National College of Ireland

Data Storage and Management

**Project B**

# HBase vs MongoDB

x18110096

Khatik Zainul Abedin

## Abstract:
Rapid growth of unstructured data is the reason why NoSQL databases came into picture. A NoSQL database is that type of database that can handle any sort of unstructured, unpredictable data and messy data whatever the systems requirement is. This paper aims to evaluate the performance of MongoDB and HBase using YCSB (Yahoo! Called the Cloud Serving Benchmark) service which is a standard open source tool mostly used for comparing the performance & evaluation of NoSQL databases.


Keywords: MongoDB, HBase, YCSB, HBase VS MongoDB.

## Introduction:
We have come a long way from the days of spreadsheets, today every other day we create as much data as we did from the beginning of time of until early 2000's. By 2020 the amount of information available will be growing from around 5 zettabytes today to 50 zettabytes (Marr, 2018). As more and more companies debate on adopting big data solutions, one of the thing they discuss is whether to use Hadoop or Spark, NoSQL database or continue using the old traditional method. NoSQL is highly scalable, limited querying, provide better performance and it is design to process unstructured data at a speed 10 times faster than RDBMS. The main purpose of this paper is to perform comparison study of HBase and MongoDB database. YCSB benchmarking tool has been used to check the performance of the databases. Analysis is carried out at different operation counts using CRUD operations. Looking at the famous database ranking website DB-Engines NoSQL databases such as MongoDB ranks $5^{th}$ and HBase ranks $17^{th}$ (Db-engines.com, 2018)

## MongoDB:
MongoDB is an open source document database which is capable of handling humongous amount of data. It works on notion of collection and document i.e. it stores data as JSON-documents and these documents are grouped in collections. "Documents lets you structure data in a way that is efficient for computer to process and natural and easy for humans to read. This is incredibly powerful for developers because they do not have to make their applications accommodate the needs of the database anymore. MongoDB accommodates them, so their applications can store data in a natural way. It also means that they can adapt, adding new data when they need to without be worrying that this simple change is going to break everything. In addition to document model, MongoDB is fundamentally different from legacy databases because it natively knows how to coordinate multiple servers to store data" (Horowitz, 2018).

### Characteristics of MongoDB (Horowitz, 2018) and (Dataflair Team, 2018):

**Fault Tolerance:** A single server failure does not affect the application because fault tolerance is natively built into MongoDB, it is done by keeping redundant copies of same data on different servers (Horowitz, 2018)

**Scalability:** MongoDB seamlessly scales across multiple servers to store and process data. So as the data volumes and performance requirements grow, you can just add more servers instead upgrading to million-dollar mainframes. This is also great for cloud environments where spreading load across lots of machines is by far the best way to scale

**Adhoc queries:** MongoDB supports Adhoc queries. Adhoc queries are the queries which we do not know while we structure the database, these queries are being updated in real time which increases the performance

**Schema Less Database**: MongoDB is flexible when dealing with databases. Here, one collection holds different documents. As it has no schema present, in the same collection it can have multiple fields, content and size different than another document

**Flexible indexing:** The performance of search queries should be high as possible, for that indexing is must. Whenever continuously searches are made in a document, indexing should be done of those fields that match our criteria

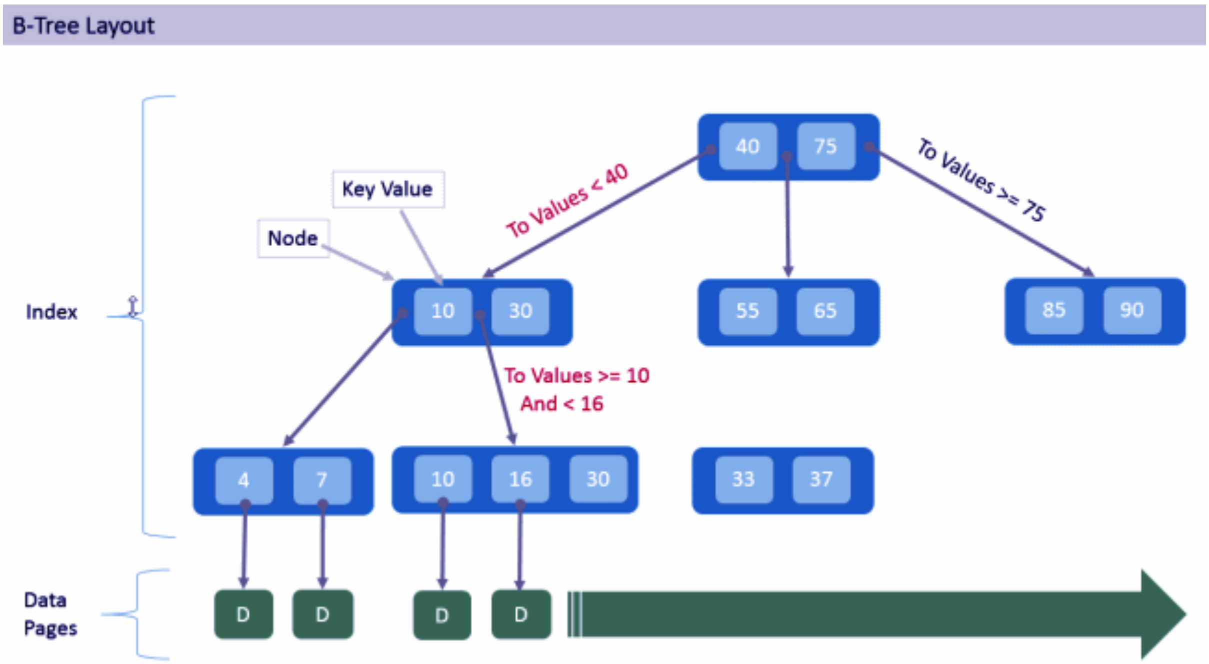



Fig (a) flexible indexing (Dataflair Team, 2018)

**Aggregations in database:**


Fig(b) aggregations in database (Dataflair Team, 2018)

Efficient usability is achieved by aggregation framework in MongoDB. Here, even after performing different task on the group data we can batch process data and get a single result. There are three ways to get an aggregation framework. The aggregation pipeline, map-reduce function, and single purpose aggregation methods

## HBase:

HBase is an open source, distributed Hadoop database which is based on Google big table. It is column oriented database that runs on top of Hadoop Distributed File System. It is written in JAVA programming language. DataFlair. Team (2018)

### HBase characteristics DeZyre (2018) :

**Rebalancing**: HBase provides automatic rebalancing among the clusters.

**Master Slave architecture**: The HBase architecture is based on the Master Slave architecture model.

**Google Big Table:** The HBase database is based on the Google Big Table.

**Replication**: HBase supports the replication of the data across different clusters.

**Linearly scalable**: HBase is linearly scalable.

**Schema-less:** Unlike SQL, HBase does not have schema based tables and has the concept of column families.

**Recovery:** HBase supports recovery from automatic failover.

**Low latency:** Random access across billions of records is supported by HBase with low latency.

**Indexing:** HDFS files are used in HBase to store the indexes which are used for faster lookups and random accesses to the records in the HBase database

## Architecture of MongoDB (mongodb.com, 2018):

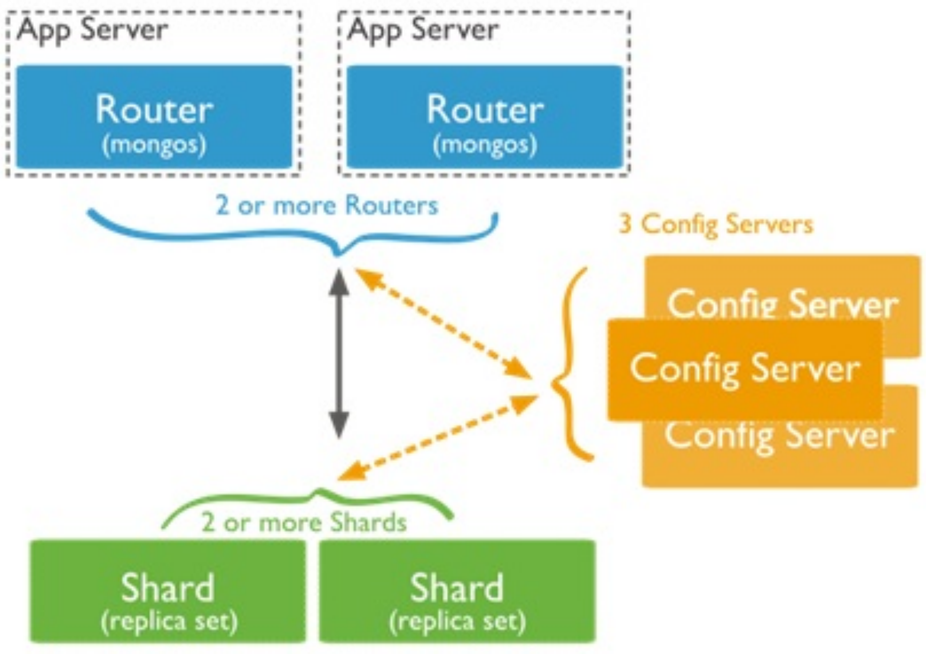



Fig 1. Production Cluster Architecture (mongodb.com, 2018)

In the research paper Khan, S. and Mane, V. (2013) mentions that the architecture of MongoDB is made with three components that are configuration servers,  shard nodes and routing services or mongos

**Shard Nodes.** Shard nodes which has one or more nodes are responsible for storing the data**.** A backup of failures can be achieved when a replicated node of one or more server behaves as a primary replica.

**Configuration Servers:** It stores routing information and meta data of MongoDB which acts as the current primary replica

**Routing Services:** It is also called mongos. Routing services handles various queries which are raised by different clients, the result of this queries will be send to the related node.To increase the its performance MongoDB uses memory mapped files. It supports auto sharding which upscales the performance and its scaling behaviour across multiple nodes.

## Architecture of HBase DeZyre (2018):

Hbase consists of tables that are dynamically distributed to serve the purpose of handling large tables. This feature in HBase is called as Auto Sharding. HBase has an architecture consisting of a single HBase master node and multiple slaves which are called as region servers. Each region server has a dedicated task of serving the regions under it. A client request is always received by the HMaster and assigned to the respective region server.

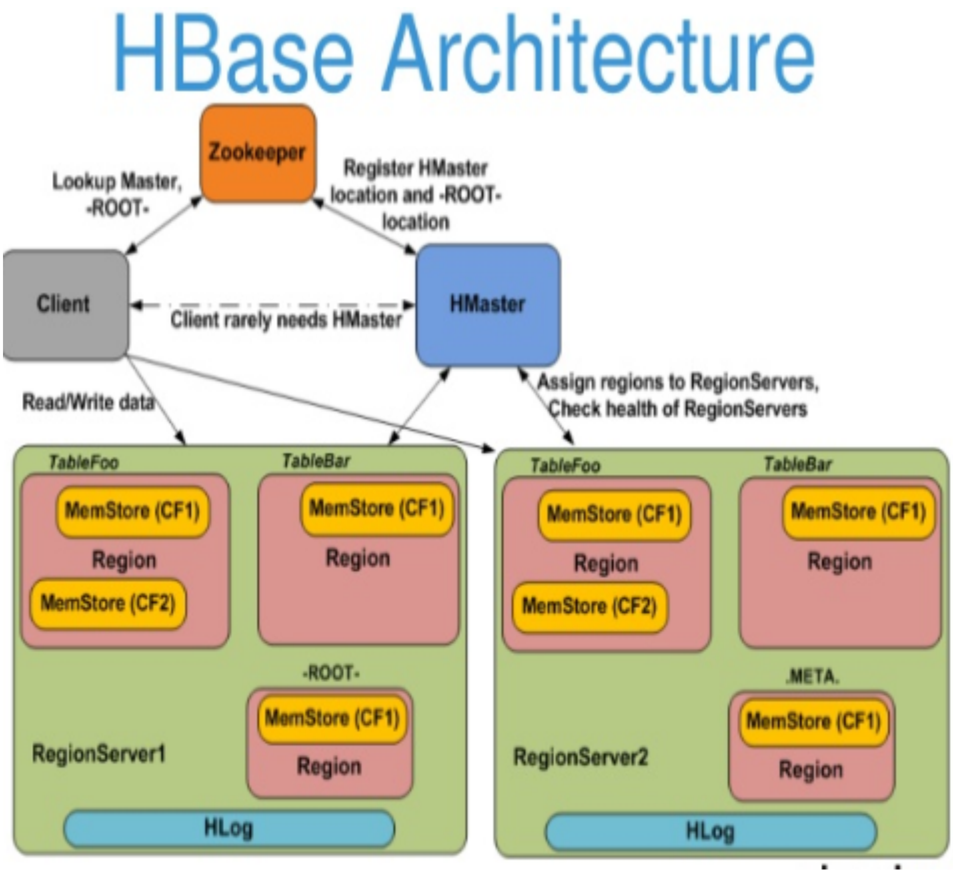



Fig. 2. HBase architecture DeZyre (2018)

The HBase architecture comprises of the three crucial components

**HMaster:** HMaster fulfils the task of balancing the load in the Hadoop cluster by assigning the regions to the regions servers. Alongside, it is also responsible for other tasks such as :

i) Managing the Hadoop cluster
ii) Failover handling
iii) Handling the DDL operations
iv) HMaster is responsible for handling the client requests in terms of metadata operations such as changing the schema etc.

**Region Server:** Region servers are the node responsible for handling the client requests. Region server runs on the Data Node of the distributed file system. Region server includes the following components:

i) Block Cache: Block cache is also called as the read cache. In most of the cases, the block cache stores the read data and when the block cache is full, the most recently used data is removed or eliminated.
ii) MemStore: Memstore is also called as the write cache as it stores the new data that is to be written on the disk.
iii) Write Ahead Log (WAL): Alongside, Hbase also has a Write Ahead Log (WAL) file which is used for data storage that is not to be stored permanently.
iv) HFile: Rows in HBase are stored as the key values on the disk using HFile which is a file storage feature in HBase.

**Zookeeper:** Zookeeper is used in HBase for assigning the regions as well as handling the region server crashes and recovering the region servers. The configuration information for HBase is handled by the zookeeper and centralized monitoring for the Hbase can be done using the zookeeper server. Zookeeper needs to be approached for whenever a client requires to communicate with a region. ZKQuorum is responsible for handling the exceptions and triggering the error messages when the node failure occurs.

Parallelly, Zookeeper maintains a log of all the region servers that are being used in the HBase cluster.

Services provided by Zookeper:

i) Setting up communication between the client and the region servers.
ii) Keeping a track of the server failures.
iii) Keeping the configuration information up to date.
iv) Ephemeral nodes that are used to represent the region servers are provided by Zookeeper.

## Why security is necessary in Nosql databases?

Security for database is the main concern for any IT organization. Security in NOSQL is very weak especially the authentication and encryption.
In a news article about MongoDB it says that " As of Sunday, security researcher and Microsoft developer Niall Merrigan identified more than 27,000 MongoDB databases seized by ransomware. By Tuesday afternoon Pacific Time, an online spreadsheet maintained by Merrigan and fellow security researcher Victor Gevers listed 32,643 victims. The attacks involve hackers who copy data from insecure databases, delete the original, and ask for a ransom of a few hundred dollars worth of Bitcoin to return the stolen data back to the owner." (Theregister.co.uk, 2017)

## Security of MongoDB (Mongodb.com, 2018):

MongoDB features can efficiently defend, detect, and control access to data. It provides various features, such as access control, authentication, encryption, so that your deployments on MongoDB is secured.

**Authentication**. Improving the overall access control to the database, MongoDB provides integration with the external security system including Kerberos, x.509 certificates, LDAP, Windows Active Directory.

**Authorization**. Role-Based Access Controls enable development and operations members to build granular permissions for a user. Role-Based Access Controls enables DevOps teams to configure granular permissions for a user or an application based on the privileges what they want to do in their job. This can be defined in the centre within an LDAP server or in MongoDB. Also, the developer can put restriction on what kind of data should be shown.

**Auditing.** Administrators who can access to MongoDB's audit log can track database operations to check whether it is DDL or DML.

**Encryption**. Here in the data can be encrypted in backups, on the disk, on the network. The main feature of the MongoDB is the protection of data at rest within the encrypted storage engine. By encrypting the database files on the disk, the developers get rid of organisation and performance overhead of external encryption mechanism.

## Security of HBase (Bertozzi, 2018):

**Authentication**: A well secured HBase aims to keep safe from threat such as unauthorized / unauthenticated users and network based threats. But, it does not guard against the authorized users who accidentally delete all the data. The authorization system is established on Simple Authentication and Security Layer (SASL) which is being implemented at RPC level, which supports Kerberos. (Bertozzi, 2018)

**Authorization**: After authentication step user will be given set of rules to follow, in which he will decide whether to perform those steps or not. It is also known as Access Controller Coprocessor or Access Control List (ACL). It gives the user the authority to write read, write, create, admin policies with table, qualifier granularity, family for a specifies user. (Bertozzi, 2018)

**HBase Native Security**: Wire encryption is main security process in HBase between HBase nodes and external clients as well as on different nodes.
For key distribution, ticket raising and authentication, HBase's wire encryption is highly dependent upon Kerberos protocol and architecture. Due to this a dedicated Kerberos service should be there for deployment. (Pallas, Gunther and Bermbach, 2016)

**VPN protected Virtual Private Cloud:**There is another way to deploy the Hbase in cloud is to run the entire cluster with in a Virtual Private Cloud which isolated from other providers infrastructure and it is only available through a devoted network connection.

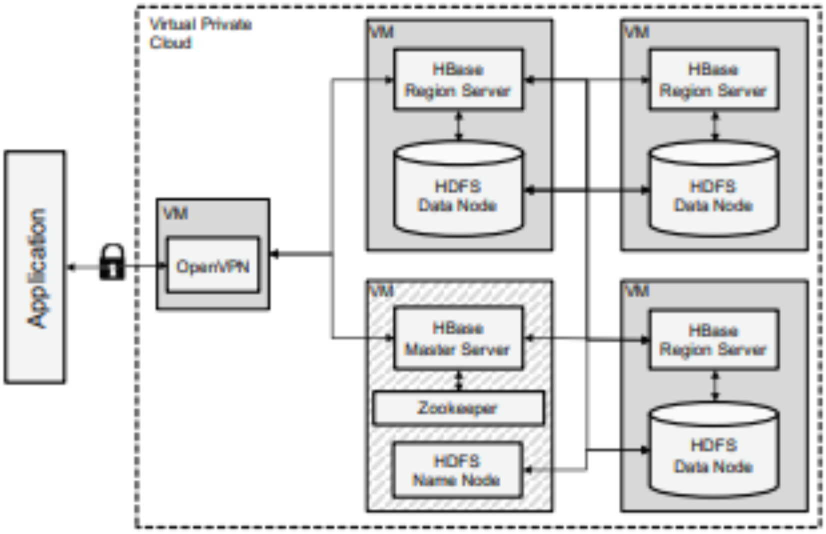



Fig. 3. HBase deployment using a VPN-protected virtual private cloud

As seen in the figure 3, without using HBase native security the cluster traffic between various nodes is secured. This is achieved because it is isolated within Virtual Private Cloud. (Pallas, Gunther and Bermbach, 2016)

## Literature Survey:

M, Houcine (2017) mentions that when carried out the experiment, with 1000 varied operations of read and write operations have been carried out many times on the databases having 600,000 records. The author found out that MongoDB is much more efficient than HBase in read operations such as Read Mostly, Read Only, Read modify write, read mostly and heavy update. And, HBase is more competent than MongoDB while performing write operations such as data load, update mostly and update only.

System Configuration that were being used are follows:
- Ubuntu 14.10, PC 64-bit, Core i5 and 6 GB RAM.
- The results which are obtained from here will be compared with the results obtained by virtual machine with Ubuntu 32-bit, 2 GB of RAM installed in a PC with OS Windows 7 32-bit and 4 GB RAM.
- Average execution time is achieved by running the test three time so that it nullifies the effect of changing th CPU and I/O
- Initializing of the YCSB tool with 6 Workloads.

Also, M, Houcine (2017) states that after benchmarking MongoDB and HBase, the experimental evaluations and results are that MongoDB performs extremely well in read operations, the main reason for this is the register's mapping of MongoDB loaded into memory, which eventually increases the reading performance. The update process in MongoDB is comparatively slow when we compare to number of update achieved. Due to the locking mechanism which happens in MongoDB database, the execution time of update gets increased. HBase performs really well in write operations, be it insertion of records or updating existing records. HBase keeps a track of changes that are made to the database using logs and cache. In HBase data replication is done automatically, it enhances the speed of update process. Before starting a read operation what HBase does is it compares all the files and returns the most recent file which eventually increases the execution time of the reading operations

## Performance Test Plan

Benchmarking is performed using YCSB tool based on the input parameters such as type of operations to be performed and the number of operations counts to be executed.

**System Description:**

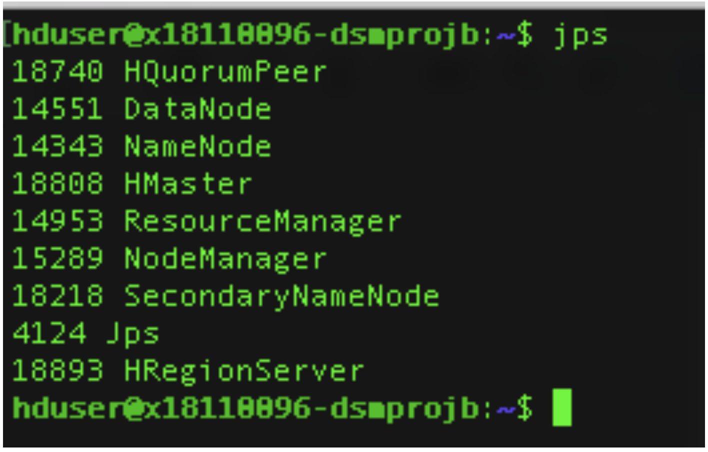
```
*** System restart required ***
Last login: Wed Dec 19 04:10:46 2018 from 192.168.100.119
ubuntu@x18110896-dsmprojb:~$ sudo lshw -short
H/W path         Device      Class        Description
==================================================
                             system       OpenStack Compute
/0                           bus          Motherboard
/0/0                         memory       96KiB BIOS
/0/400                       processor    Intel Core Processor (Haswell, no TSX, IBRS)
/0/401                       processor    Intel Core Processor (Haswell, no TSX, IBRS)
/0/1000                      memory       4GiB System Memory
/0/1000/0                    memory       4GiB DIMM RAM
/0/100                       bridge       440FX - 82441FX PMC [Natoma]
/0/100/1                     bridge       82371SB PIIX3 ISA [Natoma/Triton II]
/0/100/1.1                   storage      82371SB PIIX3 IDE [Natoma/Triton II]
/0/100/1.2                   bus          82371SB PIIX3 USB [Natoma/Triton II]
/0/100/1.2/1     usb1        bus          UHCI Host Controller
/0/100/1.2/1/1               input        QEMU USB Tablet
/0/100/1.3                   bridge       82371AB/EB/MB PIIX4 ACPI
/0/100/2                     display      GD 5446
/0/100/3                     network      Virtio network device
/0/100/3/0       ens3        network      Ethernet interface
/0/100/4                     storage      Virtio block device
/0/100/4/0       /dev/vda    disk         42GB Virtual I/O device
/0/100/4/0/1     /dev/vda1   volume       39GiB EXT4 volume
/0/100/4/0/e     /dev/vda14  volume       4095KiB BIOS Boot partition
/0/100/4/0/f     /dev/vda15  volume       105MiB Windows FAT volume
/0/100/5                     generic      Virtio memory balloon
/0/100/5/0                   generic      Virtual I/O device
ubuntu@x18110896-dsmprojb:~$
```

**Software used to perform benchmarking:**
YCSB version : **YCSB-0.14.0**
HBase version : **HBase-1.4.8**
MongoDB version : **MongoDB-3.2.10**

**In addition to the mentioned software:**

- Java Runtime Environment OpenJDK 8
- Java SE Development Kit OpenJDK 8
- Python - 2.7.11-1
- rsynch
- ssh

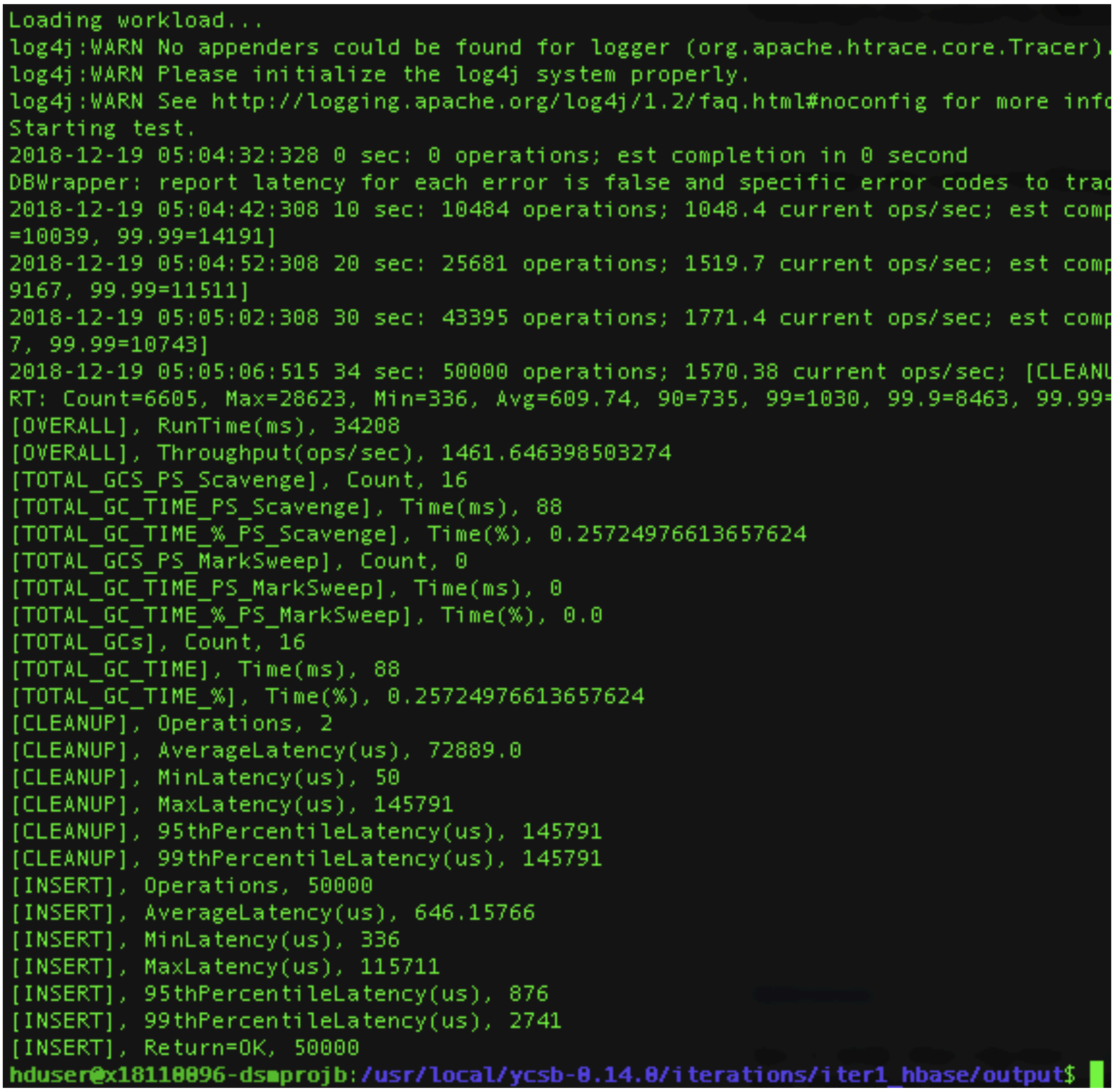
## Once all the installation is done the DFS, Yarn, HBase and Mongo Processes are run:




Workloads (Cooper, 2018):
YCSB contains a set of core workloads that specify a benchmark for the cloud systems
There are different types of workload such as A,B,C,D,E and F. For our analysis we have taken workload A (Update heavy workload) and workload C (Read Only)

| Workload | Name | Operation ratio | Application |
|---|---|---|---|
| Workload A | Update heavy workload | Read 50% - Update 50% | Storing the action logs of a session |
| Workload C | Read only | Read 100% | User profile cache, where the profiles are constructed elsewhere (eg:hadoop) |

Following is the output for **HBase** for 50000 op-count at **workload A:**



```
Loading workload...
log4j:WARN No appenders could be found for logger (org.apache.htrace.core.Tracer).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info
Starting test.
2018-12-19 05:04:32:328 0 sec: 0 operations; est completion in 0 second
DBWrapper: report latency for each error is false and specific error codes to trac
2018-12-19 05:04:42:308 10 sec: 10484 operations; 1048.4 current ops/sec; est comp
=10039, 99.99=14191]
2018-12-19 05:04:52:308 20 sec: 25681 operations; 1519.7 current ops/sec; est comp
9167, 99.99=11511]
2018-12-19 05:05:02:308 30 sec: 43395 operations; 1771.4 current ops/sec; est comp
7, 99.99=10743]
2018-12-19 05:05:06:515 34 sec: 50000 operations; 1570.38 current ops/sec; [CLEANU
RT: Count=6605, Max=28623, Min=336, Avg=609.74, 90=735, 99=1030, 99.9=8463, 99.99=
[OVERALL], RunTime(ms), 34208
[OVERALL], Throughput(ops/sec), 1461.646398503274
[TOTAL_GCS_PS_Scavenge], Count, 16
[TOTAL_GC_TIME_PS_Scavenge], Time(ms), 88
[TOTAL_GC_TIME_%_PS_Scavenge], Time(%), 0.25724976613657624
[TOTAL_GCS_PS_MarkSweep], Count, 0
[TOTAL_GC_TIME_PS_MarkSweep], Time(ms), 0
[TOTAL_GC_TIME_%_PS_MarkSweep], Time(%), 0.0
[TOTAL_GCs], Count, 16
[TOTAL_GC_TIME], Time(ms), 88
[TOTAL_GC_TIME_%], Time(%), 0.25724976613657624
[CLEANUP], Operations, 2
[CLEANUP], AverageLatency(us), 72889.0
[CLEANUP], MinLatency(us), 50
[CLEANUP], MaxLatency(us), 145791
[CLEANUP], 95thPercentileLatency(us), 145791
[CLEANUP], 99thPercentileLatency(us), 145791
[INSERT], Operations, 50000
[INSERT], AverageLatency(us), 646.15766
[INSERT], MinLatency(us), 336
[INSERT], MaxLatency(us), 115711
[INSERT], 95thPercentileLatency(us), 876
[INSERT], 99thPercentileLatency(us), 2741
[INSERT], Return=OK, 50000
hduser@x18110096-dsmprojb:/usr/local/ycsb-0.14.0/iterations/iter1_hbase/output$
```

Following is the output for **MongoDB** for 50000 op-count at **workload A:**

```
[INSERT], Return=OK, 50000
java -cp /usr/local/ycsb-0.14.0/mongodb-binding/conf:/usr/local/ycsb-0.14
4.jar:/usr/local/ycsb-0.14.0/lib/core-0.14.0.jar:/usr/local/ycsb-0.14.0/l
sb-0.14.0/mongodb-binding/lib/logback-classic-1.1.2.jar:/usr/local/ycsb-0
.25.jar:/usr/local/ycsb-0.14.0/mongodb-binding/lib/mongodb-binding-0.14.0
ing/lib/mongodb-async-driver-2.0.1.jar:/usr/local/ycsb-0.14.0/mongodb-bin
oads/workloada -p recordcount=50000 -load
Command line: -db com.yahoo.ycsb.db.MongoDbClient -s -P workloads/workloa
YCSB Client 0.14.0

Loading workload...
Starting test.
2018-12-19 00:51:17:819 0 sec: 0 operations; est completion in 0 second
mongo client connection created with mongodb://localhost:27017/ycsb?w=1
DBWrapper: report latency for each error is false and specific error code
2018-12-19 00:51:27:756 10 sec: 24305 operations; 2430.5 current ops/sec;
=4207, 99.99=12703]
2018-12-19 00:51:31:217 13 sec: 50000 operations; 7424.15 current ops/sec
=25695, Max=14863, Min=86, Avg=131.1, 90=160, 99=253, 99.9=1043, 99.99=69
[OVERALL], RunTime(ms), 13460
[OVERALL], Throughput(ops/sec), 3714.7102526002973
[TOTAL_GCS_PS_Scavenge], Count, 9
[TOTAL_GC_TIME_PS_Scavenge], Time(ms), 57
[TOTAL_GC_TIME_%_PS_Scavenge], Time(%), 0.4234769687964339
[TOTAL_GCS_PS_MarkSweep], Count, 0
[TOTAL_GC_TIME_PS_MarkSweep], Time(ms), 0
[TOTAL_GC_TIME_%_PS_MarkSweep], Time(%), 0.0
[TOTAL_GCs], Count, 9
[TOTAL_GC_TIME], Time(ms), 57
[TOTAL_GC_TIME_%], Time(%), 0.4234769687964339
[CLEANUP], Operations, 1
[CLEANUP], AverageLatency(us), 2137.0
[CLEANUP], MinLatency(us), 2136
[CLEANUP], MaxLatency(us), 2137
[CLEANUP], 95thPercentileLatency(us), 2137
[CLEANUP], 99thPercentileLatency(us), 2137
[INSERT], Operations, 50000
[INSERT], AverageLatency(us), 252.2729
[INSERT], MinLatency(us), 86
[INSERT], MaxLatency(us), 3637247
[INSERT], 95thPercentileLatency(us), 290
[INSERT], 99thPercentileLatency(us), 473
[INSERT], Return=OK, 50000
hduser@x18110096-dsmprojb:/usr/local/ycsb-0.14.0/output_p$
```

## Average workload calculations:

**Results for MongoDB at Workload A**

**TEST 1:**

| | MongoDB | | Workload A | |
|---|---|---|---|---|
| | 50000 | 100000 | 150000 | 200000 |
| Throughput | 5059.19 | 5275.09 | 4332.63 | 5037.52 |
| Read Operation | 25116 | 49812 | 75151 | 99948 |
| Average Read Latency | 162.84 | 162.01 | 158.48 | 162.75 |
| Update Operation | 24884 | 50188 | 74849 | 100052 |
| Avg Update latency | 199.11 | 197.63 | 286.96 | 221.19 |
| Insert Operation | 50000 | 100000 | 150000 | 200000 |
| Average Insert Latency | 218.3 | 180.23 | 154.24 | 147.23 |

**Test 2:**

| | MongoDB | | Workload A | |
|---|---|---|---|---|
| | 50000 | 100000 | 150000 | 200000 |
| Throughput | 5320.28 | 4733.50 | 5320.28 | 5880.62 |
| Read Operation | 75075 | 49925 | 75075 | 100163 |
| Average Read Latency | 128.0 | 136.11 | 128 | 129.24 |
| Update Operation | 74925 | 50075 | 74925 | 99837 |
| Avg Update latency | 238.24 | 267.22 | 238.24 | 199.49 |
| Insert Operation | 150000 | 100000 | 150000 | 200000 |
| Average Insert Latency | 176.16 | 264.78 | 176.16 | 219.70 |

**Average result of Test 1 and Test 2:**

| | MongoDB | | Workload A | |
|---|---|---|---|---|
| | 50000 | 100000 | 150000 | 200000 |
| Throughput | 5057.155 | 5004.295 | 4826.455 | 5459.07 |
| Read Operation | 13782.56 | 49868.5 | 75113 | 100055.5 |
| Average Read Latency | 162.92 | 149.06 | 143.24 | 145.995 |
| Update Operation | 25438 | 50131.5 | 74887 | 99944.5 |
| Avg Update latency | 199.615 | 232.425 | 262.6 | 210.34 |
| Insert Operation | 50000 | 100000 | 150000 | 200000 |
| Average Insert Latency | 209.75 | 222.505 | 165.2 | 183.465 |

## Results for MongoDB at Workload C

**Test 1:**

| | MongoDB | | Workload C | |
|---|---|---|---|---|
| | 50000 | 100000 | 150000 | 200000 |
| Throughput | 7697.044 | 6955.07 | 9106.91 | 8909.87 |
| Read Operation | 50000 | 100000 | 150000 | 200000 |
| Average Read Latency | 116.85 | 135.21 | 103.09 | 106.61 |
| Insert Operation | 50000 | 100000 | 150000 | 200000 |
| Avg Insert Latency | 411.28 | 236.25 | 207.25 | 206.87 |

**Test 2:**

| | MongoDB | | Workload C | |
|---|---|---|---|---|
| | 50000 | 100000 | 150000 | 200000 |
| Throughput | 5487.26 | 6518.90 | 6803.02 | 7395.62 |
| Read Operation | 50000 | 100000 | 150000 | 200000 |
| Average Read Latency | 164.71 | 143.73 | 139.09 | 129.75 |
| Insert Operation | 50000 | 100000 | 150000 | 200000 |
| Avg Insert Latency | 411.28 | 236.25 | 207.25 | 206.87 |

**Average Result:**

| | MongoDB | | Workload C | |
|---|---|---|---|---|
| | 50000 | 100000 | 150000 | 200000 |
| Throughput | 6592.152 | 6736.985 | 7954.965 | 8152.745 |
| Read Operation | 50000 | 100000 | 150000 | 200000 |
| Avg Read Latency | 140.78 | 139.47 | 121.09 | 118.18 |
| Insert Operation | 50000 | 100000 | 150000 | 200000 |
| Average Insert Latency | 290.395 | 202.68 | 179.135 | 180.585 |

**Results for HBase at Workload A**

**Test 1:**

| | HBase | | Workload A | |
|---|---|---|---|---|
| | 50000 | 100000 | 150000 | 200000 |
| Throughput | 2258.4 | 2571 | 2675.75 | 2729.74 |
| Read Operation | 10108 | 20640 | 31880 | 43272 |
| Average Read Latency | 244.17 | 219.9 | 222.86 | 224.02 |
| Update Operation | 25000 | 50287 | 74748 | 9992 |
| Avg Update latency | 592.20 | 538.16 | 519.34 | 512 |
| Insert Operation | 50000 | 100000 | 150000 | 200000 |
| Average Insert Latency | 646.15 | 575.51 | 652.0 | 670.31 |

**Test 2:**

| | HBase | | Workload A | |
|---|---|---|---|---|
| | 50000 | 100000 | 150000 | 200000 |
| Throughput | 2102.07 | 1407.91 | 1064.58 | 878.13 |
| Read Operation | 25025 | 50006 | 74854 | 99952 |
| Average Read Latency | 291.42 | 679.66 | 1016.68 | 1545.57 |
| Update Operation | 24975 | 49994 | 75146 | 100048 |
| Avg Update latency | 555.13 | 675.41 | 791.72 | 706.52 |
| Insert Operation | 50000 | 100000 | 150000 | 200000 |
| Average Insert Latency | 566.45 | 528.79 | 557.08 | 579.08 |

**Average result:**

| Average result | HBase | | Workload A | |
|---|---|---|---|---|
| | 50000 | 100000 | 150000 | 200000 |
| Throughput | 2180.235 | 1989.455 | 1870.165 | 1803.935 |
| Read Operation | 17566.5 | 35323 | 53367 | 71612 |
| Average Read Latency | 267.795 | 449.78 | 619.77 | 884.795 |
| Update Operation | 24987.5 | 50140.5 | 74947 | 55020 |
| Average Update latency | 573.665 | 606.785 | 655.53 | 609.26 |
| Insert Operation | 50000 | 100000 | 150000 | 200000 |
| Average Insert Latency | 606.3 | 552.15 | 604.54 | 624.695 |

# Results for HBase at Workload C

**Test 1:**

| | HBASE | | Workload C | |
|---|---|---|---|---|
| | 50000 | 100000 | 150000 | 200000 |
| Throughput | 3329.78 | 2211.99 | 1478.7 | 3336.83 |
| Read Operation | 50000 | 100000 | 150000 | 200000 |
| Average Read Latency | 266.3 | 423.14 | 662.427 | 286.84 |
| Insert Operation | 50000 | 100000 | 150000 | 200000 |
| Average Insert Latency | 545.09 | 582.09 | 525.29 | 519.27 |

**Test 2:**

| | HBASE | | Workload C | |
|---|---|---|---|---|
| | 50000 | 100000 | 150000 | 200000 |
| Throughput | 3001 | 2221.9 | 1563 | 3398.8 |
| Read Operation | 50000 | 100000 | 150000 | 200000 |
| Average Read Latency | 201.2 | 430.4 | 669.6 | 270.23 |
| Insert Operation | 50000 | 100000 | 150000 | 200000 |
| Average Insert Latency | 512.12 | 578.32 | 515.23 | 521.12 |

**Average Result for HBase at Workload C:**

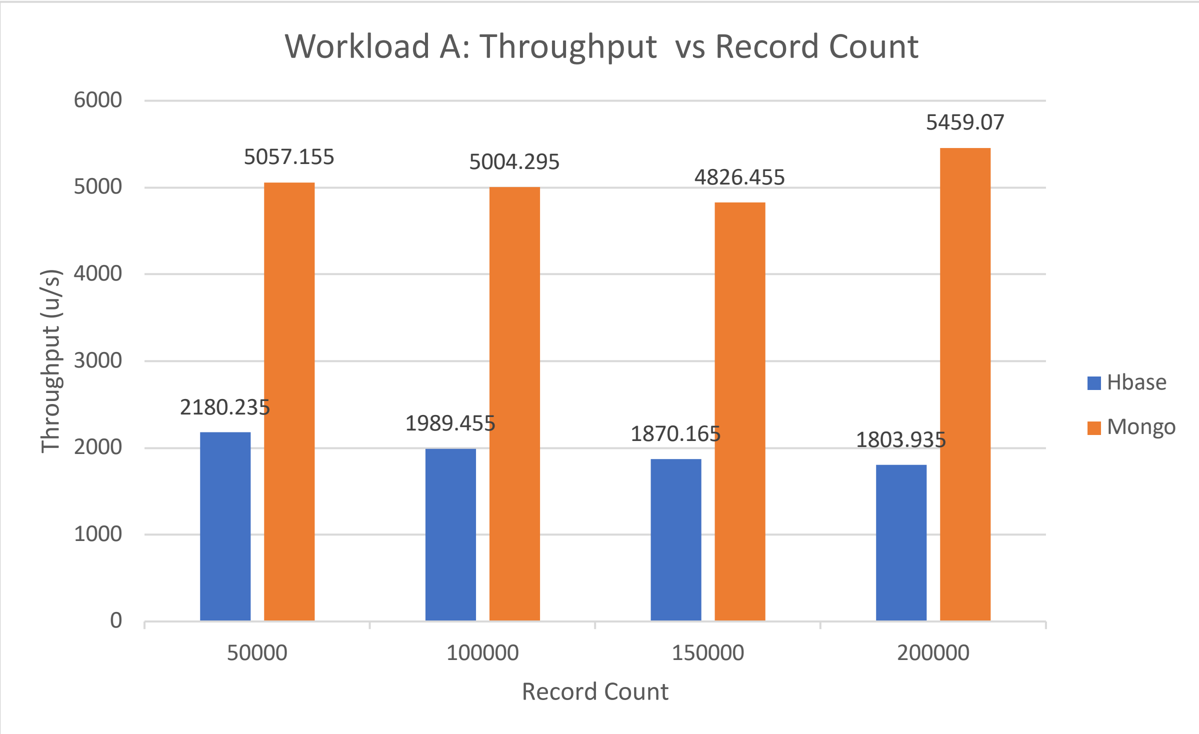
| | HBASE | | Workload C | |
|---|---|---|---|---|
| | 50000 | 100000 | 150000 | 200000 |
| Throughput | 3165.39 | 2216.945 | 1520.85 | 3367.815 |
| Read Operation | 50000 | 100000 | 150000 | 200000 |
| Average Read Latency | 233.75 | 426.77 | 666.0135 | 278.535 |
| Insert Operation | 50000 | 100000 | 150000 | 200000 |
| Average Insert Latency | 528.605 | 580.205 | 520.26 | 520.195 |

## Result:

The graphs below illustrate the result of performances of both the database MongoDB and HBase.

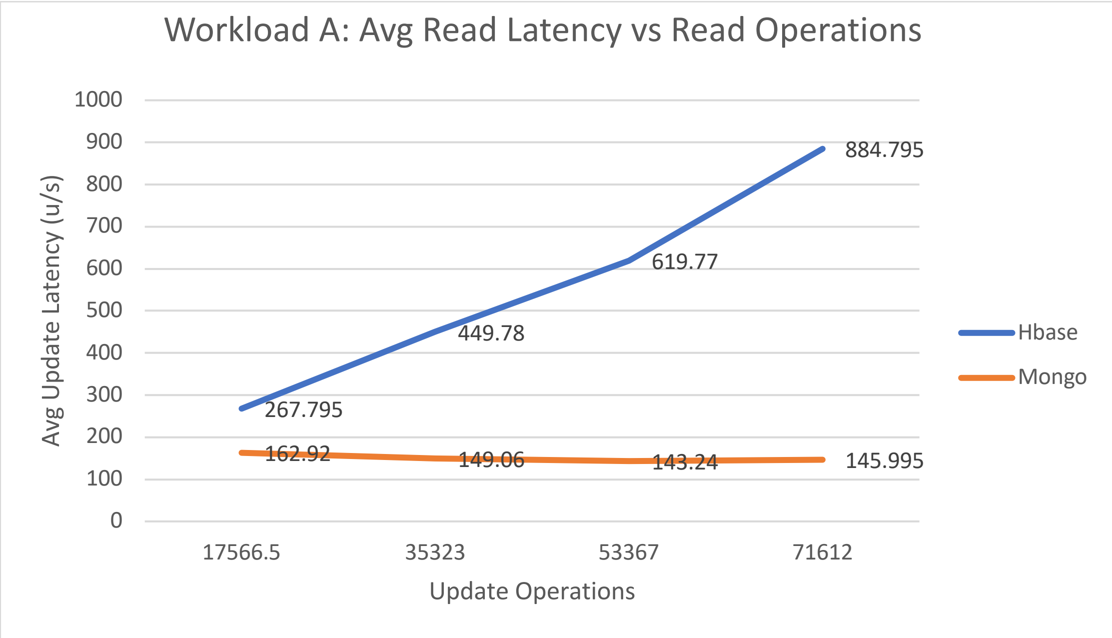
**TEST FOR WORKLOAD A**

Comparing total throughput with Record count



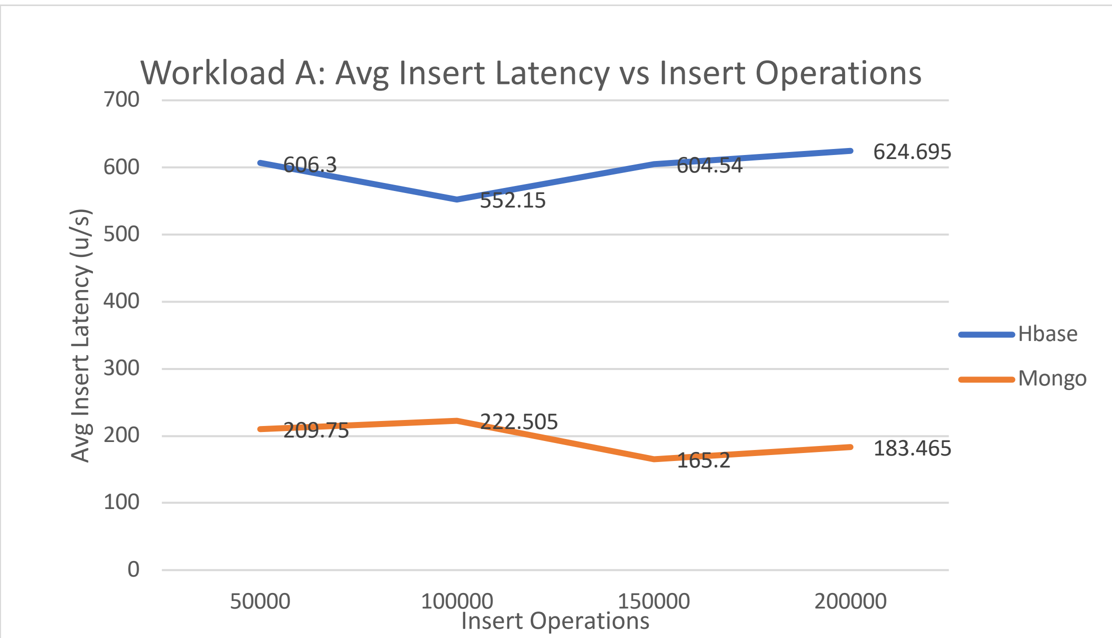

Throughput comparison gives us an idea about the performance of both the databases, which one has a upper hand in terms operations/second. In the above graph, performance based testing on total throughput with Read operation was analysed .For chosen Workload A for 50% read-50% update, low operation counts for HBase has been recorded as compared to that of Mongo DB. It is clearly evident that at 200000 record count MongoDB has shown a peak performance.

Testing and comparing Average Read latency with Read Operation




This analysis is comparing the performance of Average read latency with Read operation. It is clearly evident from the graph that HBase has more average latency than MongoDB. Also, HBase is increasing in average latency with increase in workload. Whereas MongoDB shows a stagnant performance . Thus it can be said that MongoDB performed better .

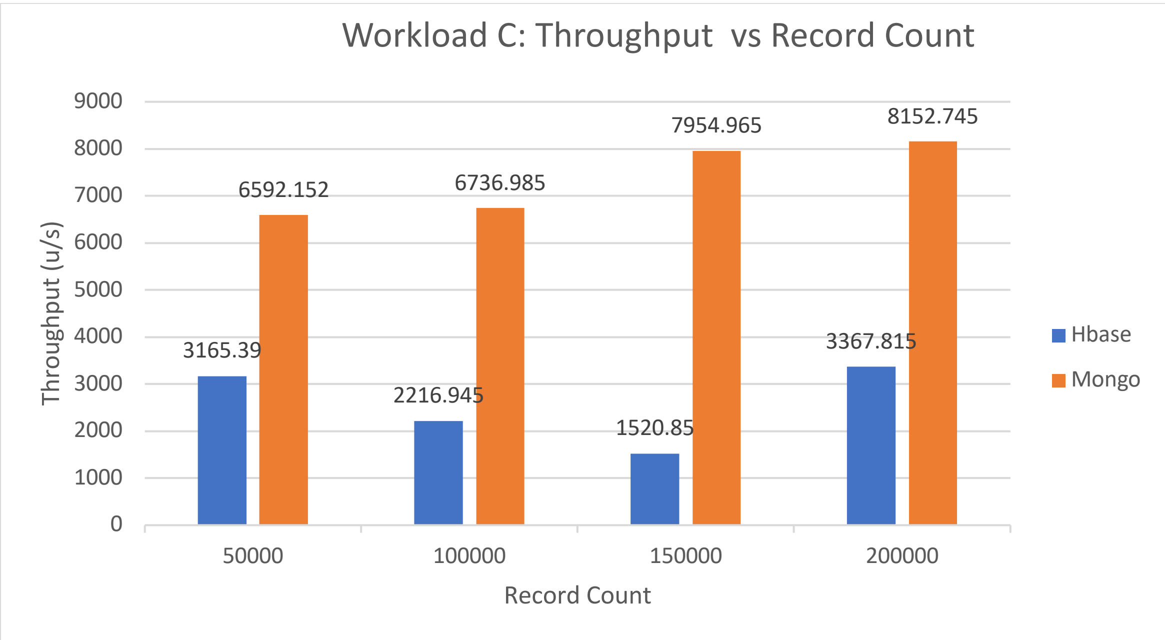
Testing Average Insert Latency with Insert Operation



This analysis is comparing the performance of Average Insert latency with Insert operation. It is clearly evident from the graph that HBase has more average latency than MongoDB . also HBase is increasing in average latency with increase in workload . Where as MongoDB has shown a better performance at insert operation . Thus it can be said that MongoDB performed better .

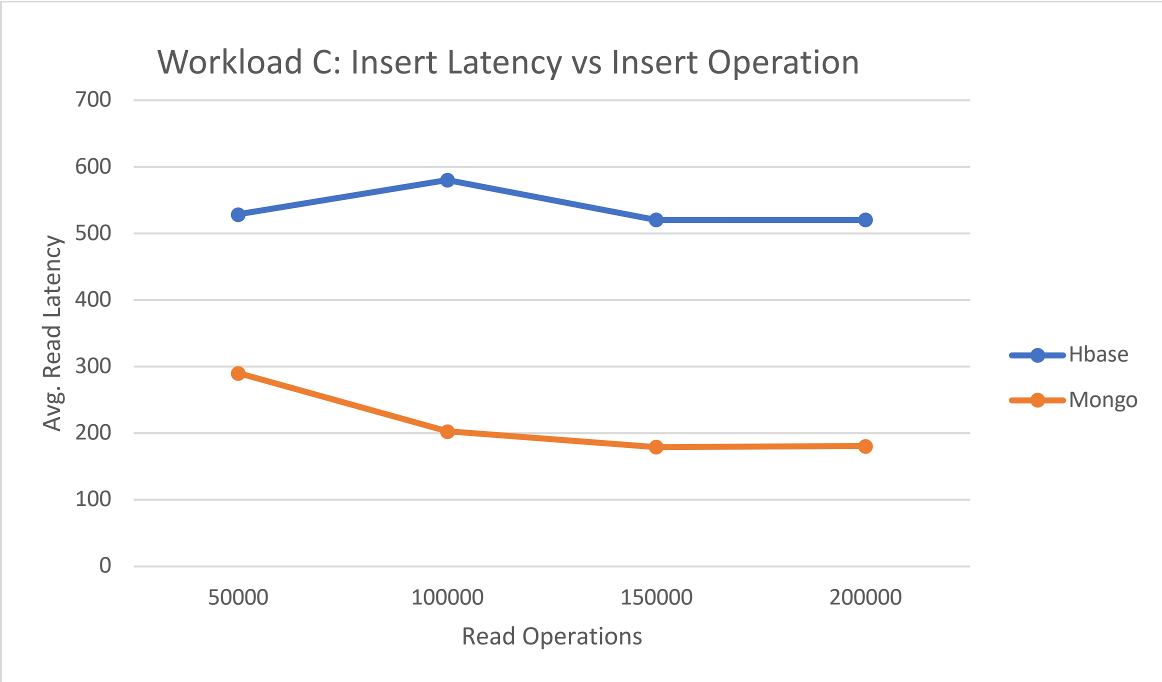
## TEST FOR WORKLOAD C

Comparison analysis of Total workload VS Record Count



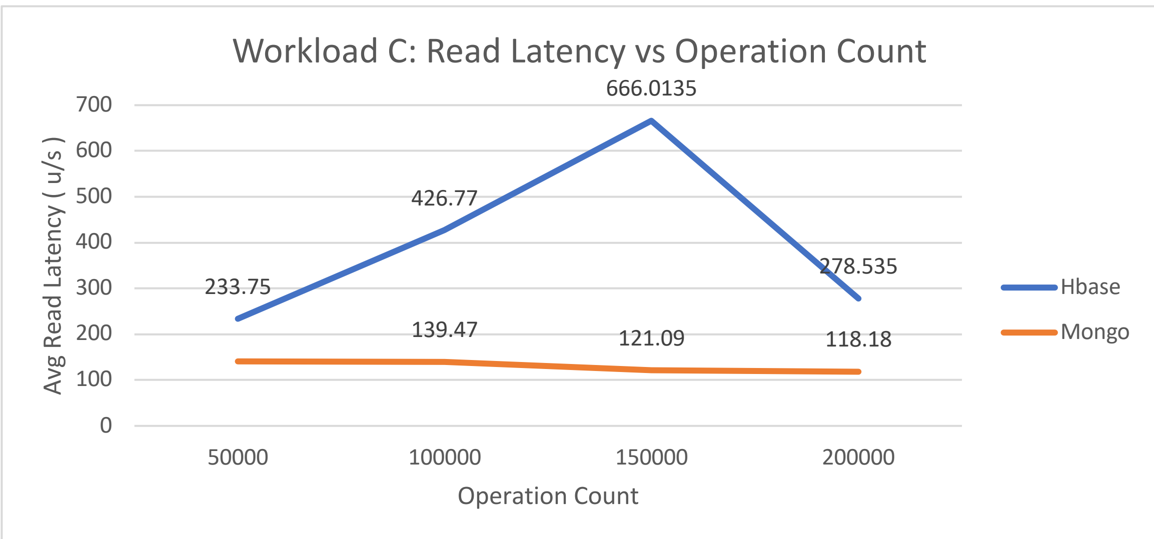

In the above graph performance based testing on total throughput with Read operation was analysed .For chosen Workload C which by default is 100 percent read has shown low operation counts for HBase as compared to that of Mongo DB .At 200,00 MongoDB has shown a peak performance . The graph clearly depicts that MongoDB has performed better compared to HBase at workload C for Throughput vs Record count.

Performance Test of Insert Latency VS Insert Operation




From the above graph it is quite clear that MongoDB at Insert operation is showing less latency as compared to HBase . It also shows a constant performance for increase in workload whereas HBase has shown greater average latency. Thus it can be concluded MongoDB has performed better for insert operation.

Performance Test of Read Latency VS Insert Operation



This analysis is comparing the performance of Average read latency with Read operation for Workload C which is default at 100 percent read. It is clearly evident from the graph that HBase has more average latency than MongoDB HBase shows a peak performance at 666.013 whereas MongoDB has shown a constant performance . Thus it can be said that MongoDB performed better .

## Conclusion:

After performing the experiment, it can be concluded that at Workload C i.ie. 100% Read by default, MongoDB has an upper hand than HBase. While comparing throughput vs record count it can be clearly seen that at 200,000 count MongoDB is performing better than HBase. Likewise, in Insert Latency vs Insert Operation and in read latency vs operation count MongoDB is better performer than HBase. Furthermore, at Workload A i.e. 50% read – 50% write when comparing throughput with record counts MongoDB has an upper hand over HBase. Also, when we compare average read latency with read operations, as the load increases HBase's average read latency increase. Whereas in MongoDB average read latency slightly changes. Clearly MongoDB outperforms HBase till 200,000 count. Also, when comparing with read latency with operation count MongoDB performs well as compare to HBase.

## References:


Marr, B. (2018). *What is Big Data? A super simple explanation for everyone*. [online] Bernard Marr. Available at: https://www.bernardmarr.com/default.asp?contentID=766 [Accessed 19 Dec. 2018].

Db-engines.com (2018) *DB-Engines - Knowledge Base of Relational and NoSQL Database Management Systems*. [online] Available at: https://db-engines.com/en/ [Accessed 19 Dec. 2018].

Dataflair Team (2018) *Learn MongoDB Features | MongoDB Tutorial for Beginners – DataFlair Team*. [online] DataFlair Team. Available at: https://data-flair.training/blogs/mongodb-features/ [Accessed 19 Dec. 2018].

Horowitz, E. (2018) MongoDB in 5 Minutes with Eliot Horowitz. Available at: https://www.youtube.com/watch?v=EE8ZTQxa0AM [Accessed at 14 December 2018]

DataFlair Team (2018) *HBase Tutorial For Beginners 2018 | Learn Apache HBase – DataFlair Team*. [online] Available at: https://data-flair.training/blogs/hbase-tutorial/ [Accessed 19 Dec. 2018].

DeZyre (2018). *Overview of HBase Architecture and its Components*. [online] Available at: https://www.dezyre.com/article/overview-of-hbase-architecture-and-its-components/295 [Accessed 19 Dec. 2018].

mongodb.com. (2018). *Production Cluster Architecture — MongoDB Manual*. [online] Available at: https://docs.mongodb.com/v3.0/core/sharded-cluster-architectures-production/ [Accessed 19 Dec. 2018].

Theregister.co.uk. (2017). *How to secure MongoDB – because it isn't by default and thousands of DBs are being hacked*. [online] Available at: https://www.theregister.co.uk/2017/01/11/mongodb_ransomware_followup/ [Accessed 18 Dec. 2018].

Mongodb.com. (2018). *MongoDB Architecture Guide*. [online] Available at: https://www.mongodb.com/collateral/mongodb-architecture-guide [Accessed 18 Dec. 2018].

Bertozzi, M. (2018). *How-to: Enable User Authentication and Authorization in Apache HBase - Cloudera Engineering Blog*. [online] Cloudera Engineering Blog. Available at: https://blog.cloudera.com/blog/2012/09/understanding-user-authentication-and-authorization-in-apache-hbase/ [Accessed 19 Dec. 2018].

Pallas, F., Gunther, J. and Bermbach, D. (2016). Pick your choice in HBase: Security or performance. *2016 IEEE International Conference on Big Data (Big Data)*.

Khan, S. and Mane, V. (2013) 'SQL support over MongoDB using metadata', *International Journal of Scientific and Research Publications,* 3(10), pp. 1-5.

M, Houcine (2017) 'Experimental comparative study of NoSQL databases: HBASE versus MongoDB by YCSB', *International Journal of Computer Systems Science & Engineering,* vol(*32*).

Abramova, V., Bernardino, J. and Furtado, P.(2014) 'Experimental evaluation of NoSQL Databases', *International Journal of Database Management Systems ( IJDMS )*, 6(3).

Cooper, B. (2018). *brianfrankcooper/YCSB*. [online] GitHub. Available at: https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads [Accessed 19 Dec. 2018].