National College of Ireland

Data Storage and Management Project

on

Performance Analysis of Cassandra and HBase Using YCSB Benchmark Test

Shareen Shahana Marottikal khader
x18141404

MSc Data Analytics – 2019/20

Submitted to: Noel Cosgrave

National College of Ireland

Project Submission Sheet – 2017/2018
School of Computing

School of Computing

Student Name:	Shareen Shahana Marottikal khader
Student ID:	x18141404
Programme:	MSc Data Analytics
Year:	2019/20
Module:	Data Storage & Management
Lecturer:	Noel Cosgrave
Submission Due Date:	08/05/2019
Project Title:	Performance Analysis of Cassandra and HBase Using YCSB Benchmark Test

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:	May 08, 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 the 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):	

Table of Contents

Performance Analysis of Cassandra and HBase Using YCSB Benchmark Test

Shareen Shahana Marottikal Khader

Abstract

NoSQL is used by many Tech giants such as Twitter, Facebook, Netflix, Google, and Amazon. This is mainly because of the ability of NoSQL to scale without downtime. It is also highly available, reliable and scalable. Since Tech giants are using NoSQL, small and mid-level companies are signing up to use NoSQL. However, a proper benchmark must be done to choose which database is apt for client requirement. The Report used Yahoo! Cloud Serving Benchmark Tests to analyze the performance of Cassandra and HBase. The Report analyzed that of 50% read and 50% write operation with large Opcount Cassandra performs well than that of HBase. For 100% Read Operation it is evident that HBase performs better than Cassandra for a limited Opcount. The Report also inferred that security features of Cassandra is better when compared to HBase.

1 Introduction

In the Old times Human beings used to store information of libraries, hospitals, government offices, etc. In the Modern era, people decided to store these data into the computer with the help of databases. In 2009 relational database management systems were introduced as there was increase in the need of data storage management system. The Relational Database is not so good with an unstructured, large number, denormalized, not schema-oriented data. It is difficult to scale relational database into a large scale which leads to the arrival of NoSQL databases such as Apache Cassandra, HBase, MongoDB, etc.

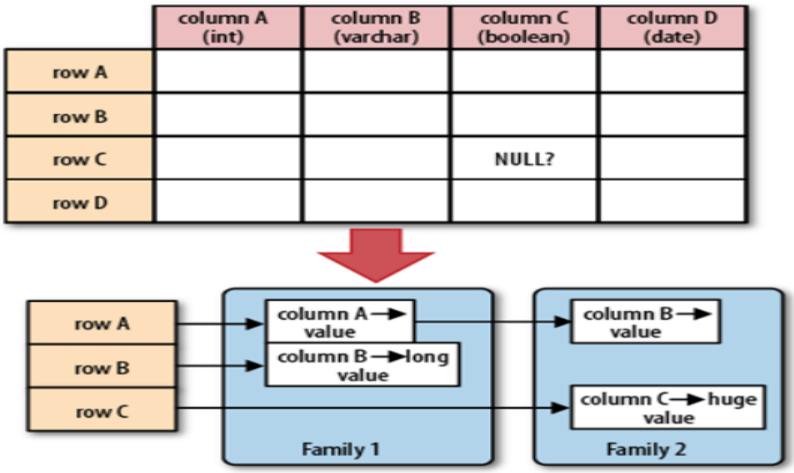
DBaaS providers always limit their hardware information that their systems are provisioned on. Yahoo developed a Java open-source specification and program suite called YCSB to compare the relative performance of different NoSQL Databases using workloads. This Report analyses the performances between Cassandra and HBase using the YCSB benchmark Tests. Yahoo! Cloud Serving Benchmark Test includes Workload A, Workload B, Workload C, Workload D, Workload E, Workload F. Workload A is a heavy Workload which has 50 % reads and 50 % writes. Workload B is 95% read workload and 5% write workload. workload C is 100% of the reading workload. Workload D is read recent workload updated. Workload E is reading short range records. workload F is read, write and modify workload. The Report evaluates the NoSQL database Cassandra and HBase by performing Workload A and workload C benchmark tests.

2 HBase

In 2007 Apache HBase is developed on top of the Hadoop database system as part of Apache Software License, version 2.0.it is developed right after Google's Bigtable. HBase is a distributed column-oriented, non-relational NoSQL, scalable, opensource Database System with the dynamic database schema.it stores data as key-value pair column vise.it stores data with the help of HDFS support MapReduce jobs. It handles the shifting load and failures gracefully when compared to other databases. HBase is a tool developed using java. the real-time read and write access to big data can be achieved using HBase. It claims that HBase is designed to host billions of columns X millions of rows of the very large table. It also distributed system it supports failure over on regional servers. HBase consists of a set of tables, which are stored in HDFS also as Hadoop makes use of batch processing, therefore search operation for large datasets are easily performed. Ali Hammood (October 2016).

The read and writes were strictly consistent on HBase.it uses sharding technique for database partition of tables. There are Java APIs available for easy client access. reading techniques use Bloom Filters and Block cache to improve the performance. HBase tables are connected using Thrift Gateway APIS and REST-ful Web service. these APIS supports and binary data encoding, XML and Protobuf. HBase uses predicate pushdown to filter to reduce the data sent to the network. HBase also has JRuby-based (JIRB) shell. Hadoop metrics subsystem is used to export metrics with the help of JMX and Files or Ganglia.it replicate the data across the clusters. HBase handles the failure is automatically. The record provides low latency to access millions of records hence randomly accessed. the randomly accessed property is achieved by implementing a hash table to store key-value pair for HDFS for faster lookup. HBase is a schema-less database which defines only column families.

The basic building block of HBase is a column.one or more column forms a row and each row is referenced by a row key uniquely. The row is alphabetically sorted by the multiple rows can form a table. The database will have multiple tables.



2.1 characteristics of HBase

The following are the key characteristics of HBase.

- **Consistency**

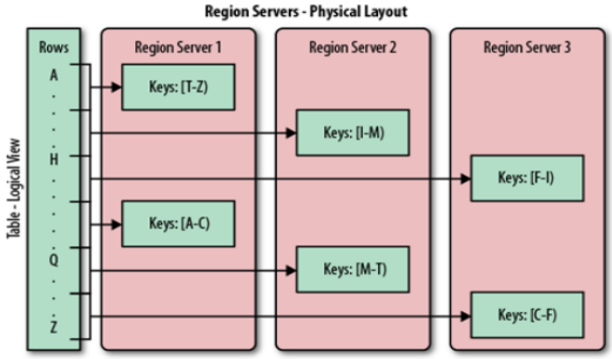
  High performing environment can choose HBase as Database because it offers consistent speed for both reads and writes

- **Atomic Read and Write**

  Automatic read and write means If one process is reading or writing all other processes will be held from reading or writing.so we can see that application is single threaded which will intern prevents fault during multiple manipulations.

- **Sharding**

  Sharding is a process of splitting the region into subregions as soon as it reaches the threshold. by this technique, HBase can decrease I/O processing time



- **High Availability**
  Data is high available even if any node failure occurs. Failure overcome is done using The master server monitors all the sharding metadata including all regional servers.

- **Client API**
  It can be easily accessible programmatically access through Java API'S.

- **Scalability**
  The linear and modular form of scalability is supported by HBase.

- **Hadoop/HDFS integration**
  HBase can be integrated with another system such as Hadoop or HDFS

- **Distributed storage**
  Storage mode is distributed that means the data is spread over multiple systems.

- **Data Replication**
  HBase operates on data replication principle. The data backup copied over multiple nodes in case of failover recovery.

- **Failover Support and Load Sharing**
  Failover support and load sharing are automated in HBase as it uses multiples regional servers.

- **API Support**
  It can be easily accessible by Client API'S with the help of Thrift gateway and a REST-ful Webservices.

- **MapReduce Support**
  Using this property HBase support parallel processing on big data.

- **Backup Support**
  HBase table is backup using Hadoop MapReduce job

- **Sorted Row Keys**
  HBase rows are stored in sorted order hence accessing data will be optimized.

- **Real-time Processing**
  Using block cache and Bloom filters HBase supports real-time query processing.

- **Faster Lookups**
  Random access of HBase is achieved using a hash table .indexed HDFS data storing method also help for faster lookup.

- **Type of Data**
  HBase supports structured as well as semi structured data well.

- **Schema-less**
  HBase is column type is family and it is the schema-less database.

- **High Throughput**
  HBase offers high throughput due to its high security and easy management characteristics.

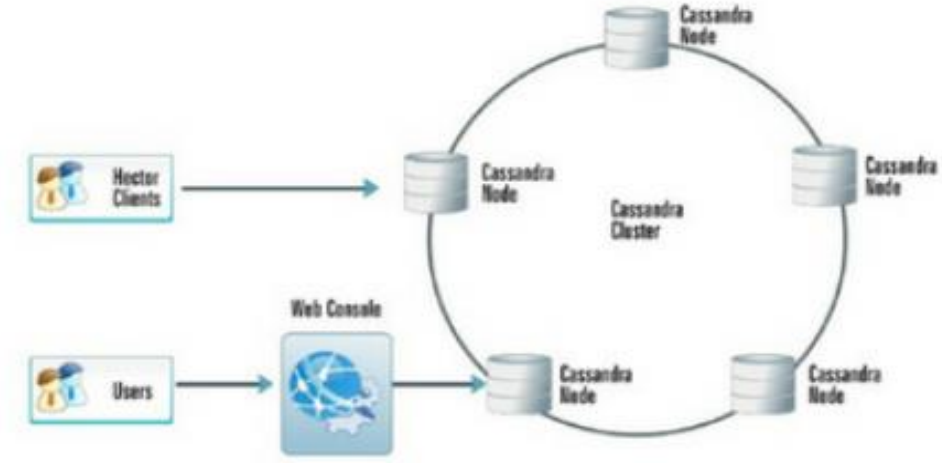
3 Apache Cassandra

Apache Cassandra database is chosen when a system needs to scale and made available without reducing its performance. Linear scalability and fault-tolerance most important characters are proven by Apache Cassandra.it provide high-performance decentralized, distributed, open source, column-oriented, consistent, scalable, fault-tolerant, distributed storage system service without a single point of failure.

Apache Cassandra is having Google's Bigtable Data model and Amazon's Dynamo distribution design. Apache Cassandra is created on Facebook. it operates entirely different from the relational database. The Dynamo-style replication model with no single point of failure is used for the implementation of Cassandra. Facebook, Twitter, Cisco, Rackspace, eBay, Twitter, Netflix, and more big companies use Cassandra as their distributed storage system.

Difference between RDBMS and Cassandra is that Cassandra deals with unstructured data while RDBMS deals with structured data. Cassandra is flexible schema and RDBMS is fixed schema. In RDBMS array of Arrays used as a table while in Cassandra key value pair is used. RDBMS table is the outer most container that contains data while in Cassandra key is used as an outer most container. The tables are the entities of the RDBMS database where a column of families or table is the entities of the keyspace.

The Row is an individual record in RDBMS, while the row is unit of replication in Cassandra. the column represents an attribute of relation in RDMD while the column is a unit of storage in Cassandra. RDMS support concept of foreign keys, join while in Cassandra relationships are represented using the collection.



The figure above shows how apache Cassandra is deployed in distributed mode. Users and hector clients access Cassandra nodes connected distributed mod in the Cassandra cluster.

3.1 characteristics of Cassandra

The following are the main features of Cassandra.

- **PROVEN**
  The Cassandra is used over 1500 more Bigdata companies such as Constant Contact, CERN, Comcast, eBay, GitHub, GoDaddy, Hulu, Instagram, Intuit, Netflix, Reddit, The Weather Channel.

- **FAULT TOLERANT**
  Cassandra is designed in such a way that it is fault tolerant. if any node goes downs distributed node will operate and use replica node with no time.

- **PERFORMANT**
  Cassandra outperforms all other competitive NoSQL alternatives .this could be achieved due to the architecture of Cassandra choose.

- **DECENTRALIZED**
  This says that Cassandra does not have any center node to cause a single point of failure. Every node in the cluster is identical.

- **SCALABLE**
  The Cassandra has the ability to grow the system in to large in production whenever required.

- **DURABLE**
  Cassandra is designed to support the system which cannot afford down time even if the entire data center goes down.

- **YOU'RE IN CONTROL**
  Cassandra each update is choosed carefully between synchronous or asynchronous replication.

- **ELASTIC**
  A new node machine can be added in Cassandra cluster without interruption or downtime to application. Whenever a new machine added read write throughput increase drastically.

- **PROFESSIONALLY SUPPORTED**
  The contracts and services are available from third parties are well supported by Cassandra.

- **Elastic scalability**
  The Cassandra is highly scalable and elastic. in order to add more data and more customer Cassandra just need to add nodes without interpreting the existing system.

- **Always on architecture**
  The Cassandra is fault tolerant it has no single point of failure. Cassandra is designed for heavy business application which cannot afford failure.

- **Fast linear-scale performance**
  The Cassandra is linearly scalable as whenever we add a node to Cassandra cluster the throughput increases linearly. This quickly decreases response time.

- **Flexible data storage**
  The structured, semi-structured, and unstructured data types are supported by Cassandra.
  It can support change in data structure dynamically.

- **Easy data distribution**
  Data replication distribution across Cassandra cluster is easy and flexible.

- **Transaction support**
  Cassandra supports ACID (Atomicity, Consistency, Isolation, and Durability) property

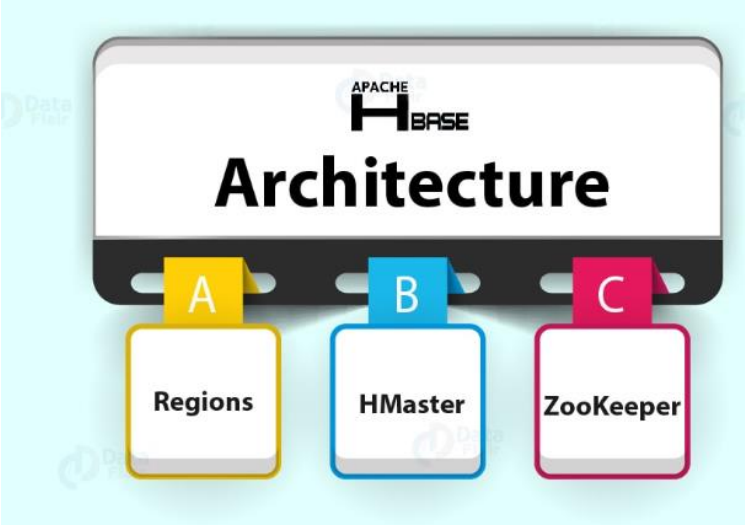
- **Fast writes**
  Cassandra is compactible to hardware with lower specifications. Cassandra is known for fast writes and store hundreds of od terabytes easily with of reducing read efficiency.

4 Database Architecture

Data base architecture of HBase and Cassandra are discussed below

4.1 HBase

HBase Architecture consists of three major components regions, Hmaster and Zookeeper. All three of them are HBase servers.

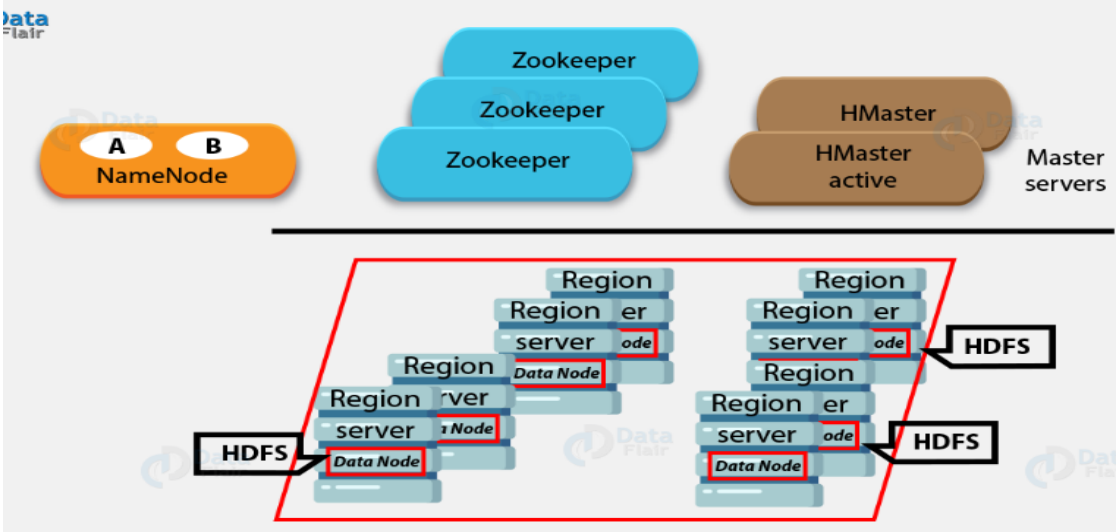


The Region servers will provide data access to clients while reading and writes.it is the number of rows between a start key end and end key which is assigned to the region.it can serve ten thousand regions. each region is managed in sorted order. Default region size is 256MB which is configurable.

The HMaster will be performing ddl operations as well as region allocation process. The Responsibilities of Hmaster in HBase are Coordinating the region servers and Admin functions. Admin functions basically include the DDL operation on HBase tables. Coordinating the region servers is the process of monitoring all the region's instances are up. And it also assigns the regions on start-up.

The zookeeper helps to maintain live cluster state in HBase. The zookeeper will contain all the information about available servers which are alive and will also notify the cluster if any node is down.

The data Node is used store the data which will be accessed through region server. That is if a client needs data access they can contact region server which will be query the data node and return the data to the client. In HBase data is stored in the form of file is called HDFS. The data node is collocated with Region Server which will enable Data locality.
data locality means keeping the data close to the location we need. Name node is the meta data of the information stored.



Zookeeper maintains all the information about the active nodes through a regular heartbeat request is called Ephemeral nodes. HMaster and region servers communicate with a zookeeper to know which server has to be used for the operations. An ephemeral node exists as long as the session is active. Using this process zookeeper determines which node has to be the right node for the operations. Each region server in HBase initiates an Ephemeral pulse request. From those requests, zookeeper determines active HMaster. As long as the heart beat received by the zookeeper from the HMaster, it will consider HMaster is alive, otherwise, it will consider as a failure of HMaster and then new HMaster is assigned to the cluster. If any of the HMaster or region nodes fails, then zookeeper considers as the session expired. All the listeners will be notified with deleted nodes details. If a region server is failed active HMaster will recover the region servers as soon as it receives the failure request. Similarly, an HBase HMaster is inactive it receives HMaster failure request, it will reboot automatically and serve as an HMaster.

Read or Write

If a client wants to read or write in to any table in HBase can send a request to zookeeper which will return region server details which contains the meta row data.

META Table

The Meta table is a catalog table which holds all the region details in HBase Cluster. The structure of the meta table is like a binary tree consist of key and values. Where the key is region Id and value is the region server.

Region Server Components

The components of Region Server are listed below

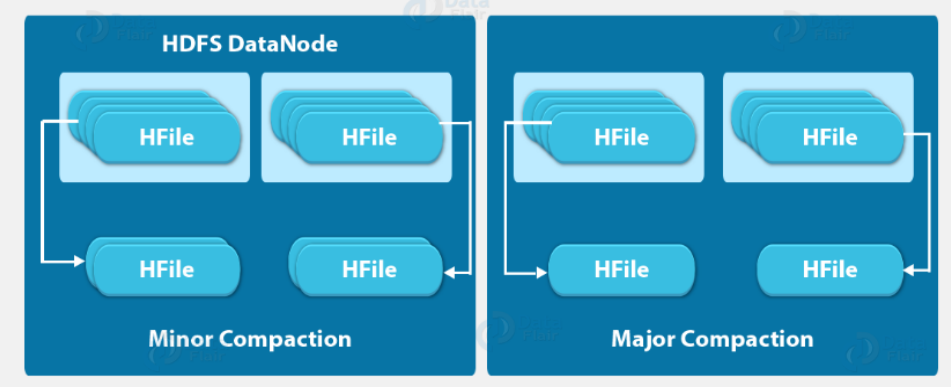
- **WAL:** it is a file in HBase which new data is stored before persisting in to permanent storage. We also use the same file for the recovery of the node on failure.
- **BlockCache:** is cache stored to use read operations. also delete of the ache occur based on the least used member.
- **MemStore:** it is cache used for writing process. its function is to store new data which is not yet stored permanently
- **Hfiles:** these are the files stores the rows as sorted key value pairs.

Write Steps

- The first step is to write the data in to write-ahead log (WAL) file whenever client sends a written request.
- The Put request acknowledgement is sent to the client after adding those data in to memstores.

Compaction

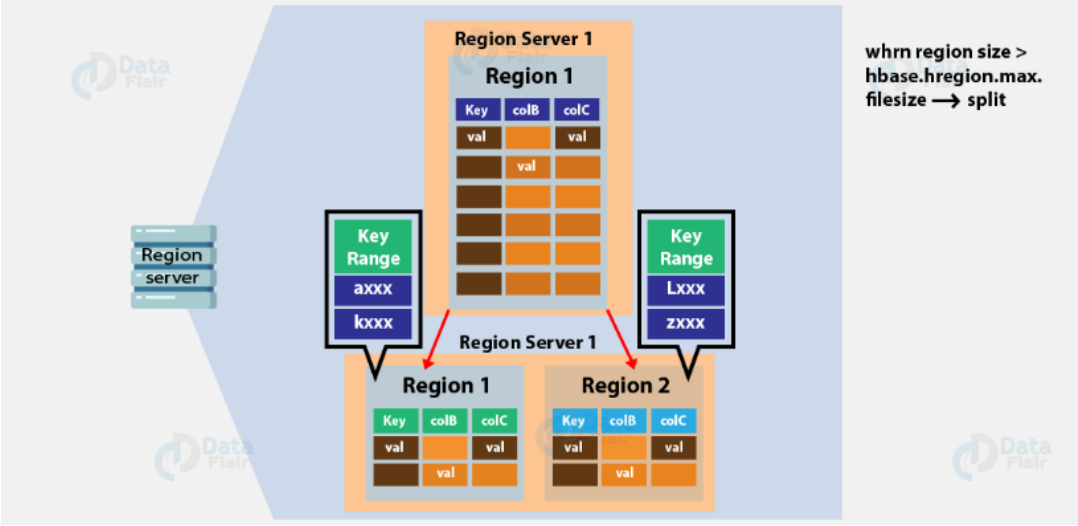
The process of combining the multiple HFiles in order to reduce the number of disks seeks needed for a read and the storage is called compaction



- **Minor Compaction:** HBase uses merge sort to pick smaller Hfile and recommit it into bigger once
- **Major Compaction:** in this process, HBase merge and recommits the smaller HFiles of a region to a new HFile, it merges same column families together

Region Split

HBase Regions have Two child regions, the split is performed and informed in Hmaster when a region becomes very large then HMaster allocate the region into a new Region server for load balancing.



Data Replication

All read and writes of HBase is handled by Primary node of Hdfs. write-ahead logs and HFile blocks are replicated by hdfs. the Data replication is automated in HDFS along with Data safety .first copy is written in local then it is copied in HDFS . after this it replicates into

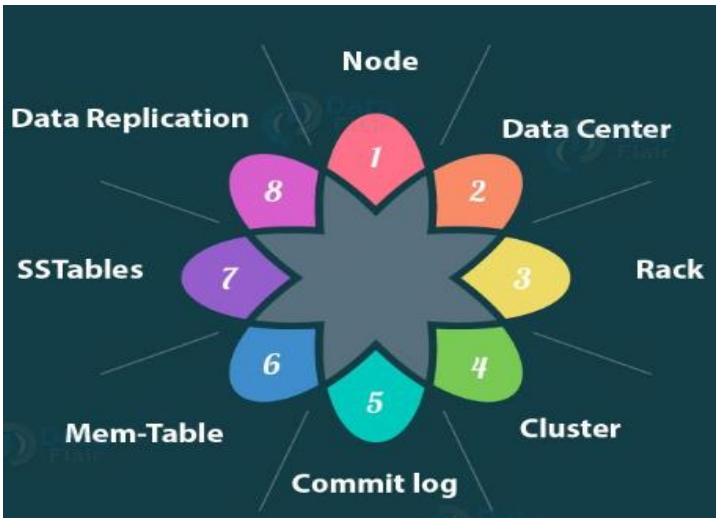
secondary and the tertiary nods so on.

Crash Recovery

The Zookeeper is notified by Hmaster whenever a region server fails. HMaster rebuilds the regions of Region server crashed.

4.2 Apache Cassandra

It is developed considering the hardware failure also when compared to HBase. Hence it Contains strategies to avoid a single point of failure. Cassandra is modeled a ring topology.



i.e. are its nodes are logically distributed like a ring. Hence Cassandra is lacking masters and slaves.it Replicates Data on all homogeneous nodes of Cassandra cluster.in order to make data durable, each writes activity is sequentially captured in to log. The nodes communicate with each other in every second to keep homogeneous behavior. The cache will be stored into Memtable. The Memtable is written after Indexing this data. when Memtable is full the data will be written into SSTable data File on disk. Automatically Data will be partitioned and replicated. Compaction process in Cassandra will help to update SSTables periodically and remove tombstones. The client can request to any node a read or write activity with the help the coordinator node.

Data Replication

Cassandra replicates data over the cluster in order to avoid the hardware failure on a single point. Following are the factors effecting replication in Cassandra.

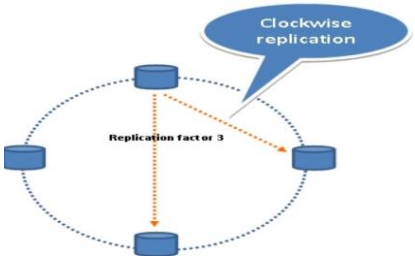
Replication Factor

The number of copies saved across the server if a single point of failure occurs. replicating factor is 4 means 4 copy of data is replicated over the cluster.

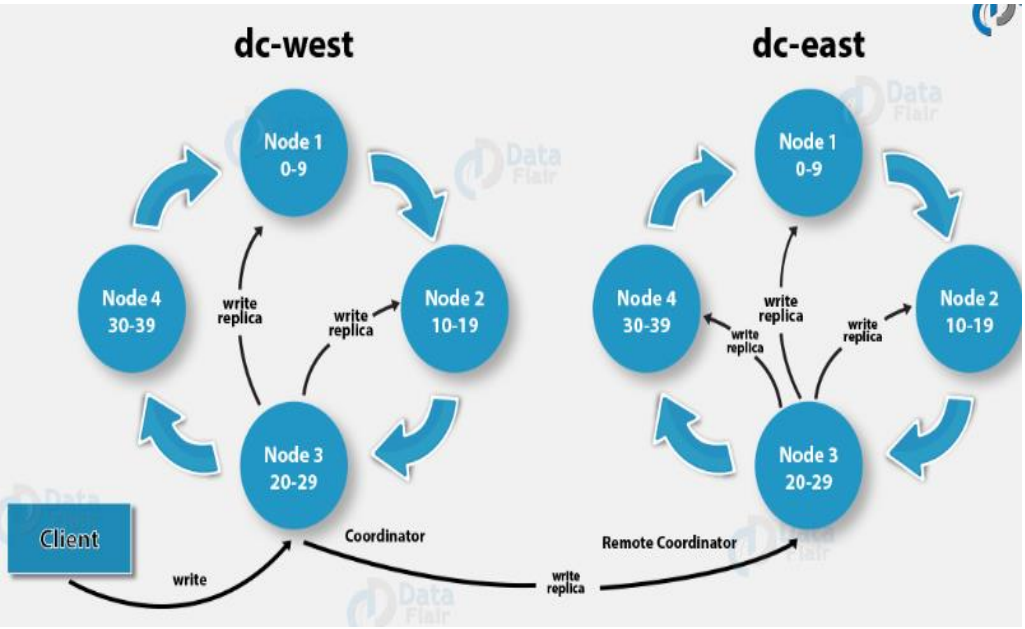
Replication Strategy

There are two types of replication strategy.

Simple strategy: Strategy is used when there is only a single data center for Cassandra cluster. Data is copied across all the node in clock wise direction.



Network topology strategy: This topology is highly recommended one as it has hope of expansion. each data center is logically connected in a ring manner and each data center has a rack of nodes in clock wise direction.



Following are the few terminologies used by Cassandra architecture.

Cassandra Nodes: Nodes are the basic unit of cassandra. The data will be stored in nodes

Cassandra Data Center: it is the collection of nodes. It is logically a centralized place

Cassandra Rack: The severs are placed on top another in a unit called rack.

Cassandra Cluster: it is the collection of multiple datacenters

Cassandra Commit log: every writing activity is recorded into the log in order to make data durable. This very useful to recover the data when the crash happens.

MemTables: this is equivalent to cache every delete or update operation is recorded in memtable. when the Memtable is full data will be written into SSTables on disk

SSTables: every periodic interval of time the memtable is flushed in to SSTables

Data Replication: in order to overcome single point of failure Cassandra replicate data over the cluster

5 Security in Cassandra and HBase

NoSQL is being popular on these days as tech giant Companies such as Facebook, Netflix, Google, and Amazon use NoSQL databases. NoSQL databases are suggested for clients having large data need to get high performance also a need to scale their database without bringing down existing infrastructure (system downtime). Availability and Scalability are definitely good but organization such as financial banks expect security also.

Cassandra and HBase are the trending Databases available on the market. It is necessary to check the security aspect of both databases. Major social networking sites such as Twitter, Facebook, etc. are using Cassandra while Companies like Bank of America, Bloomberg, Verizon and much more use HBase. Like all NoSQL databases, both Cassandra and HBase have many security issues. Security implementation will spoil the performance of the database by making them inflexible and heavy. But we can infer that both Cassandra and HBase provides some level of features that ensure the security of data. Cassandra has authentication and authorization in both inter-node, client-to-node encryption. HBase also provides secure communication with the technologies it relays upon. HBase provides some method by which data is secured from sniffers or other network attacks. HBase, provide cell-level access while Cassandra enable row-level access only. Cassandra has an option to define user roles. User level of data is access is defined for Cassandra. But in the case of HBase administrator must set visibility for data sets and only permitted users can access datasets.

HBase is secured using user authentication as well as MapReduce task used for secured token. Simple Authentication and Security Layer with the help of Kerberos is used to secure the user level. According to (Pallas et al. 2016) HBase uses two approaches to secure data with in and out of the network. HBase native security, HBase independent architecture VPN and VPC (virtual private clouds). The Author found that using HBase native security leads to significant performance impacts of up to 47% for realistic cluster sizes of 6 or 12 data nodes. hence this option is not at all feasible. The Author also conclude that The HBase independent architecture VPN and VPC can be choose over HBase native security. The author also found that users of cloud-based HBase deployments will get Either performant system or a secure one. deploying HBase in the cloud requires additional security measures. Data in transit within, to, and from an HBase cluster must be encrypted. Such security measures, though, come at a price.

According to (Heeyoul Kim et al 2016) outside attackers can access internal cluster structure information and transferring data. The Author Present novel way to enhance Cassandra security by means of the group key. The group key model is designed considering Cassandra's physical structure, and it is a decentralized model where the nodes are divided into several subgroups. The proposed model helps to protect the internal cluster from outside by confirming the node's cluster membership.

6 Literature Survey

According to Ali Hammood (2016), Cassandra and HBase are Colum Family databases both will perform very well with the cases where heavy loads. he also says that this is happening due to the optimized data model designs. Read operation HBase perform Less when compared to Cassandra as well as mongo db. This Report test bench march Test bet ween Cassandra and HBase. this Report agrees with Author that when Read operation HBase Average latency is more that means the through put will be less for the HBase. The Report shows that both Cassandra and HBase perform well with heavy loads like the author findings but specifies more that Cassandra performs better than HBase while Insert Operation than that of HBase. So, the Report agrees with the findings of Ali Hammood(2016).

According to Veronika (2014) in terms of optimization, NoSQL databases can be divided into two categories, the databases optimized for reads and the databases optimized for updates. Thus, MongoDB, Redis, and OrientDB are databases optimized to perform read operations, while Colum Family databases, Cassandra and HBase, have a better performance during execution of updates. The Report Tests the How Cassandra and HBase behave while Update Operation happens.it shows that Cassandra performs better that of HBase .and both HBase and Cassandra perform well with Updating operation as well. Which we can conclude that report agree with Veronika(2014) as well but it will further analyses bet ween HBase Cassandra, unlike her report.

According to Alexander (2015), Cassandra performs better than Voldemort, by giving overall lower latency and higher throughput in the same configuration with the same resources. The Report also Test that How Cassandra performs compared to HBase. the Report also found that Cassandra outperforms HBase, providing overall lower latency and higher throughput in the same configuration, hence we can conclude that Cassandra performs better than that of both HBase and Voldemort. This could be due to the decentralized data model design Cassandra. hence here The Report analyzed and found the Cassandra as an outperformer all together.

According to Houcine Matallah(2017), MongoDB is overall more efficient compared to HBase in the read operations: Read Only, Read Mostly, Heavy Update and Read write, unlike HBase which confirmed its better performance in write operations: Data loading, Update Only, Update mostly. The Report found that HBase is less competitor when compared Cassandra in all ways that are Read write, Read Only. Read and Update are Operation are compared for constant configurations. MongoDB is not considered here.

According to John Klein,.(2015) throughput that varied from 225 to 3200 operations per second between database products, while read operation latency varied by a factor of 5 and write latency by a factor of 4 (with the highest throughput product delivering the highest latency). The author also found that achieving strong consistency reduced throughput by 10-25% compared to eventual consistency

According to (Abramova et al., 2014a, 2014b, 2014c, Barata et al., 2015) Cassandra seems to have a clear advantage in terms of the characteristics necessary to implement this system because it provides good writes speeds without sacrificing performance.

According to (Pallas et al. 2016) HBase uses two approaches to secure data with in and out of the network. HBase native security, HBase independent architecture VPN and VPC(virtual private clouds).The Author found that using HBase native security leads to significant performance impacts of up to 47% for realistic cluster sizes of 6 or 12 data nodes. hence this option is not at all feasible. The Author also conclude that The HBase independent architecture VPN and VPC can be choose over HBase native security. The author also found that users of cloud-based HBase deployments will get Either performant system or a secure one. It difficult to get both performing and secured one as security implementation all together reduces the performance.

According to (Heeyoul Kim et al 2016) outside attackers can access internal cluster structure information and transferring data. The Author Present novel way to enhance Cassandra security by means of the group key. The group key model is designed considering Cassandra's physical structure, and it is a decentralized model where the nodes are divided into several subgroups. The proposed model helps to protect the internal cluster from outside by confirming the node's cluster membership

7 Performance Test Plan

The Report studies Two distributed NoSQL databases Cassandra and HBase using Yahoo Cloud Serving Benchmark (YCSB). This Report analyses the performances between Cassandra and HBase using the YCSB benchmark Tests. Yahoo! Cloud Serving Benchmark Test includes Workload A, Workload B, Workload C, Workload D, Workload E, Workload F. Workload A is a heavy Workload which has 50 % reads and 50 % writes. Workload B is 95% read workload and 5% write workload. workload C is 100% of the reading workload. Workload D is read recent workload updated. Workload E is reading short range records. workload F is read, write and modify workload. The Report evaluates the NoSQL database Cassandra and HBase by performing Workload A and workload C benchmark tests. The Test harness script utilized to execute benchmark Tests . each test is taken trice and the average is considered as Final value. This is done because each time the test value difference in certain extend. The output is exported into the local system as the zip file. Excel is created help of automated script which reads the Zip file and calculates the average values. Tableau is used for visualization.

7.1 User System Specification

- Machine: windows
- Model: XPS 15 9570
- Processor: 2.20GHz GHz Intel Core i7
- Operating System: High SierraRAM: 16GB LPDDR3

7.2 Virtual Machine Specification

- Cassandra
- HBase: Hadoop and HDFS distributed Mode

7.3 Open stack Configuration

- Cassandra - 3.11.4
- HBase - 1.4.9: Hadoop and HDFS
- YCSB - 0.14.0
- Java Development Kit
- Java Runtime Environment
- Python

7.4 Workloads considered for testing HBase and Cassandra Database Performance

- Workload A
- Workload C

7.5 Operational Counts used for Workloads

- 100000
- 150000
- 200000
- 250000
- 300000

8 Evaluation and Results

Performance evaluation of Cassandra and HBase is done here Using the YCSB Bench mark tool. YCSB has a bunch of predefined performance evaluation test which allows us to compare the Databases.it has a range of workloads we use here is Workload A and C by varying the operation count int 5 times for each Operational count workload is executed three times. this is to make sure the Data used for the Report is accurate. The Test harness script utilized to execute benchmark tests. Test harness is an automation framework used to execute the benchmark tests. Output of. Test harness is exported in to ycsb home path. The output of each test is varied by the operational count. The Test harness first Load /Insert the data in to Database followed by the bench mark Tests will run. The Output of the Frame work is Two Files. They are load response file and run the response file. Load response file will have details about how fast the load operation happened in a data base while the other other file is the result of Bench Tests. Visualization of output done using Tableau. basis of reliability, scalability, and availability comparison of Cassandra and HBase are done.

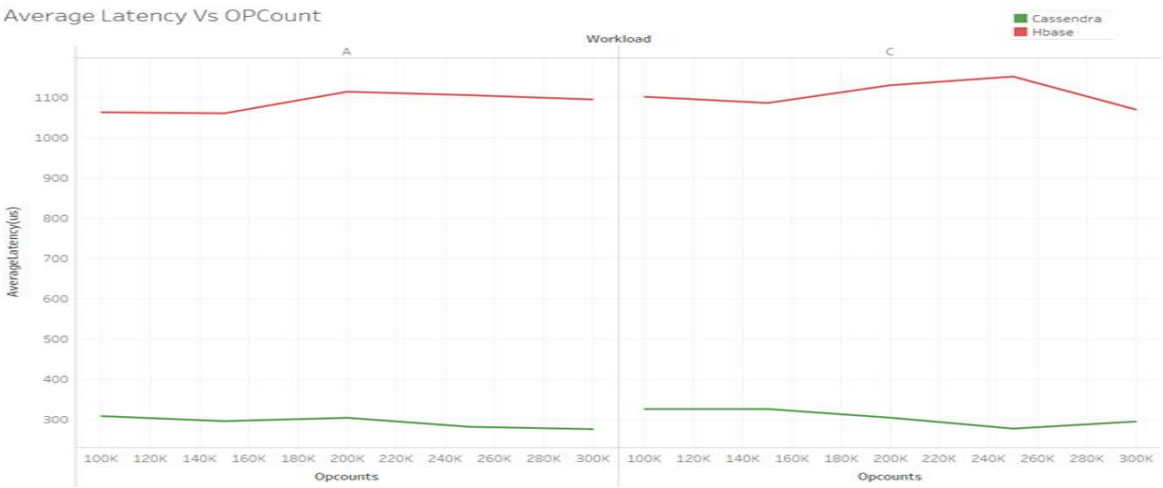
8.1. Loading

This section evaluated how Insert Operation perform for both workloads workload A and workload C load response file is evaluated the following data is collected. The average is calculated and for throughput, average Latency, and RunTime.

Database	Workload	Opcounts	record count (Insert)	AL1	AL2	AL3	AverageLatency(us)	TP1	TP2	TP3	Throughput(ops/sec)	RunTime	RunTime	RunTime	RunTime(ms)
Hbase	A	200000	200000	1114.267	1109.358	1119.296	1114.3069	884.4431	885.038	871.9802	880.487101	226131	225979	229363	227157.6667
Hbase	A	250000	250000	1105.424	1128.537	1083.381	1105.7809	894.4576	869.6832	902.3381	888.826304	279499	287461	277058	281339.3333
Hbase	A	300000	300000	1070.686	1167.492	1046.883	1095.02	917.9842	834.04	937.7843	896.602815	326803	359695	319903	335467
Hbase	A	100000	100000	1066.636	1063.082	1059.852	1063.1902	901.8759	909.281	908.7439	906.633622	110880	109977	110042	110299.6667
Hbase	A	150000	150000	1066.883	1042.825	1072.211	1060.6396	917.2685	904.3063	917.4031	912.992647	163529	165873	163505	164302.3333
Cassendra	**A**	100000	100000	281.6241	337.2895	306.0861	308.33323	2860.576	2469.502	2672.796	2667.62449	34958	40494	37414	37622
Cassendra	**A**	200000	200000	349.9876	286.9987	275.301	304.09576	2592.386	3099.67	3211.458	2967.83818	77149	64523	62277	67983
Cassendra	**A**	150000	150000	343.1903	260.2146	283.8933	295.76605	2569.197	3274.18	3027.795	2957.05734	58384	45813	49541	51246
Cassendra	**A**	250000	250000	292.6944	263.113	289.3986	281.73531	3229.515	3567.453	3260.09	3352.35286	77411	70078	76685	74724.66667
Cassendra	**A**	300000	300000	263.062	279.0067	285.9443	276.00432	3606.203	3405.299	3317.52	3443.00705	83190	88098	90429	87239
Hbase	C	250000	250000	1117.842	1112.191	1225.988	1152.007	884.9902	877.9539	804.3991	855.781062	282489	284753	310791	292677.6667
Hbase	C	200000	200000	1188.47	1076.047	1126.802	1130.4397	823.6825	911.1493	859.3575	864.729781	242812	219503	232732	231682.3333
Hbase	C	100000	100000	1084.974	1099.099	1121.517	1101.8632	892.9847	867.3629	859.6161	873.321226	111984	115292	116331	114535.6667
Hbase	C	150000	150000	1079.093	1105.094	1074.793	1086.3266	908.2047	890.821	901.1066	900.044087	165161	168384	166462	166669
Hbase	C	300000	300000	1035.027	1118.028	1056.678	1069.9109	944.995	876.3679	928.8069	916.723268	317462	342322	322995	327593
Cassendra	**C**	150000	150000	342.1194	322.7497	312.9933	325.95412	2572.59	2711.35	2774.797	2686.24569	58307	55323	54058	55896
Cassendra	**C**	100000	100000	315.8059	306.1599	355.3765	325.78075	2606.678	2676.874	2365.408	2549.65369	38363	37357	42276	39332
Cassendra	**C**	200000	200000	310.3138	280.4724	322.2263	304.3375	2887.169	3164.607	2792.438	2948.07151	69272	63199	71622	68031
Cassendra	**C**	300000	300000	315.5573	271.8588	297.1578	294.85798	3030.058	3487.845	3205.642	3241.18166	99008	86013	93585	92868.66667
Cassendra	**C**	250000	250000	283.8226	278.2971	269.7798	277.29981	3317.674	3256.693	3492.693	3355.68657	75354	76765	71578	74565.66667

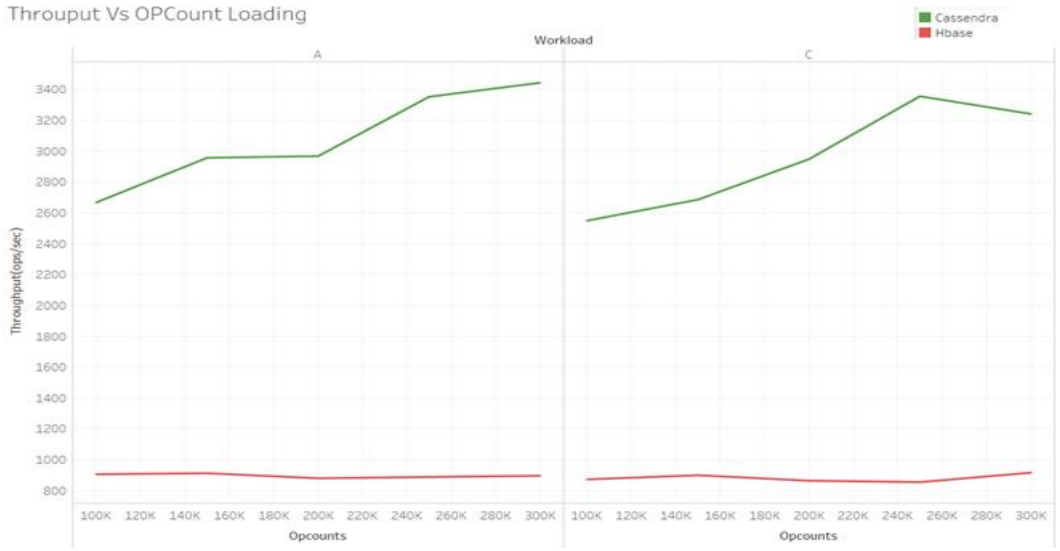
8.1.1. Average Latency VS OP Count for Load Operation

From the below figured it is analyzed that the average latency is less for Cassandra and more for HBase. It also noticed that in Cassandra operational count increases average latency reduces.



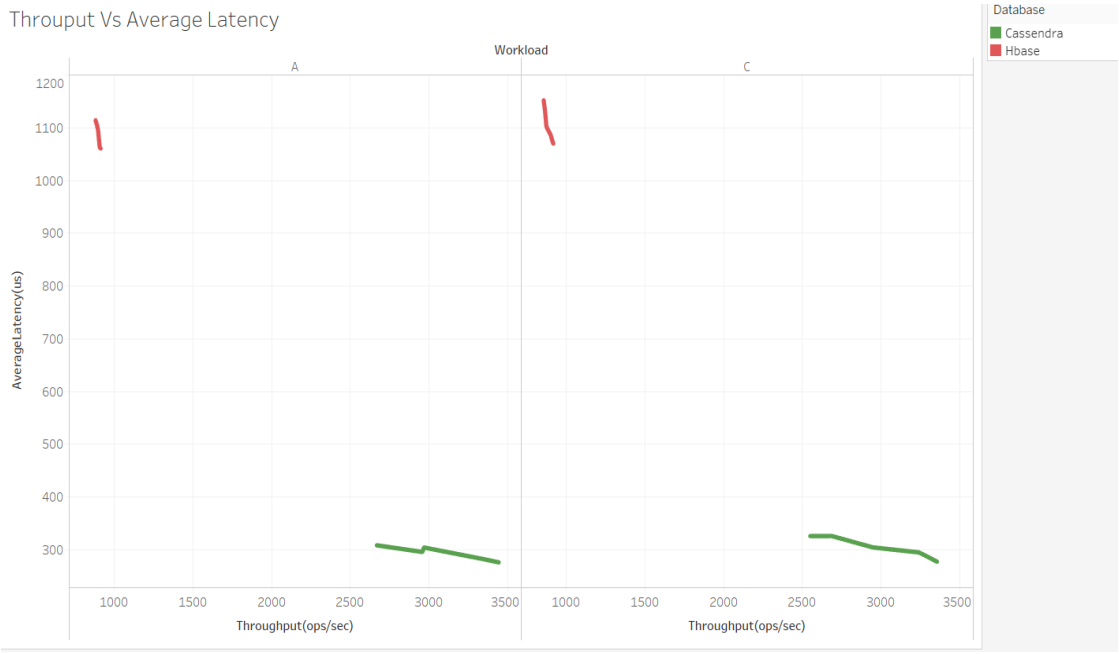
8.1.2. Through put VS OP Count for Load Operation

The scalability of Cassandra is directly proportional to Throughput. I.e. throughput increase drastically when operational(scalability) count increases.



HBase has less throughput compared to Cassandra. That means availability of HBase is less compared to the availability of the Cassandra while Loading Data in to DB.

8.1.3. Through put VS Average Latency for Load Operation

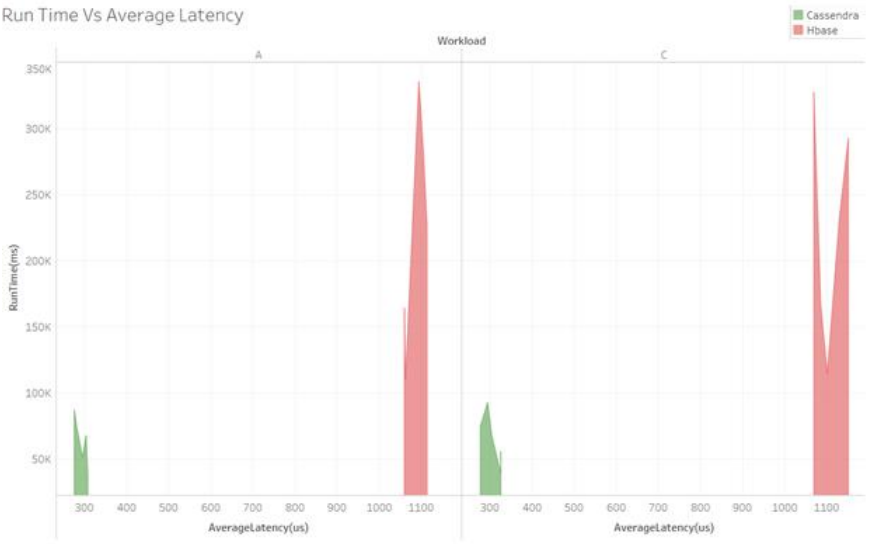


The above figure shows that Average Latency is inversely proportional to Throughput and vice versa. For both Workload A and C.

8.1.4. Run Time VS Average Latency Vs Through put for Load

The Average latency and Runtime are directly proportional and is less for Cassandra when compared to HBase. The report can conclude that Cassandra loads faster than HBase for both workloads. From the graph, Report can also conclude that throughput is inversely proportional to

runtime and Average latency is directly proportional to runtime.

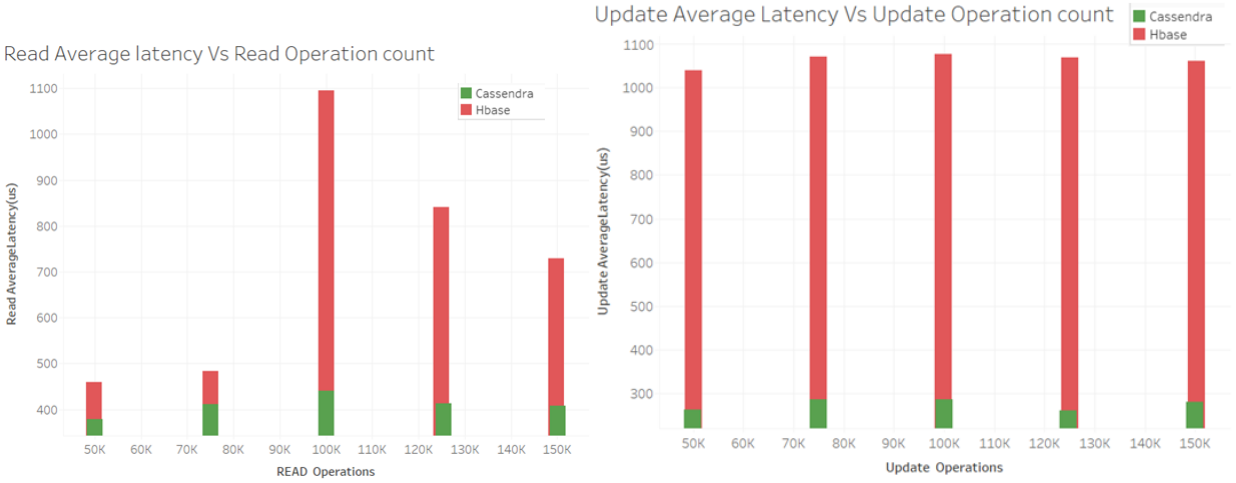


8.2. Running Bench Mark Testing

This section evaluated how Benchmark Test on Reading and Update Operation performance for both workloads workload A and workload C. Run response file is evaluated, and the following data is collected.

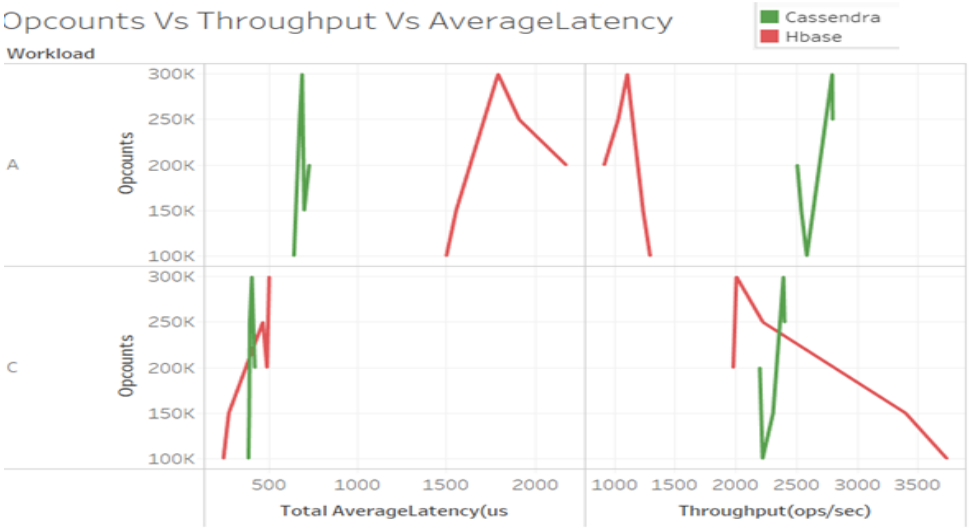
Database	Workload	Opcounts	READ Operations	READ Calculation	Read AverageLatency(us)	Update Operations	Update calcluation	Update AverageLatency(us)	Total AverageLatency(us	Throughput(ops/sec)	RunTime(ms)
Hbase	A	100000	49903	49940;498	459.0799	50097	50060;501	1040.244	1499.324	1292.111	77410.67
Hbase	A	150000	75016	74974;747	483.7549	74984	75026;752	1070.502	1554.257	1234.053	121582.3
Hbase	A	200000	100139	100241;10	1095.356	99861	99759;998	1077.155	2172.511	914.0352	220049
Hbase	A	250000	124906	124886;12	840.5199	125094	125114;12	1068.252	1908.772	1029.407	243138
Hbase	A	300000	149764	149594;14	728.2864	150236	150406;15	1061.951	1790.237	1104.891	272190.3
Hbase	C	100000	100000		244.1988				244.1988	3735.207	26905.67
Hbase	C	150000	150000		274.2431				274.2431	3395.991	44183.67
Hbase	C	200000	200000		489.7265				489.7265	1977.066	101232
Hbase	C	250000	250000		466.0258				466.0258	2221.584	119286.7
Hbase	C	300000	300000		501.4755				501.4755	2003.215	153356.3
Cassendra	A	100000	50031	49916;502	378.2946	49969	50084;497	262.642	640.9366	2582.863	38792.33
Cassendra	A	150000	75018	74938;747	411.1452	74981	75062;752	287.1633	698.3085	2537.516	59285
Cassendra	A	200000	100036	99905;100	440.936	99964	100095;99	285.8763	726.8123	2504.138	79961.67
Cassendra	A	250000	125352	125408;12	412.7494	124648	124592;12	260.5076	673.257	2795.541	89528.67
Cassendra	A	300000	150048	149631;15	407.0116	149951	150369;14	280.1482	687.1598	2792.019	107515.7
Cassendra	C	100000	100000		385.0721				385.0721	2218.411	45266
Cassendra	C	150000	150000		389.0678				389.0678	2304.981	65243.67
Cassendra	C	200000	200000		420.3664				420.3664	2195.77	91086
Cassendra	C	250000	250000		392.3873				392.3873	2399.412	104237.3
Cassendra	C	300000	300000		404.0823				404.0823	2390.486	125557.7

8.2.1. Opcount VS Average Latency for read and update Operation Workload A



Analyzing the relationship between Reading and update operation and the Nosql data bases Read Average and update latency of Cassandra less when compared to the HBase. For reading Operation Average latency of HBase fluctuate when operational count increases.
But Cassandra seems to be almost similar over rage op count 50 K to 150 k .and update operation seems to be stable for HBase.

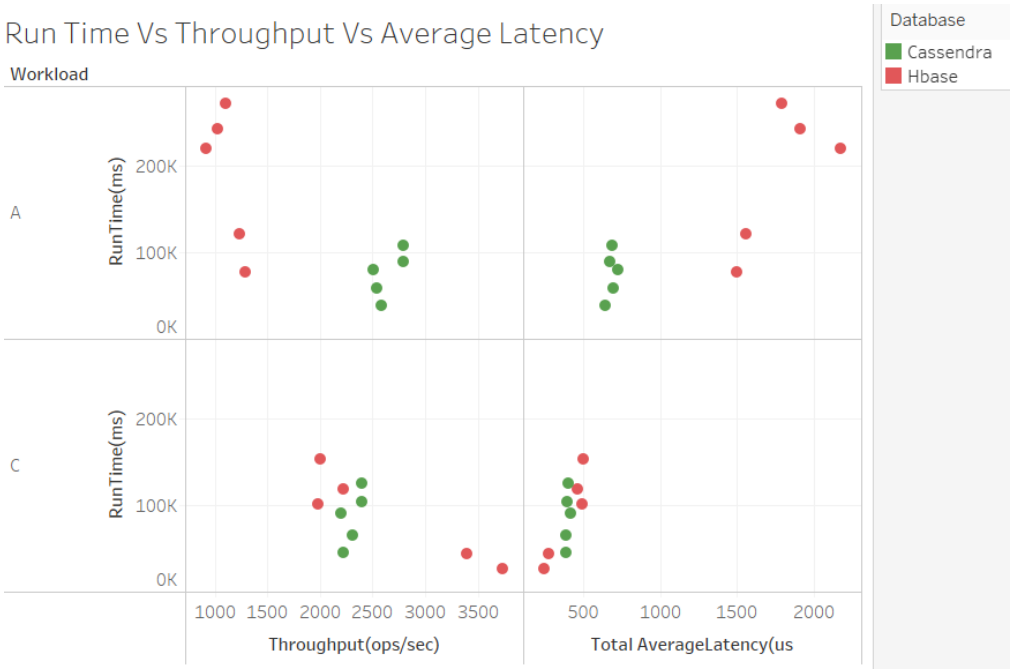
8.2.2. Opcount VS Average Latency Vs Throughput



The figure above analyses the relationship between Operational count average latency and throughput. From the figure, it can be inferred that for 50 % read %50 write Cassandra perform better when compared to HBase. That is, it has a lower average latency and higher throughput. In the case of 100% read, we can see HBase Average latency of reading operation Overlap with Cassandra. We can see that as the average latency is reduced for reading Operation in HBase the throughput is high. From the above research, the report can conclude that HBase performs better with a smaller number of Op Count than that of Cassandra. If the opcount size is more the We should choose Cassandra.

8.2.3. Runtime Average Latency Vs Throughput

In 50% read and 50% write load HBase take more time to run benchmark Test when compared to Cassandra. In the case of 100% read both HBase and Cassandra Overlap the Runtime . when compared to Cassandra, HBase perform well with a lower number of records. even 100% Read HBase perform better than Cassandra.



9 Conclusion

NoSQL Data base is designed to perform well with a large amount of data. So, if your requirement is scalability then it is good to choose NoSQL over RDBMS. Some of the major NoSQL Databases are Cassandra, HBase, Mongodb, etc. This Report compares HBase and Cassandra to know Which one performs better. It is impossible to say that one database is better than other. Both Cassandra and HBase got its own advantages over other.it is better to choose Database according to Requirement. Cassandra is available highly, No Single of failure and reduced administration while on the other hand, HBase is good for faster reading of data in lesser Operational Count.

It is evident that if a system is focusing on performance, the security will be compromising. much financial organization chooses Cassandra over HBase because it provides a better level of security, unlike HBase.

The Report Analyzed that Cassandra performs better than HBase in the case of 50% read 50% write operation .it has less Average latency more throughput and less runtime. Hence in the case of workload, A Cassandra is highly available when compared to HBase.

The HBase perform well with 100 % Read operation than that of cassandra.it is analyzed that workload C HBase overlaps with the performance of Cassandra and more it performs better than that of Cassandra. i.e. is it has less Average latency more throughput and less runtime.

The Report can't conclude that which one is best but, yet we can say that large Operational Count of 50% read and 50% write Cassandra performs well than that of HBase . while 100% Read Operation, it is evident that HBase performs better for a limited amount of Records.

References

Pallas, F., G¨unther, J. & Bermbach, D. (2016), 'Pick your choice in HBase: Security or performance', *in '2016 IEEE International Conference on Big Data (Big Data)'*,p p. 548–554

Ali Hammood, M. S. (2016), 'A comparison of nosql database systems: A study on mongodb, apache HBase, and apache Cassandra', pp. 20 – 23.

Abramova, V., Bernardino, J., Furtado, P. (2014),' Evaluating Cassandra Scalability with YCSB', ., *Springer International Publishing, Cham*, pp. 199–207

Veronika Abramova, Jorge Bernardino, P. F. (2014), 'Which nosql database? a performance overview', *OJDB* 1
URL: https://d-nb.info/1132360862/34

Alexander Pokluda, Wei Sun(2015), 'Benchmarking Failover Characteristics of Large-Scale Data Storage Applications: Cassandra and Voldemort' *University of Waterloo*
URL: http://www.alexanderpokluda.ca/coursework/cs848/CS848%20Project%20Report%20-%20Alexander%20Pokluda%20and%20Wei%20Sun.pdf

Houcine Matallah(2017), 'Experimental comparative study of NoSQL databases: HBASE versus MongoDB by YCSB','*Experimental comparative study of NoSQL databases*'

URL:
https://www.researchgate.net/publication/320127244_Experimental_comparative_study_of_NoSQL_databases_HBASE_versus_MongoDB_by_YCSB

John Klein, Ian Gorton, Neil Ernst, Patrick Donohoe, Kim Pham, Chrisjan Matser(2015),' *Performance Evaluation of NoSQL Databases: A Case Study'*

URL: https://resources.sei.cmu.edu/asset_files/ConferencePaper/2015_021_001_454132.pdf

Heeyoul Kim(2016), Secure Group Communication for Cassandra*, Circuits, Control, Communication, Electricity, Electronics, Energy, System, Signal and Simulation 2016,*

URL:
https://www.researchgate.net/publication/305472306_Secure_Group_Communication_for_Cassandra

DataFlair (1 May 2019), ' Cassandra Tutorials'.
URL: https://data-flair.training/blogs/apache-cassandra-tutorial/

DataFlair (2, May 2019), 'HBase Tutorials'.
URL: https://data-flair.training/blogs/hbase-tutorial/