES

[1] R. Lomotey and R. Deters, "Data mining from document-append nosql," *International Journal of Services Computing* (IJSC), vol. 2, no. 2, pp. 17–29, 2014.

[2] "Apache HBase Documentation." https://hbase.apache.org/ Accessed 02-08-2019.

[3] "Apache HBase Architecture Overview." https://hbase.apache.org/book.html Accessed on: 03-08-2019.

[4] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "BigTable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[5] V. Bhupathiraju and R. P. Ravuri, "The dawn of big data-hbase," *Conference on IT in Business, Industry and Government (CSIBIG)*, pp. 1–4, IEEE, 2014.

[6] J. M. Zahoor, "HBase HMaster Architecture," 2012. http://blog.zahoor.in/2012/08/ hbase-hmaster-architecture/ Accessed on: 04-08-2019.

[7] A. Mehra, "Introduction to Apache Cassandra's Architecture," 2015. https://dzone. com/articles/introduction-apache-cassandrasAccessed on: 04-08-2019.

[8] R. Savaram, "Apache Cassandra Architecture Overview."

[9] "Apache Tephra." https://tephra.apache.org/ Accessed on: 04-08-2019.

[10] P. Rana, "Transaction Management in Cassandra," 2017. https://blog.knoldus.com/ transaction-management-in-cassandra/ Accessed on: 09-08-2019.

[11] A. Khattab, A. Algergawy, and A. Sarhan, "Mag: A performance evaluation framework for database systems," *Knowledge-Based Systems*, vol. 85, pp. 245–255, 2015.

[12] A. T. Kabakus and R. Kara, "A performance evaluation of in-memory databases," *Journal of King Saud University-Computer and Information Sciences*, vol. 29, no. 4, pp. 520–525, 2017.

[13] F. Ricca, "A java wrapper for dlv.," *Answer Set Programming*, Citeseer, 2003.

[14] A. Gandini, M. Gribaudo, W. J. Knottenbelt, R. Osman, and P. Piazzolla, "Perform- ance evaluation of nosql databases," *European Workshop on Performance Engineering*, pp. 16–29, Springer, 2014.

[15] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," *Proceedings of the 1st ACM symposium on Cloud computing*, pp. 143–154, ACM, 2010.

[16] J. de Oliveira Assis, V. C. Souza, M. M. Paula, and J. B. S. Cunha, "Performance evalu- ation of nosql data store for digital media," *12th Iberian Conference on Information Systems and Technologies (CISTI)*, pp. 1–6, IEEE, 2017.

[17] V. D. Jogi and A. Sinha, "Performance evaluation of mysql, cassandra and hbase for heavy write operation," *3rd International Conference on Recent Advances in Information Technology (RAIT)*, pp. 586–590, IEEE, 2016.

AUTHORS

First Author – Prashanth Jakkula, M.Tech Computer Science and Engineering, M.Sc in Cloud Computing, National College of Ireland