Benchmarking NoSQL Databases

Team Members

1. Sarat Chandra Vysyaraju (scv2114)
2. Venciya Antony Bupal George (va2325)
3. Nitesh Chauhan (nc2663)

Introduction

In the development of software systems, databases play a key role in establishing efficient information storage and retrieval systems. In order to support the various challenges faced today, these systems have evolved from being isolated systems that are confined to generating and storing data to very dynamic entities hosted on the cloud and distributed environments. As the amount of data increases, the interest in NoSQL continues to rise. Many international internet companies such as Google, Amazon and Facebook have their own NoSQL solutions designed to process the exploding amount of data being generated. The main aim of our project is to use YCSB (Yahoo 's Cloud Serving Benchmark) tool to compare the relative performance of various NoSQL database management systems and determine which NoSQL database is appropriate for a given workload. This would be accomplished by using a set of evaluation metrics on different workloads.

What did we learn ?

The project exposed us to the YCSB tool and the basic working architecture of a benchmarking system. It taught us about a few important aspects related to the connections between the nodes and services of the NoSQL databases during their installation. It also taught us about how to use drivers to connect YCSB to the databases and develop custom workloads to observe how the database behaves under different possible scenarios. On analysis of the results obtained after testing the workloads on the data stored in the different databases, we observe that MongoDB performs better than HBase and Cassandra when it comes to inserts and updates. Cassandra performs almost as good as MongoDB in the case of 100% reads, although it does seem to suffer after it crosses a certain throughput threshold.

What Challenges did we face ?

Initially, we had provisioned 10GB of storage space for the AWS instance, but later while creating certain workloads we realized that it was insufficient and hence we had to reprovision more space. As a result of reprovisioning we had to reinstall the databases and reload data into them. Another key challenge that we faced was that while trying to install Hadoop which was a prerequisite for installing HBase, the connection between the namenode and the secondary node was failing again and again. Initially we couldn’t figure out the issue as a similar installation worked well in a local system, however we figured out that as we were doing ssh to connect to the AWS instance and through this when we were trying to connect the name and secondary nodes to start the hdfs (Hadoop distributed file system), the permission was being denied and then we had to generate the private and public ssh keys to enable appropriate permissions to each of the constituent nodes and services of Hadoop to connect. This eventually solved the issue.

We also came across the heap space allocation issue when all three databases were run in parallel. This is because loading in 1 million data points into all three database simultaneously consumed a lot of RAM. Also, we did come across a few challenges while making an amazon machine image.

Project Outline

We started out our project by setting up an Ubuntu 14.0.4 Virtual Machine on Amazon Web Services so that all members of the group could have access to the system remotely and can work independently. We then shortlisted the NoSQL databases that we intended to compare namely

1. Cassandra 3.5
2. HBase 2.7.1
3. MongoDB 3.2.6

We then came up with the following workloads

1. 50% Reads & 50% Updates.
2. 95% Reads (heavy read) & 5% Updates.
3. 100% Reads (Read Intensive).
4. 95% Reads (heavy read) & 5% Insert.
5. 95% Updates (heavy write) & 5% Reads.

This is the list of the various steps/tasks involved in the project, which we would describe in detail in subsequent sections:

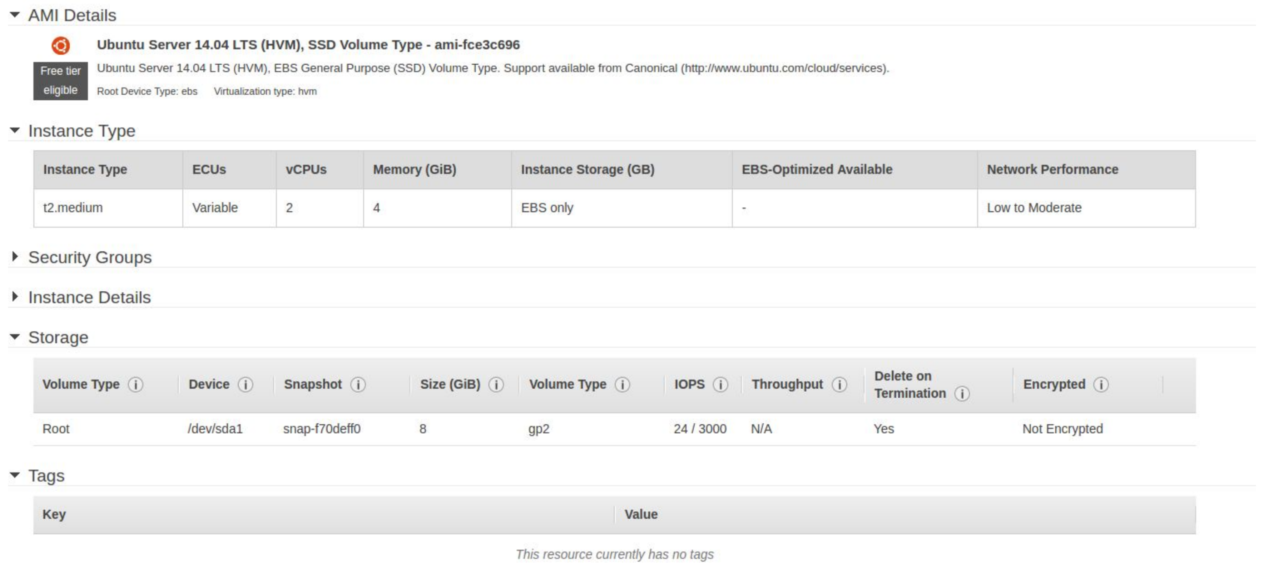
1. Installing the YCSB (Yahoo ’s Cloud Serving Benchmark) tool.
2. Installing Cassandra in Standalone mode.
3. Installing HBase in Standalone mode.
4. Installing MongoDB in Standalone mode.
5. Establishing drivers between the database and the YCSB Client.
6. Loading one million data points of 1KB each into the Databases.
7. Building workloads by specifying the reads and writes percentiles.
8. Running different workloads on the data that contains one million data points.
9. Evaluating the performance of the databases under different workloads.

Thus, the project would involve establishing the above listed NoSQL databases locally and then populating an near identical dataset in each of the databases. The data would then be tested over the various workloads using the above listed parameters. Each of the workloads would simulate specific tasks in NoSQL databases having implications to various practical usage of databases. Thus we will observe the various latency and throughput parameters for each of the above cases to analyse and find the optimal use of databases for different tasks.

Provisioning Machine for Benchmarking

We decided to provision an Amazon EC2 instance for benchmarking all the databases. This will ensure that there is a platform that is consistent when benchmarking all the databases. We decided to proceed with a medium tier EC2 instance. The configuration of the EC2 instance is as follows:

2 cores of CPU.
4 GB of Memory.
8 GB of General Purpose SSD.
10 GB External General Purpose SSD for installations.
Ubuntu Trusty Image.



AMI Details

Ubuntu Server 14.04 LTS (HVM), SSD Volume Type - ami-fce3c696

Free tier eligible

Ubuntu Server 14.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).

Root Device Type: ebs Virtualization type: hvm

Instance Type

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
t2.medium	Variable	2	4	EBS only	-	Low to Moderate

Security Groups

Instance Details

Storage

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput	Delete on Termination	Encrypted
Root	/dev/sda1	snap-f70deff0	8	gp2	24 / 3000	N/A	Yes	Not Encrypted

Tags

Key	Value

This resource currently has no tags

Setting up YCSB

We downloaded the latest release of YCSB:

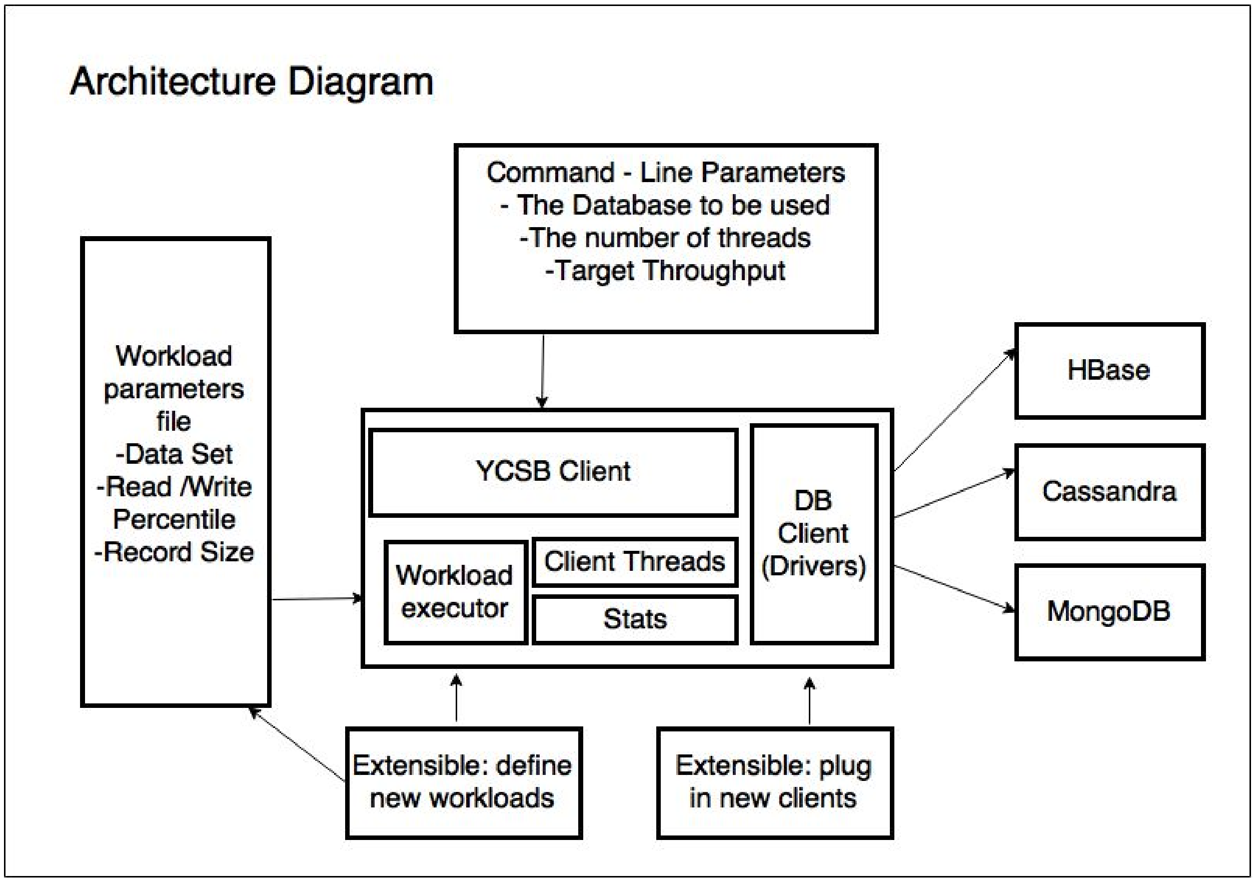
```
curl -O --location
https://github.com/brianfrankcooper/YCSB/releases/download/0.8.0/ycsb-
0.8.0.tar.gz
```

```
tar xfvz ycsb-0.8.0.tar.gz
cd ycsb-0.8.0
```

Architecture Diagram

In this section, we shall briefly describe the various components of the benchmarking system and how they interact with each other.

The YCSB client establishes connections with HBase, Cassandra and MongoDB through the DB Client drivers. Custom workloads can be defined using the workload executor. In order to do that the workload parameter file needs to specify the data set, read / write/ insert percentiles and the record size. Having built the workloads, we have to then specify the database to be used, the number of threads for the operations and the target throughput to the YCSB client to run the workloads using the command line parameters. On running the workloads, the workload executor then outputs the stats of all the latency parameters for the specific throughput values which are logged into the log files.



Cassandra Setup for YCSB

We have installed Apache Cassandra version 3.5. In this section we would briefly describe the steps involved in the setup

Prerequisites

Java (Preferably JDK 8)

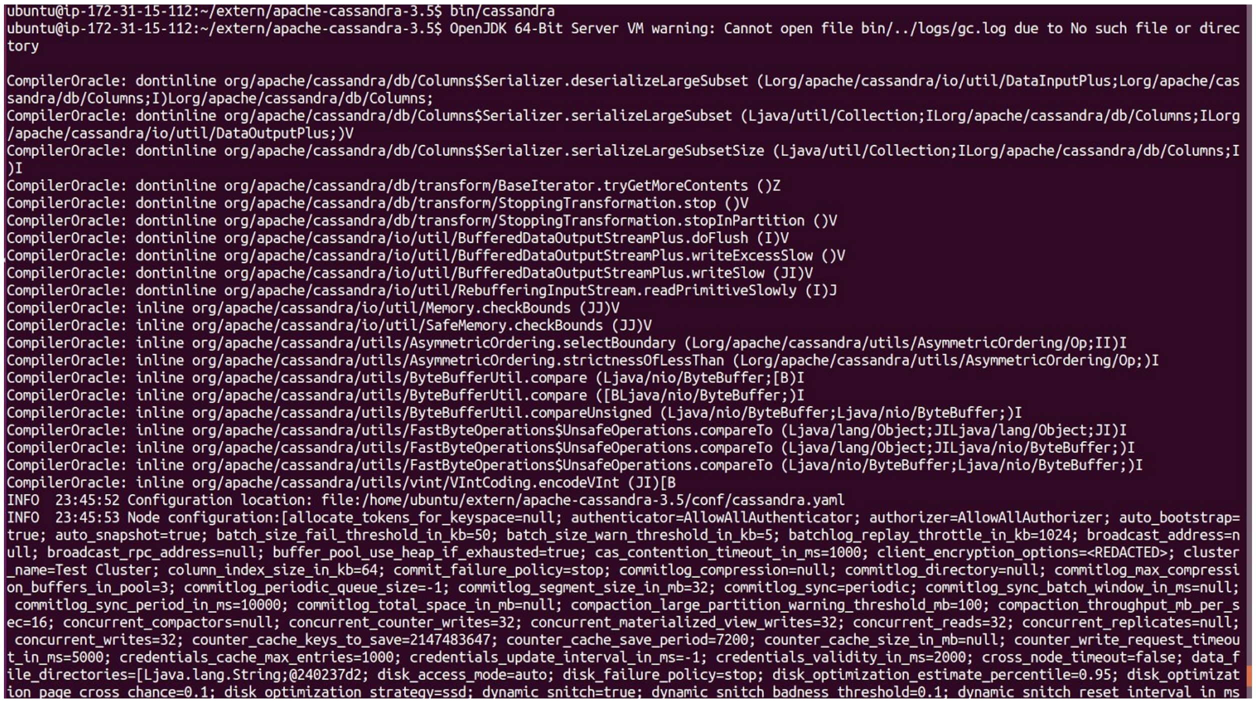
1. We downloaded Apache Cassandra from the Apache Cassandra website. We then untared the apache-cassandra-3.5-bin.tar.gz file by using the following command

   ```
   tar xvfz apache-cassandra-3.5-bin.tar.gz
   ```

2. Start Cassandra server using the following command

   ```
   bin/cassandra
   ```

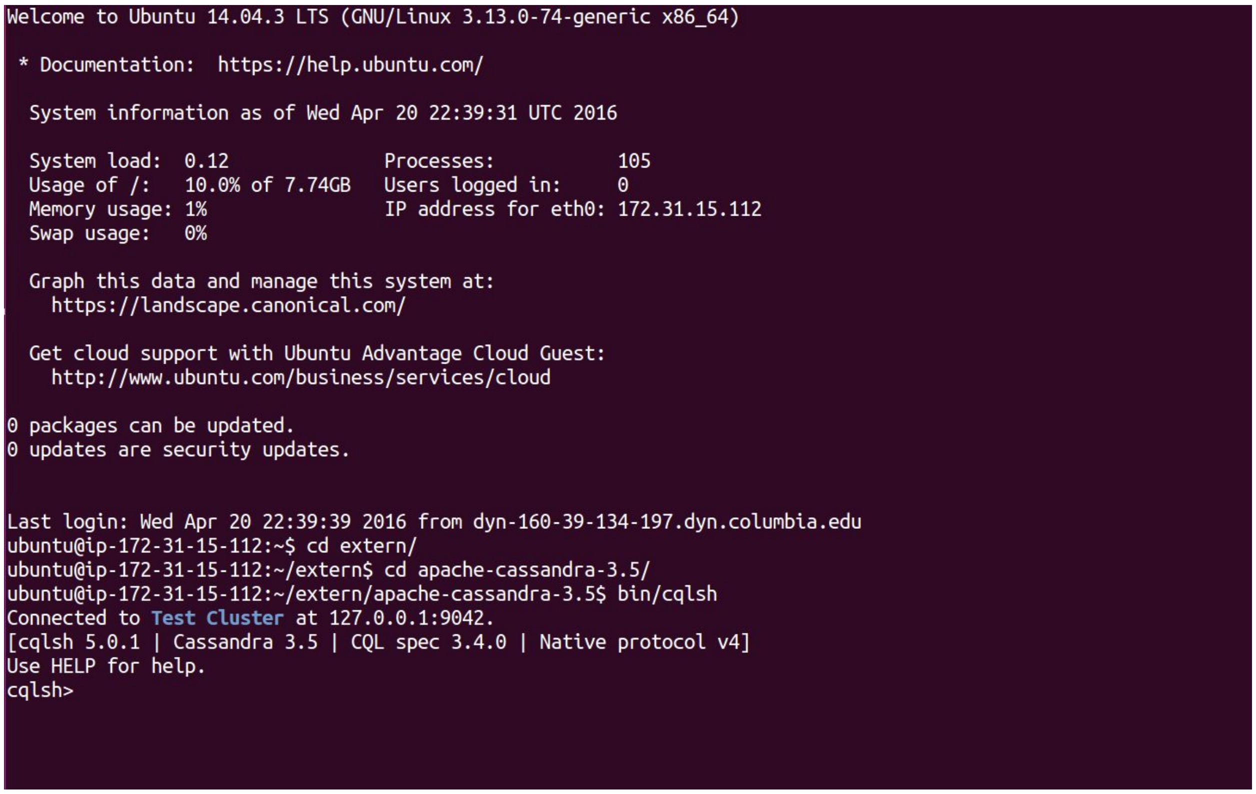
   The output will look like the screenshot below:



```
ubuntu@ip-172-31-15-112:~/extern/apache-cassandra-3.5$ bin/cassandra
ubuntu@ip-172-31-15-112:~/extern/apache-cassandra-3.5$ OpenJDK 64-Bit Server VM warning: Cannot open file bin/../logs/gc.log due to No such file or direc
tory

CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.deserializeLargeSubset (Lorg/apache/cassandra/io/util/DataInputPlus;Lorg/apache/cas
sandra/db/Columns;I)Lorg/apache/cassandra/db/Columns;
CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.serializeLargeSubset (Ljava/util/Collection;ILorg/apache/cassandra/db/Columns;ILorg
/apache/cassandra/io/util/DataOutputPlus;)V
CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.serializeLargeSubsetSize (Ljava/util/Collection;ILorg/apache/cassandra/db/Columns;I
)I
CompilerOracle: dontinline org/apache/cassandra/db/transform/BaseIterator.tryGetMoreContents ()Z
CompilerOracle: dontinline org/apache/cassandra/db/transform/StoppingTransformation.stop ()V
CompilerOracle: dontinline org/apache/cassandra/db/transform/StoppingTransformation.stopInPartition ()V
CompilerOracle: dontinline org/apache/cassandra/io/util/BufferedDataOutputStreamPlus.doFlush (I)V
CompilerOracle: dontinline org/apache/cassandra/io/util/BufferedDataOutputStreamPlus.writeExcessSlow ()V
CompilerOracle: dontinline org/apache/cassandra/io/util/BufferedDataOutputStreamPlus.writeSlow (JI)V
CompilerOracle: dontinline org/apache/cassandra/io/util/RebufferingInputStream.readPrimitiveSlowly (I)J
CompilerOracle: inline org/apache/cassandra/io/util/Memory.checkBounds (JJ)V
CompilerOracle: inline org/apache/cassandra/io/util/SafeMemory.checkBounds (JJ)V
CompilerOracle: inline org/apache/cassandra/utils/AsymmetricOrdering.selectBoundary (Lorg/apache/cassandra/utils/AsymmetricOrdering/Op;II)I
CompilerOracle: inline org/apache/cassandra/utils/AsymmetricOrdering.strictnessOfLessThan (Lorg/apache/cassandra/utils/AsymmetricOrdering/Op;)I
CompilerOracle: inline org/apache/cassandra/utils/ByteBufferUtil.compare (Ljava/nio/ByteBuffer;[B)I
CompilerOracle: inline org/apache/cassandra/utils/ByteBufferUtil.compare ([BLjava/nio/ByteBuffer;)I
CompilerOracle: inline org/apache/cassandra/utils/ByteBufferUtil.compareUnsigned (Ljava/nio/ByteBuffer;Ljava/nio/ByteBuffer;)I
CompilerOracle: inline org/apache/cassandra/utils/FastByteOperations$UnsafeOperations.compareTo (Ljava/lang/Object;JILjava/lang/Object;JI)I
CompilerOracle: inline org/apache/cassandra/utils/FastByteOperations$UnsafeOperations.compareTo (Ljava/lang/Object;JILjava/nio/ByteBuffer;)I
CompilerOracle: inline org/apache/cassandra/utils/FastByteOperations$UnsafeOperations.compareTo (Ljava/nio/ByteBuffer;Ljava/nio/ByteBuffer;)I
CompilerOracle: inline org/apache/cassandra/utils/vint/VIntCoding.encodeVInt (JI)[B
INFO  23:45:52 Configuration location: file:/home/ubuntu/extern/apache-cassandra-3.5/conf/cassandra.yaml
INFO  23:45:53 Node configuration:[allocate_tokens_for_keyspace=null; authenticator=AllowAllAuthenticator; authorizer=AllowAllAuthorizer; auto_bootstrap=
true; auto_snapshot=true; batch_size_fail_threshold_in_kb=50; batch_size_warn_threshold_in_kb=5; batchlog_replay_throttle_in_kb=1024; broadcast_address=n
ull; broadcast_rpc_address=null; buffer_pool_use_heap_if_exhausted=true; cas_contention_timeout_in_ms=1000; client_encryption_options=<REDACTED>; cluster
_name=Test Cluster; column_index_size_in_kb=64; commit_failure_policy=stop; commitlog_compression=null; commitlog_directory=null; commitlog_max_compressi
on_buffers_in_pool=3; commitlog_periodic_queue_size=-1; commitlog_segment_size_in_mb=32; commitlog_sync=periodic; commitlog_sync_batch_window_in_ms=null;
 commitlog_sync_period_in_ms=10000; commitlog_total_space_in_mb=null; compaction_large_partition_warning_threshold_mb=100; compaction_throughput_mb_per_s
ec=16; concurrent_compactors=null; concurrent_counter_writes=32; concurrent_materialized_view_writes=32; concurrent_reads=32; concurrent_replicates=null;
 concurrent_writes=32; counter_cache_keys_to_save=2147483647; counter_cache_save_period=7200; counter_cache_size_in_mb=null; counter_write_request_timeou
t_in_ms=5000; credentials_cache_max_entries=1000; credentials_update_interval_in_ms=-1; credentials_validity_in_ms=2000; cross_node_timeout=false; data_f
ile_directories=[Ljava.lang.String;@240237d2; disk_access_mode=auto; disk_failure_policy=stop; disk_optimization_estimate_percentile=0.95; disk_optimizat
ion_page_cross_chance=0.1; disk_optimization_strategy=ssd; dynamic_snitch=true; dynamic_snitch_badness_threshold=0.1; dynamic_snitch_reset_interval_in_ms
```

3. Open a new terminal window and start Cassandra Interactive Shell

   ```
   bin/cqlsh
   ```



```
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.13.0-74-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Wed Apr 20 22:39:31 UTC 2016

  System load:  0.12              Processes:           105
  Usage of /:   10.0% of 7.74GB   Users logged in:     0
  Memory usage: 1%                IP address for eth0: 172.31.15.112
  Swap usage:   0%

  Graph this data and manage this system at:
    https://landscape.canonical.com/

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

0 packages can be updated.
0 updates are security updates.


Last login: Wed Apr 20 22:39:39 2016 from dyn-160-39-134-197.dyn.columbia.edu
ubuntu@ip-172-31-15-112:~$ cd extern/
ubuntu@ip-172-31-15-112:~/extern$ cd apache-cassandra-3.5/
ubuntu@ip-172-31-15-112:~/extern/apache-cassandra-3.5$ bin/cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.5 | CQL spec 3.4.0 | Native protocol v4]
Use HELP for help.
cqlsh>
```

4. Creating a table for use with YCSB

For keyspace ycsb, table usertable:

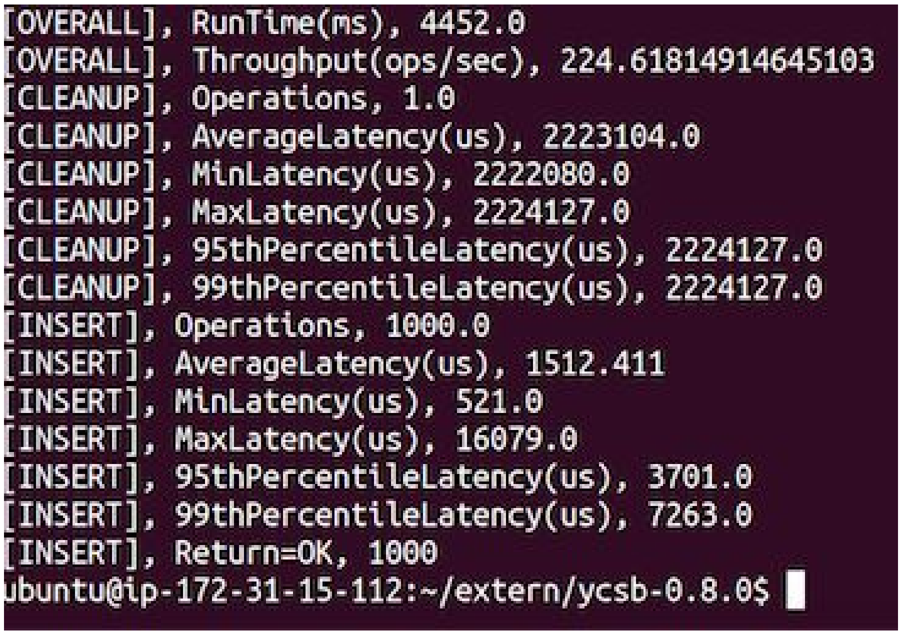
```
cqlsh> create keyspace ycsb
           WITH    REPLICATION    =    {'class'    :    'SimpleStrategy',
'replication_factor': 3 };
cqlsh> USE ycsb;
cqlsh> create table usertable (
   y_id varchar primary key,
   field0 varchar,
   field1 varchar,
   field2 varchar,
   field3 varchar,
   field4 varchar,
   field5 varchar,
   field6 varchar,
   field7 varchar,
   field8 varchar,
    field9 varchar);
```

Run the sample workload `workloads` on Cassandra database using the command specified below. YCSB comes with some pre-configured workloads. We will be using `workloada` to benchmark the databases.

```
./bin/ycsb load cassandra2-cql -P workloads/workloada -p
hosts=localhost
```

The -P parameter tell YCSB which workload to run and -p provides you a way to specify various Cassandra instance specific properties.

Executing the above command gives us the the following parameters of latency and throughput for `workload` on Cassandra as shown in the image below:



```
[OVERALL], RunTime(ms), 4452.0
[OVERALL], Throughput(ops/sec), 224.61814914645103
[CLEANUP], Operations, 1.0
[CLEANUP], AverageLatency(us), 2223104.0
[CLEANUP], MinLatency(us), 2222080.0
[CLEANUP], MaxLatency(us), 2224127.0
[CLEANUP], 95thPercentileLatency(us), 2224127.0
[CLEANUP], 99thPercentileLatency(us), 2224127.0
[INSERT], Operations, 1000.0
[INSERT], AverageLatency(us), 1512.411
[INSERT], MinLatency(us), 521.0
[INSERT], MaxLatency(us), 16079.0
[INSERT], 95thPercentileLatency(us), 3701.0
[INSERT], 99thPercentileLatency(us), 7263.0
[INSERT], Return=OK, 1000
ubuntu@ip-172-31-15-112:~/extern/ycsb-0.8.0$
```

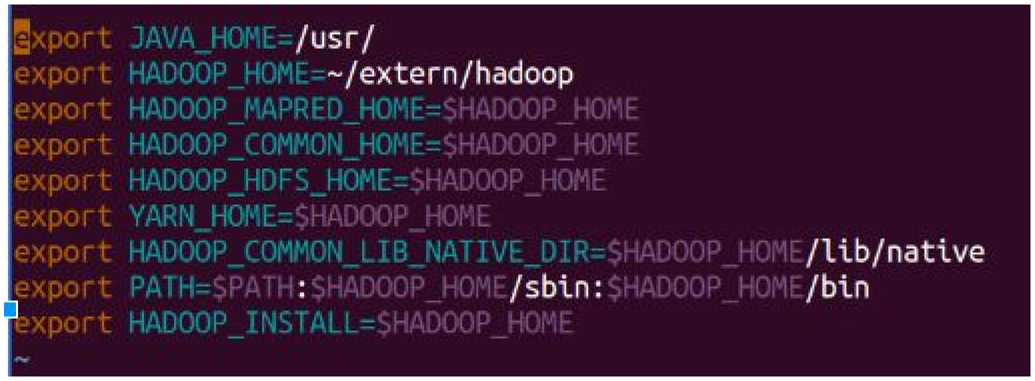
HBase Setup for YCSB

We have installed the HBase version 1.0.3. This process involved establishing the following prerequisites before the installing the above mentioned version of HBase:
Setting up a JDK of version 8 or above which we had already accomplished for Cassandra during its set-up. However, we had to set JAVA_HOME to the location of JDK and update the $PATH variable to $PATH:$JAVA_HOME/bin in the bashrc.
Setting up Hadoop in the standalone mode. We had installed Hadoop version 2.7.1 and updated the configuration files for the particular standalone mode.

Hadoop set-up

Here we discuss about the steps we followed to set Hadoop.

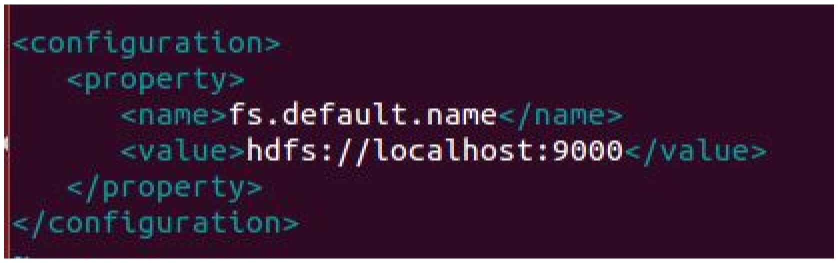
1. We downloaded the Hadoop version 2.7.1 from the Apache Hadoop website and untar the file.
2. Set the Hadoop environment variables by appending them to the bashrc file as shown in the image below. Along with this we also updated the JAVA_HOME variable with location of java in the hadoop-env.sh file at $HADOOP_HOME/etc/hadoop



```
export JAVA_HOME=/usr/
export HADOOP_HOME=~/extern/hadoop
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_INSTALL=$HADOOP_HOME
~
```

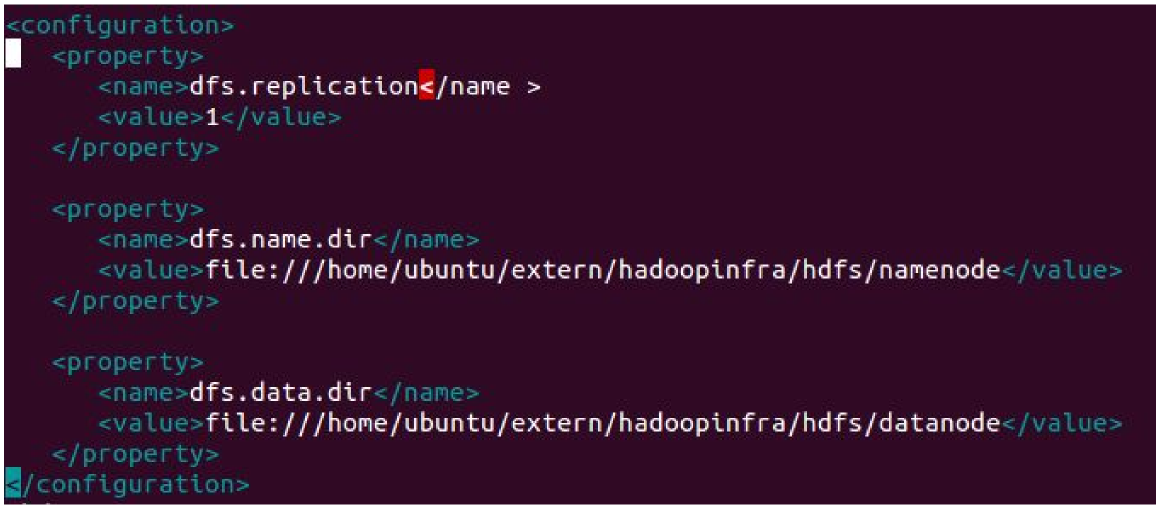
3. We need to make changes to the Hadoop configuration files located at $HADOOP_HOME/etc/hadoop according to our desired Hadoop infrastructure. We edit the following the configuration files

a. Core-site.xml : This file contains the port number used for Hadoop instance, memory allocated for file system, memory limit for storing data, and the size of Read/Write buffers. We set the localhost to port 9000



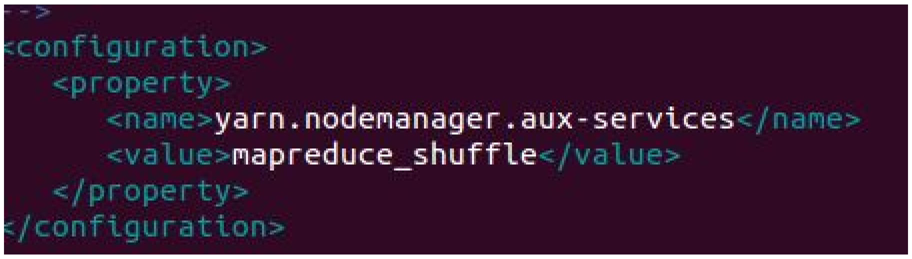
```
<configuration>
   <property>
      <name>fs.default.name</name>
      <value>hdfs://localhost:9000</value>
   </property>
</configuration>
```

b. hdfs-site.xml: This file contains the value of replication data, the namenode path, and the datanode path of our local file systems where we want to store the Hadoop infrastructure. In this file we had set the dfs data replication value to 1 and set the datanode and namenode paths as shown below



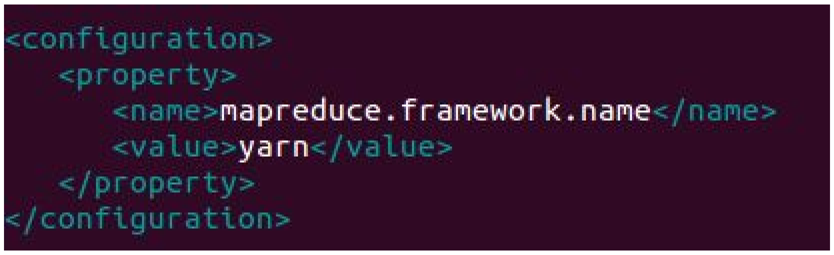
```
<configuration>
   <property>
      <name>dfs.replication</name >
      <value>1</value>
   </property>

   <property>
      <name>dfs.name.dir</name>
      <value>file:///home/ubuntu/extern/hadoopinfra/hdfs/namenode</value>
   </property>

   <property>
      <name>dfs.data.dir</name>
      <value>file:///home/ubuntu/extern/hadoopinfra/hdfs/datanode</value>
   </property>
</configuration>
```

c. yarn-site.xml: This file is used to configure yarn into Hadoop. We add the yarn nodemanager property in the file as shown belowmapred-site.xml: This file is used to specify the MapReduce framework that will be used by Hadoop. First we create the file by copying the mapred-site.xml.template to mapred-site.xml. Then we specify the MapReduce framework as yarn



```
-->
<configuration>
   <property>
      <name>yarn.nodemanager.aux-services</name>
      <value>mapreduce_shuffle</value>
   </property>
</configuration>
```

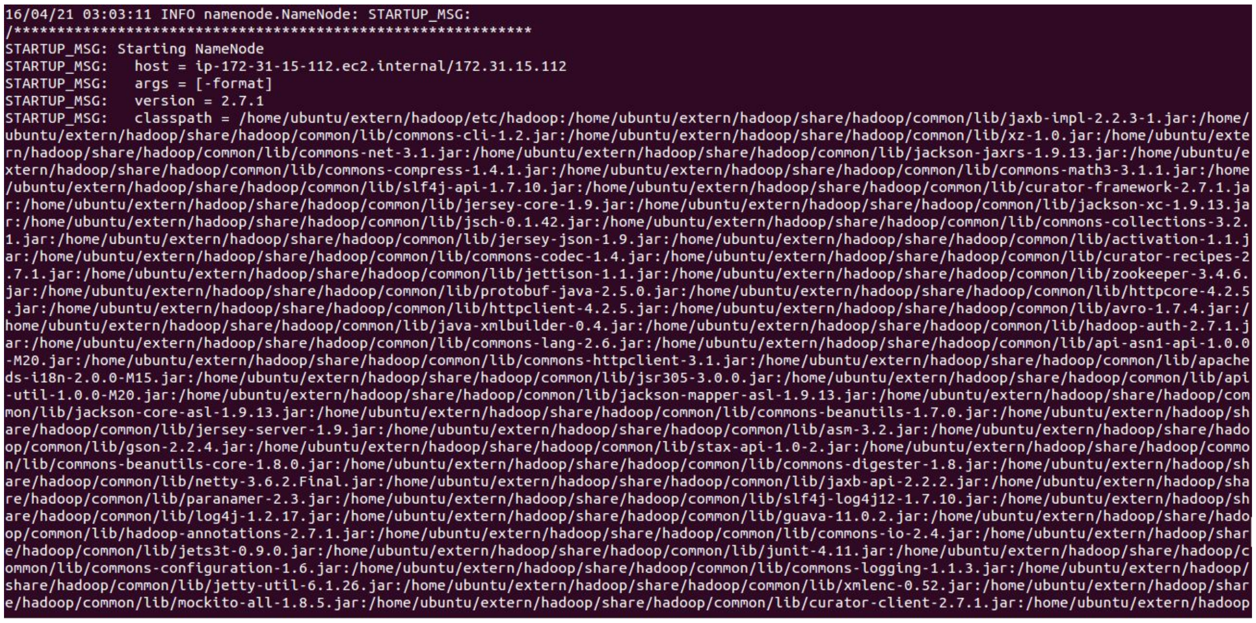
d. mapred-site.xml: This file is used to specify the MapReduce framework that will be used by Hadoop. First we create the file by copying the mapred-site.xml.template to mapred-site.xml. Then we specify the MapReduce framework as yarn



```
<configuration>
   <property>
      <name>mapreduce.framework.name</name>
      <value>yarn</value>
   </property>
</configuration>
```

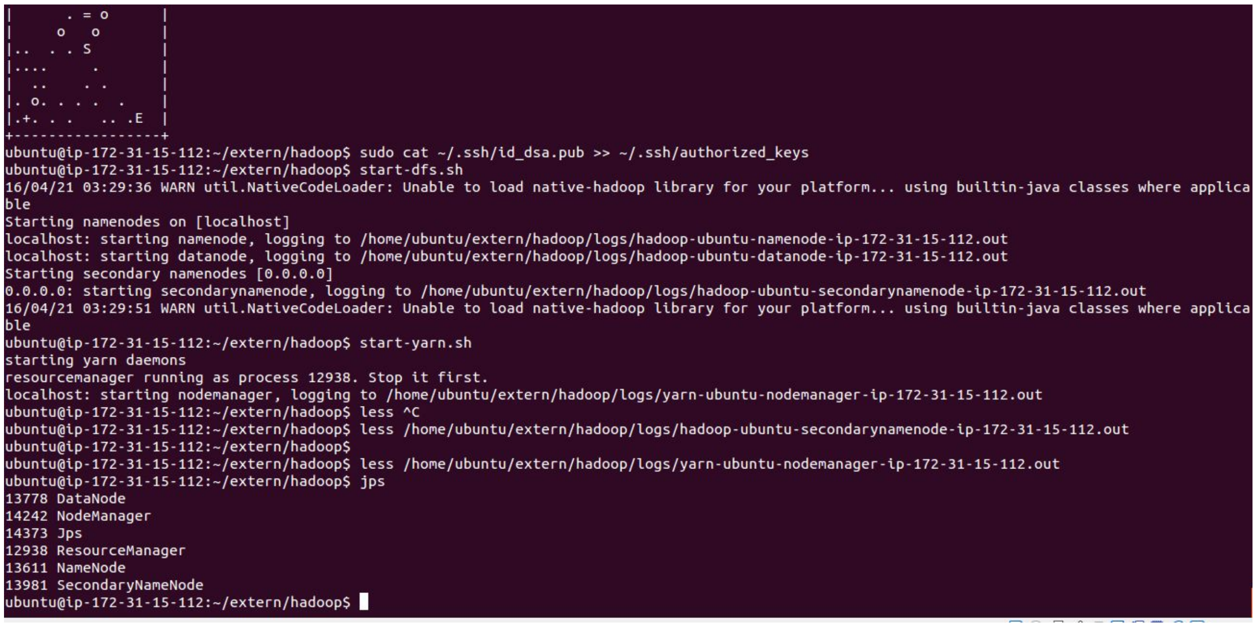
4. Having set the configuration files, we format the namenode using the command

```
hdfs namenode -format
```



```
16/04/21 03:03:11 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = ip-172-31-15-112.ec2.internal/172.31.15.112
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 2.7.1
STARTUP_MSG:   classpath = /home/ubuntu/extern/hadoop/etc/hadoop:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/jaxb-impl-2.2.3-1.jar:/home/
ubuntu/extern/hadoop/share/hadoop/common/lib/commons-cli-1.2.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/xz-1.0.jar:/home/ubuntu/exte
rn/hadoop/share/hadoop/common/lib/commons-net-3.1.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/jackson-jaxrs-1.9.13.jar:/home/ubuntu/e
xtern/hadoop/share/hadoop/common/lib/commons-compress-1.4.1.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/commons-math3-3.1.1.jar:/home
/ubuntu/extern/hadoop/share/hadoop/common/lib/slf4j-api-1.7.10.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/curator-framework-2.7.1.ja
r:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/jersey-core-1.9.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/jackson-xc-1.9.13.ja
r:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/jsch-0.1.42.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/commons-collections-3.2.
1.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/jersey-json-1.9.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/activation-1.1.j
ar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/commons-codec-1.4.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/curator-recipes-2
.7.1.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/jettison-1.1.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/zookeeper-3.4.6.
jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/protobuf-java-2.5.0.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/httpcore-4.2.5
.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/httpclient-4.2.5.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/avro-1.7.4.jar:/
home/ubuntu/extern/hadoop/share/hadoop/common/lib/java-xmlbuilder-0.4.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/hadoop-auth-2.7.1.j
ar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/commons-lang-2.6.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/api-asn1-api-1.0.0
-M20.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/commons-httpclient-3.1.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/apache
ds-i18n-2.0.0-M15.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/jsr305-3.0.0.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/api
-util-1.0.0-M20.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/jackson-mapper-asl-1.9.13.jar:/home/ubuntu/extern/hadoop/share/hadoop/com
mon/lib/jackson-core-asl-1.9.13.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/commons-beanutils-1.7.0.jar:/home/ubuntu/extern/hadoop/sh
are/hadoop/common/lib/jersey-server-1.9.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/asm-3.2.jar:/home/ubuntu/extern/hadoop/share/hado
op/common/lib/gson-2.2.4.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/stax-api-1.0-2.jar:/home/ubuntu/extern/hadoop/share/hadoop/commo
n/lib/commons-beanutils-core-1.8.0.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/commons-digester-1.8.jar:/home/ubuntu/extern/hadoop/sh
are/hadoop/common/lib/netty-3.6.2.Final.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/jaxb-api-2.2.2.jar:/home/ubuntu/extern/hadoop/sha
re/hadoop/common/lib/paranamer-2.3.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar:/home/ubuntu/extern/hadoop/sh
are/hadoop/common/lib/log4j-1.2.17.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/guava-11.0.2.jar:/home/ubuntu/extern/hadoop/share/hado
op/common/lib/hadoop-annotations-2.7.1.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/commons-io-2.4.jar:/home/ubuntu/extern/hadoop/shar
e/hadoop/common/lib/jets3t-0.9.0.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/junit-4.11.jar:/home/ubuntu/extern/hadoop/share/hadoop/c
ommon/lib/commons-configuration-1.6.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/commons-logging-1.1.3.jar:/home/ubuntu/extern/hadoop/
share/hadoop/common/lib/jetty-util-6.1.26.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/xmlenc-0.52.jar:/home/ubuntu/extern/hadoop/shar
e/hadoop/common/lib/mockito-all-1.8.5.jar:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/curator-client-2.7.1.jar:/home/ubuntu/extern/hadoop
```

After formatting the namenode, we verified the successful running of Hadoop DFS and Yarn Script by starting them using the commands `start-dfs.sh` and `start-yarn.sh`. We then ran the jps command to check if the resources are live as shown in the image below. However, when we were trying to start the nodes initially, it wasn't authorized to create the nodes as we were running it on a VM on AWS EC2. So, we had to generate SSH keys to establish the authorisation to start the nodes.

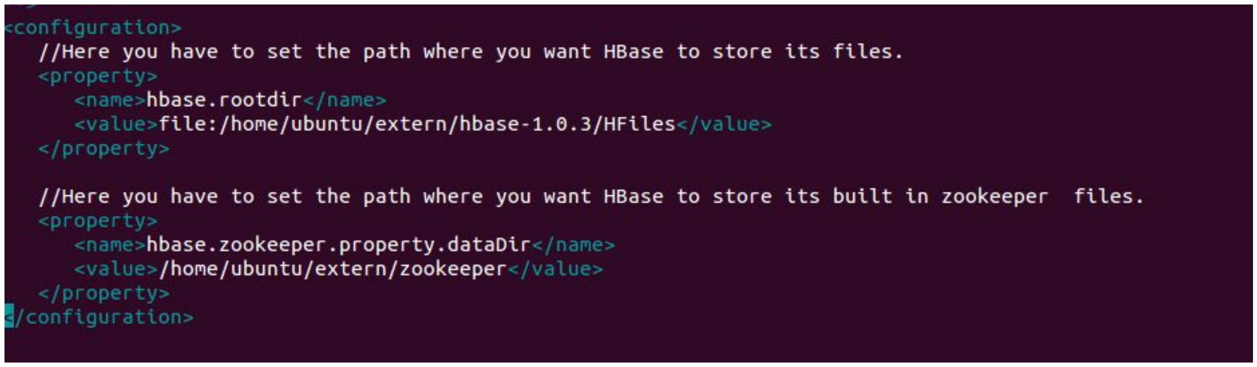


```
|      . = o      |
|     o   o       |
|..  . . S        |
|....     .       |
|  ..    . .      |
|. o. . . .  .    |
|.+. .  .   .. .E |
+-----------------+
ubuntu@ip-172-31-15-112:~/extern/hadoop$ sudo cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
ubuntu@ip-172-31-15-112:~/extern/hadoop$ start-dfs.sh
16/04/21 03:29:36 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applica
ble
Starting namenodes on [localhost]
localhost: starting namenode, logging to /home/ubuntu/extern/hadoop/logs/hadoop-ubuntu-namenode-ip-172-31-15-112.out
localhost: starting datanode, logging to /home/ubuntu/extern/hadoop/logs/hadoop-ubuntu-datanode-ip-172-31-15-112.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /home/ubuntu/extern/hadoop/logs/hadoop-ubuntu-secondarynamenode-ip-172-31-15-112.out
16/04/21 03:29:51 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applica
ble
ubuntu@ip-172-31-15-112:~/extern/hadoop$ start-yarn.sh
starting yarn daemons
resourcemanager running as process 12938. Stop it first.
localhost: starting nodemanager, logging to /home/ubuntu/extern/hadoop/logs/yarn-ubuntu-nodemanager-ip-172-31-15-112.out
ubuntu@ip-172-31-15-112:~/extern/hadoop$ less ^C
ubuntu@ip-172-31-15-112:~/extern/hadoop$ less /home/ubuntu/extern/hadoop/logs/hadoop-ubuntu-secondarynamenode-ip-172-31-15-112.out
ubuntu@ip-172-31-15-112:~/extern/hadoop$
ubuntu@ip-172-31-15-112:~/extern/hadoop$ less /home/ubuntu/extern/hadoop/logs/yarn-ubuntu-nodemanager-ip-172-31-15-112.out
ubuntu@ip-172-31-15-112:~/extern/hadoop$ jps
13778 DataNode
14242 NodeManager
14373 Jps
12938 ResourceManager
13611 NameNode
13981 SecondaryNameNode
ubuntu@ip-172-31-15-112:~/extern/hadoop$
```

This completed the prerequisites for HBase

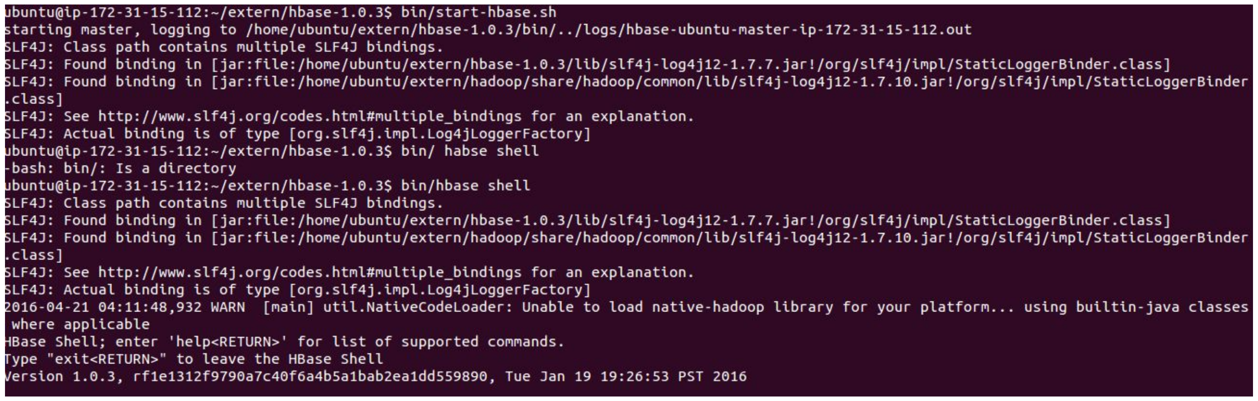
HBase installation involved the following steps:

1. We downloaded the HBase version 2.7.1 from the Apache HBase website and untarred it
2. We then updated the JAVA_HOME variable in the hbase-env.sh file in the conf folder of HBase.
3. Inorder to run the HBase in standalone mode we edit the hbase-site.xml configuration file by adding two properties corresponding to the HBase root directory and HBase zookeeper data directory as shown in the image below.



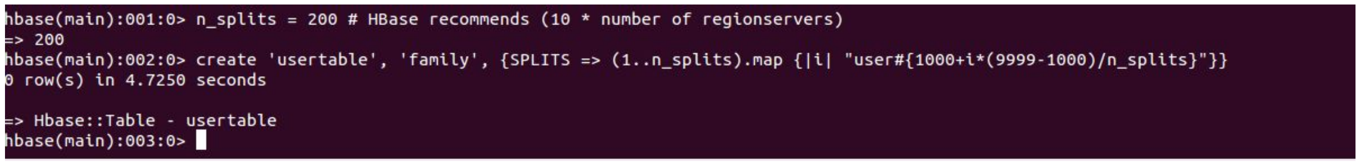
```
<configuration>
   //Here you have to set the path where you want HBase to store its files.
   <property>
      <name>hbase.rootdir</name>
      <value>file:/home/ubuntu/extern/hbase-1.0.3/HFiles</value>
   </property>

   //Here you have to set the path where you want HBase to store its built in zookeeper  files.
   <property>
      <name>hbase.zookeeper.property.dataDir</name>
      <value>/home/ubuntu/extern/zookeeper</value>
   </property>
</configuration>
```

4. Now we ran the command start-hbase.sh to verify if the installation had been successful. This starts the master node as shown in the image below. Following this we had initiated the HBase shell to perform a simple task as shown.



```
ubuntu@ip-172-31-15-112:~/extern/hbase-1.0.3$ bin/start-hbase.sh
starting master, logging to /home/ubuntu/extern/hbase-1.0.3/bin/../logs/hbase-ubuntu-master-ip-172-31-15-112.out
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/ubuntu/extern/hbase-1.0.3/lib/slf4j-log4j12-1.7.7.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder
.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
ubuntu@ip-172-31-15-112:~/extern/hbase-1.0.3$ bin/ habse shell
-bash: bin/: Is a directory
ubuntu@ip-172-31-15-112:~/extern/hbase-1.0.3$ bin/hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/ubuntu/extern/hbase-1.0.3/lib/slf4j-log4j12-1.7.7.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/ubuntu/extern/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder
.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2016-04-21 04:11:48,932 WARN  [main] util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes
 where applicable
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.0.3, rf1e1312f9790a7c40f6a4b5a1bab2ea1dd559890, Tue Jan 19 19:26:53 PST 2016
```

This completed the successful installation of HBase. Following this we create a small table on HBase called ‘usertable’ with a column family called ‘family’ as shown in the image below.

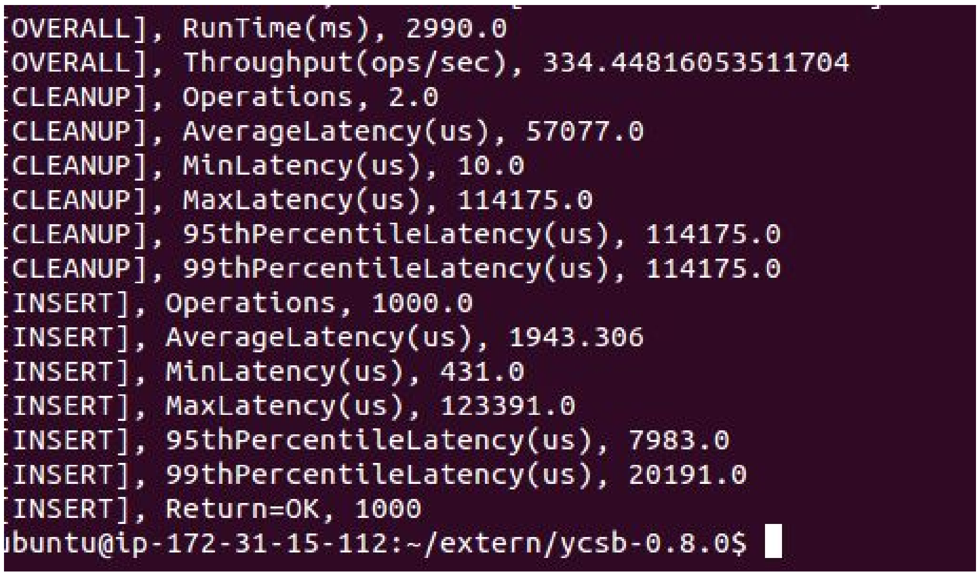


```
hbase(main):001:0> n_splits = 200 # HBase recommends (10 * number of regionservers)
=> 200
hbase(main):002:0> create 'usertable', 'family', {SPLITS => (1..n_splits).map {|i| "user#{1000+i*(9999-1000)/n_splits}"}}
0 row(s) in 4.7250 seconds

=> Hbase::Table - usertable
hbase(main):003:0>
```

Using this table as the test data we run a test workload on the YCSB using the hbase10 driver supported by YCSB to connect to HBase in the following way

```
bin/ycsb load hbase10 -P workloads/workloada -cp /HBASE-HOME-DIR/conf
-p table=usertable -p columnfamily=family
```

This generated the various parameters of latency and throughput for this particular test data on HBase as shown in the image below.

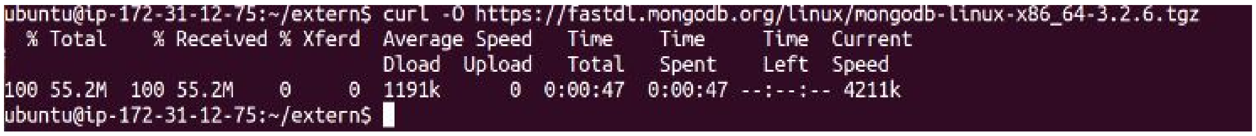


```
[OVERALL], RunTime(ms), 2990.0
[OVERALL], Throughput(ops/sec), 334.44816053511704
[CLEANUP], Operations, 2.0
[CLEANUP], AverageLatency(us), 57077.0
[CLEANUP], MinLatency(us), 10.0
[CLEANUP], MaxLatency(us), 114175.0
[CLEANUP], 95thPercentileLatency(us), 114175.0
[CLEANUP], 99thPercentileLatency(us), 114175.0
[INSERT], Operations, 1000.0
[INSERT], AverageLatency(us), 1943.306
[INSERT], MinLatency(us), 431.0
[INSERT], MaxLatency(us), 123391.0
[INSERT], 95thPercentileLatency(us), 7983.0
[INSERT], 99thPercentileLatency(us), 20191.0
[INSERT], Return=OK, 1000
ubuntu@ip-172-31-15-112:~/extern/ycsb-0.8.0$
```

MongoDB Setup for YCSB

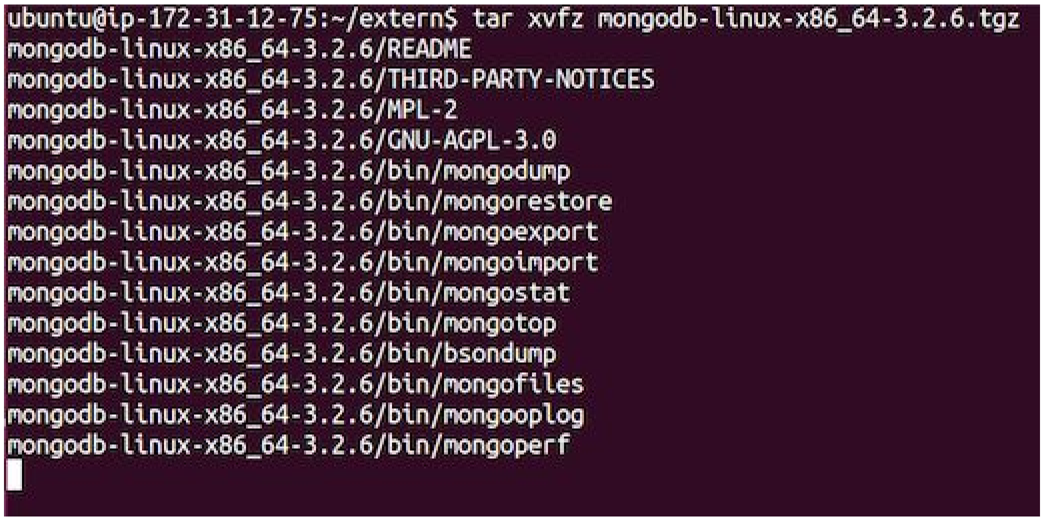
We have installed MongoDB version 3.2.6.

We downloaded the MongoDB setup file using the curl command.



```
ubuntu@ip-172-31-12-75:~/extern$ curl -O https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-3.2.6.tgz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 55.2M  100 55.2M    0     0  1191k      0  0:00:47  0:00:47 --:--:-- 4211k
ubuntu@ip-172-31-12-75:~/extern$
```

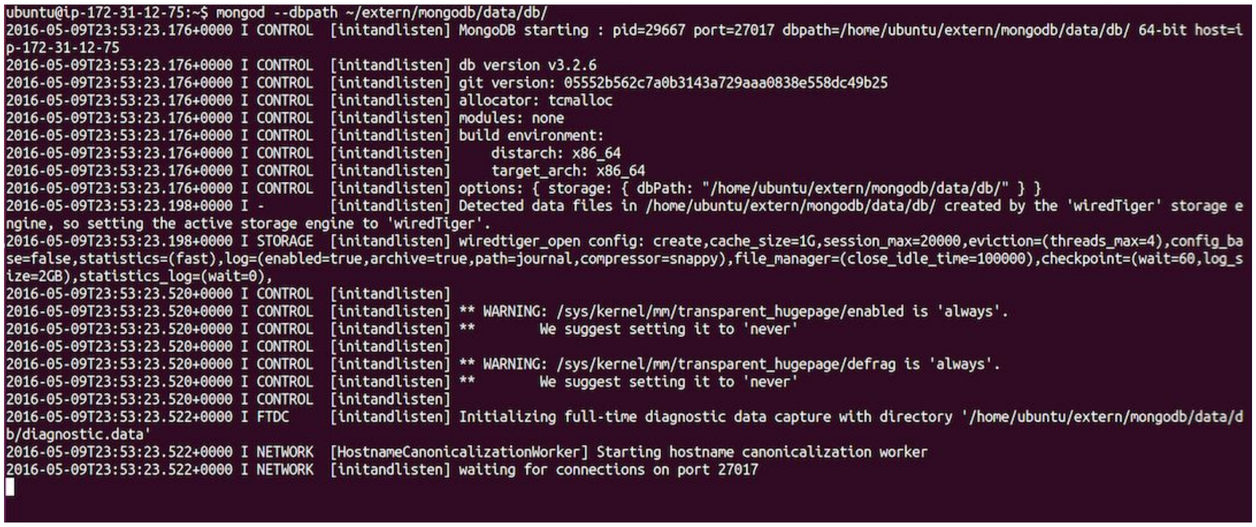
We then used the untar command to untar the tgz file.



```
ubuntu@ip-172-31-12-75:~/extern$ tar xvfz mongodb-linux-x86_64-3.2.6.tgz
mongodb-linux-x86_64-3.2.6/README
mongodb-linux-x86_64-3.2.6/THIRD-PARTY-NOTICES
mongodb-linux-x86_64-3.2.6/MPL-2
mongodb-linux-x86_64-3.2.6/GNU-AGPL-3.0
mongodb-linux-x86_64-3.2.6/bin/mongodump
mongodb-linux-x86_64-3.2.6/bin/mongorestore
mongodb-linux-x86_64-3.2.6/bin/mongoexport
mongodb-linux-x86_64-3.2.6/bin/mongoimport
mongodb-linux-x86_64-3.2.6/bin/mongostat
mongodb-linux-x86_64-3.2.6/bin/mongotop
mongodb-linux-x86_64-3.2.6/bin/bsondump
mongodb-linux-x86_64-3.2.6/bin/mongofiles
mongodb-linux-x86_64-3.2.6/bin/mongooplog
mongodb-linux-x86_64-3.2.6/bin/mongoperf
```

We then started mongodb using the command

```
mongod --dbpath ~/extern/mongodb/data/db/
```

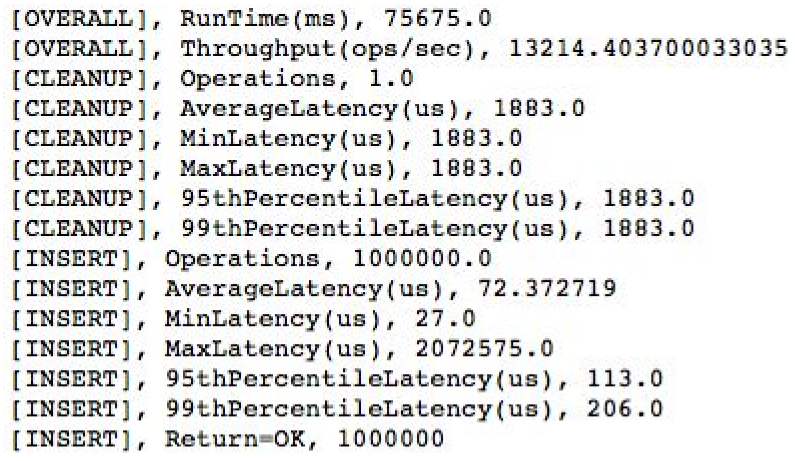


```
ubuntu@ip-172-31-12-75:~$ mongod --dbpath ~/extern/mongodb/data/db/
2016-05-09T23:53:23.176+0000 I CONTROL  [initandlisten] MongoDB starting : pid=29667 port=27017 dbpath=/home/ubuntu/extern/mongodb/data/db/ 64-bit host=i
p-172-31-12-75
2016-05-09T23:53:23.176+0000 I CONTROL  [initandlisten] db version v3.2.6
2016-05-09T23:53:23.176+0000 I CONTROL  [initandlisten] git version: 05552b562c7a0b3143a729aaa0838e558dc49b25
2016-05-09T23:53:23.176+0000 I CONTROL  [initandlisten] allocator: tcmalloc
2016-05-09T23:53:23.176+0000 I CONTROL  [initandlisten] modules: none
2016-05-09T23:53:23.176+0000 I CONTROL  [initandlisten] build environment:
2016-05-09T23:53:23.176+0000 I CONTROL  [initandlisten]     distarch: x86_64
2016-05-09T23:53:23.176+0000 I CONTROL  [initandlisten]     target_arch: x86_64
2016-05-09T23:53:23.176+0000 I CONTROL  [initandlisten] options: { storage: { dbPath: "/home/ubuntu/extern/mongodb/data/db/" } }
2016-05-09T23:53:23.198+0000 I -        [initandlisten] Detected data files in /home/ubuntu/extern/mongodb/data/db/ created by the 'wiredTiger' storage e
ngine, so setting the active storage engine to 'wiredTiger'.
2016-05-09T23:53:23.198+0000 I STORAGE  [initandlisten] wiredtiger_open config: create,cache_size=1G,session_max=20000,eviction=(threads_max=4),config_ba
se=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000),checkpoint=(wait=60,log_s
ize=2GB),statistics_log=(wait=0),
2016-05-09T23:53:23.520+0000 I CONTROL  [initandlisten]
2016-05-09T23:53:23.520+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2016-05-09T23:53:23.520+0000 I CONTROL  [initandlisten] **        We suggest setting it to 'never'
2016-05-09T23:53:23.520+0000 I CONTROL  [initandlisten]
2016-05-09T23:53:23.520+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/defrag is 'always'.
2016-05-09T23:53:23.520+0000 I CONTROL  [initandlisten] **        We suggest setting it to 'never'
2016-05-09T23:53:23.520+0000 I CONTROL  [initandlisten]
2016-05-09T23:53:23.522+0000 I FTDC     [initandlisten] Initializing full-time diagnostic data capture with directory '/home/ubuntu/extern/mongodb/data/d
b/diagnostic.data'
2016-05-09T23:53:23.522+0000 I NETWORK  [HostnameCanonicalizationWorker] Starting hostname canonicalization worker
2016-05-09T23:53:23.522+0000 I NETWORK  [initandlisten] waiting for connections on port 27017
```

We loaded the data in MongoDB and then we ran a test workload on the YCSB using the MongDB driver supported by YCSB to connect to MongoDB in the following way

```
./bin/ycsb run mongodb -P workloads/workloada -s -threads 2 -target 40 -p recordcount=1000000 -p mongodb.url=mongodb://localhost:27017/ycsb?w=0 > ~/extern/performance_logs/mongo/workloada-40.log
```

This generated the various parameters of latency and throughput for this particular test data on MongoDB as shown in the image below.



```
[OVERALL], RunTime(ms), 75675.0
[OVERALL], Throughput(ops/sec), 13214.403700033035
[CLEANUP], Operations, 1.0
[CLEANUP], AverageLatency(us), 1883.0
[CLEANUP], MinLatency(us), 1883.0
[CLEANUP], MaxLatency(us), 1883.0
[CLEANUP], 95thPercentileLatency(us), 1883.0
[CLEANUP], 99thPercentileLatency(us), 1883.0
[INSERT], Operations, 1000000.0
[INSERT], AverageLatency(us), 72.372719
[INSERT], MinLatency(us), 27.0
[INSERT], MaxLatency(us), 2072575.0
[INSERT], 95thPercentileLatency(us), 113.0
[INSERT], 99thPercentileLatency(us), 206.0
[INSERT], Return=OK, 1000000
```

Building Workloads on YCSB

The main workloads that we considered were

A. 50% Reads & 50% Updates.
B. 95% Read (heavy read) & 5% Updates.
C. 100% Reads (Read Intensive).
D. 95% Read (heavy read) & 5% Insert.
E. 95% Updates (heavy write) & 5% Reads

Inserts correspond to creating a new data item. While update involves modifying the value of a data item.

Creating a Workload

One can also choose to create a custom workload. The description of some of the core properties is as follows:

- fieldcount: the number of fields in a record (default: 10)
- fieldlength: the size of each field (default: 100)
- readallfields: should reads read all fields (true) or just one (false) (default: true)
- readproportion: what proportion of operations should be reads (default: 0.95)
- updateproportion: what proportion of operations should be updates (default: 0.05)
- insertproportion: what proportion of operations should be inserts (default: 0)
- scanproportion: what proportion of operations should be scans (default: 0)
- readmodifywriteproportion: what proportion of operations should be read a record, modify it, write it back (default: 0)

- requestdistribution: what distribution should be used to select the records to operate on – uniform, zipfian or latest (default: uniform)
- maxscanlength: for scans, what is the maximum number of records to scan (default: 1000)
- scanlengthdistribution: for scans, what distribution should be used to choose the number of records to scan, for each - scan, between 1 and maxscanlength (default: uniform)
- insertorder: should records be inserted in order by key (“ordered”), or in hashed order (“hashed”) (default: hashed)
- operationcount: number of operations to perform.
- maxexecutiontime: maximum execution time in seconds. The benchmark runs until either the operation count has exhausted or the maximum specified time has elapsed, whichever is earlier.
- table: the name of the table (default: usertable)
- recordcount: number of records to load into the database initially (default: 0)

Workload A 50% Read - 50% Update

This workload has the following properties

```
Recordcount =1000
operationcount=1000
workload=com.yahoo.ycsb.workloads.coreworkload
readallfields=true
readproportion=0.5
updateproportion=0.5
scanproportion=0
insertproportion=0
requestdistribution=zipfian
```

Workloads B 95% Read (Read Heavy) and 5% Update

The predominant read workload has the following properties

```
Readcount =1000
operationcount=1000
workload=com.yahoo.ycsb.workloads.coreworkload
readallfields=true
readproportion=0.95
updateproportion=0.05
scanproportion=0
insertproportion=0
requestdistribution=zipfian
```

Workloads C 100% (Read intensive)

The read intensive workload has the following properties

```
recordcount= 1000000
operationcount=1000
workload=com.yahoo.ycsb.workloads.coreworkload
readallfields=true
readproportion=1.0
updateproportion=0
scanproportion=0
insertproportion=0
requestdistribution=zipfian
```

Workloads D 95% Read (Read Heavy) and 5% Insert

The predominant read workload has the following properties

```
Readcount =1000
operationcount=1000
workload=com.yahoo.ycsb.workloads.coreworkload
readallfields=true
readproportion=0.95
updateproportion=0
scanproportion=0
insertproportion=0.05
requestdistribution=zipfian
```

Workloads E 95% Update (write heavy) and 5% Read

The predominant write workload has the following properties

```
Readcount =1000
operationcount=1000
workload=com.yahoo.ycsb.workloads.coreworkload
readallfields=true
readproportion=0.05
updateproportion=0.95
scanproportion=0
insertproportion=0
requestdistribution=zipfian
```

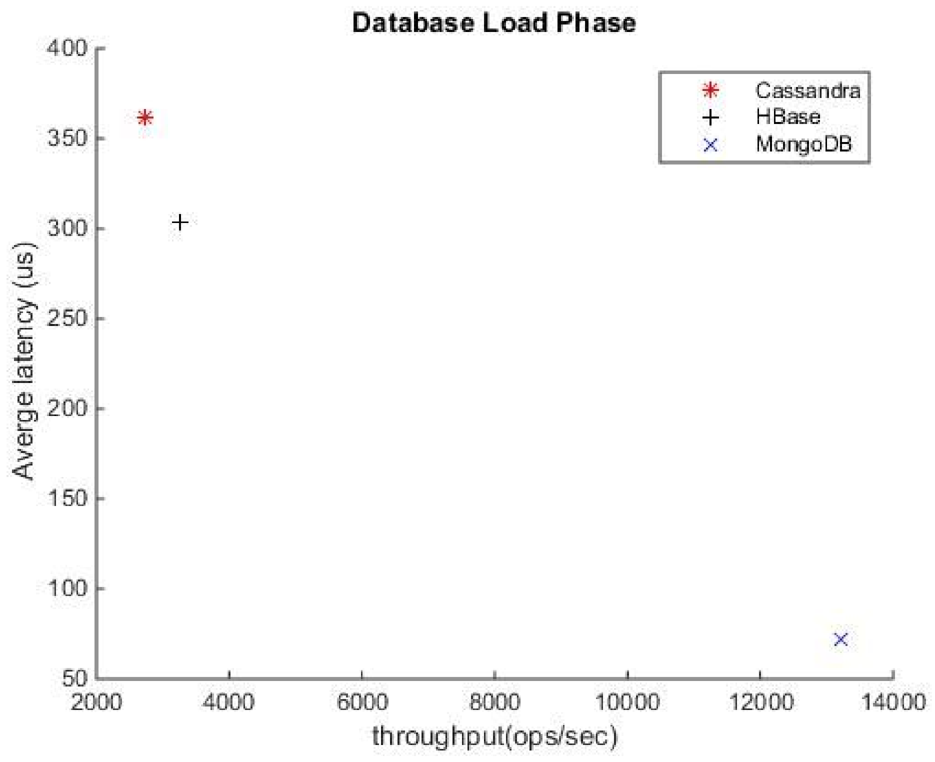
In this way we had built all the 5 workloads on YCSB

Analysis & Results

In the graphs below, we have plotted the results obtained for the load phase and running the above listed workloads.

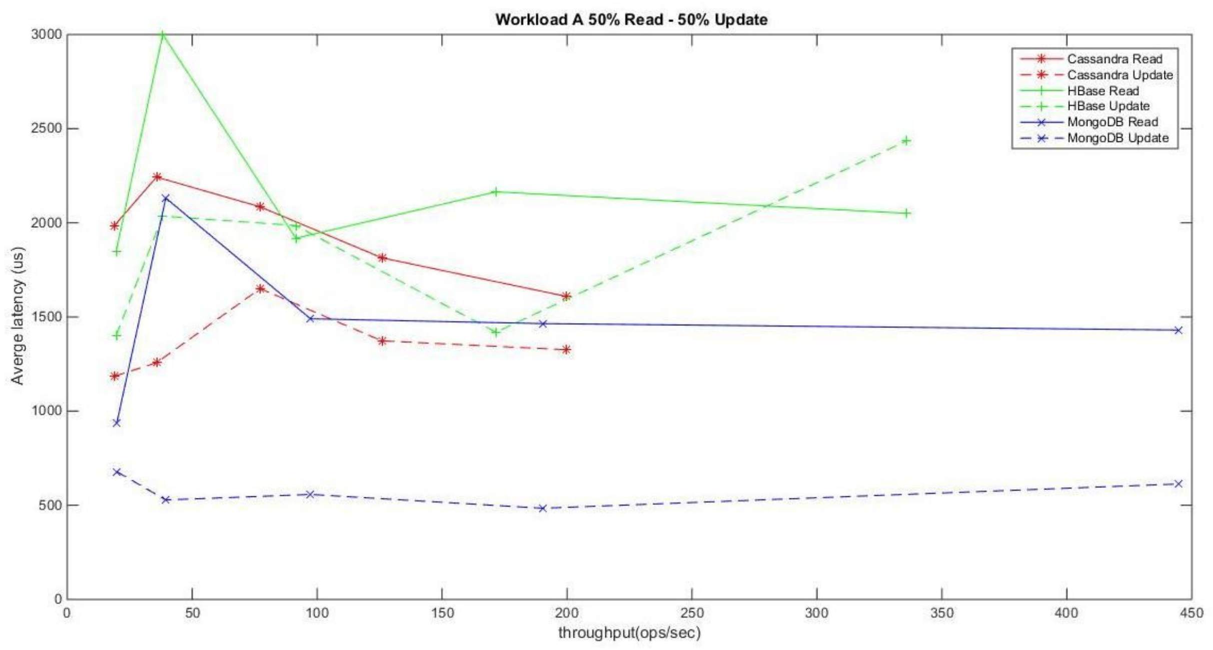
Load Phase

In the load phase we loaded 1 million data points of 1KB each. The plot below shows the performance of the load operation. As we can see MongoDB could handle higher throughput value and thus the average latency for the operations was relatively lower. Whereas the throughput and average latency values for Cassandra and HBase have similar characteristics.



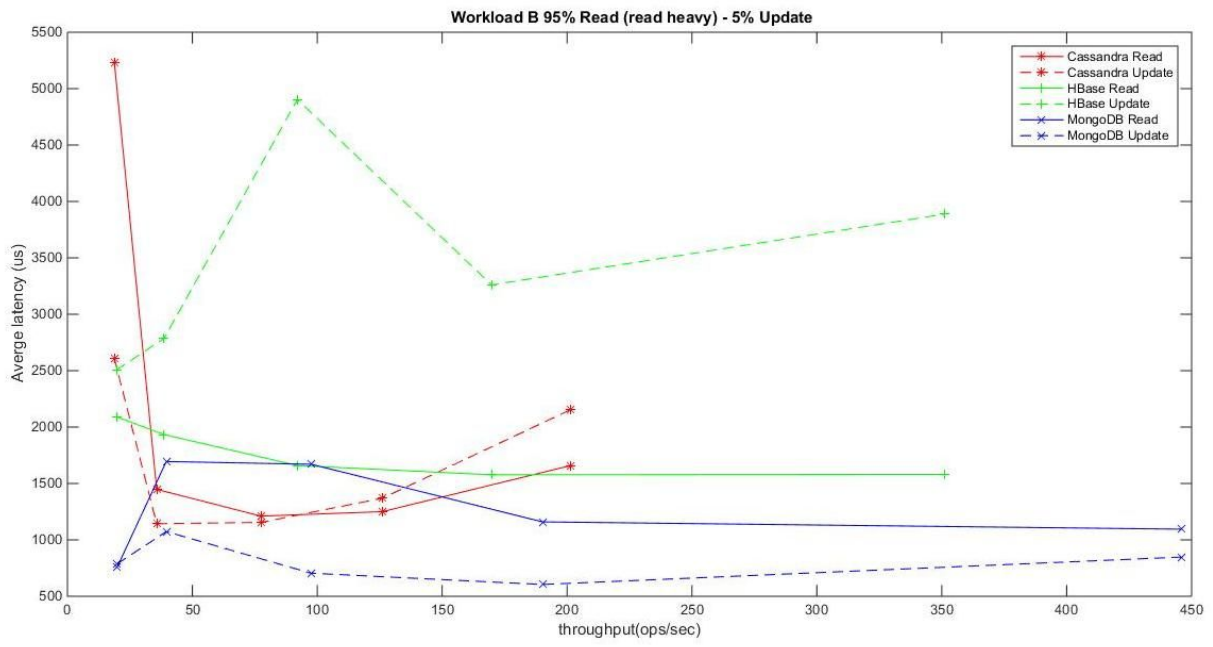
50% Reads and 50% Updates

The dotted lines indicate Updates while the solid lines indicate Reads. It is seen that, in the case of Updates, MongoDB's average latency is almost constant irrespective of the throughput. The average latency of MongoDB is seen to be lower than that of Cassandra and HBase in the case of Reads..



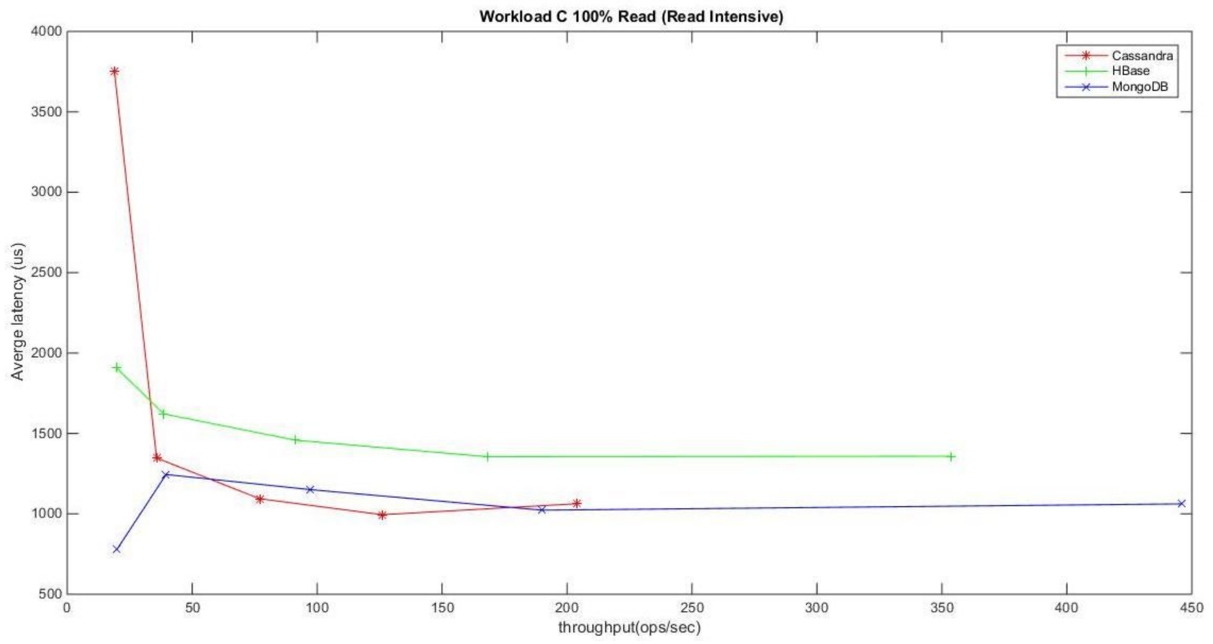
95% READS AND 5% UPDATES

Just like in the above graph, the dotted lines indicate updates while the thick lines indicates Reads. It is seen that the average latency is low for MongoDB in the case of Update even when the number of operations per second increases. However for Reads Cassandra is found to perform better upto a certain threshold of throughput, after which the average latency increases steeply in contrast to MongoDB whose average latency remains constant.



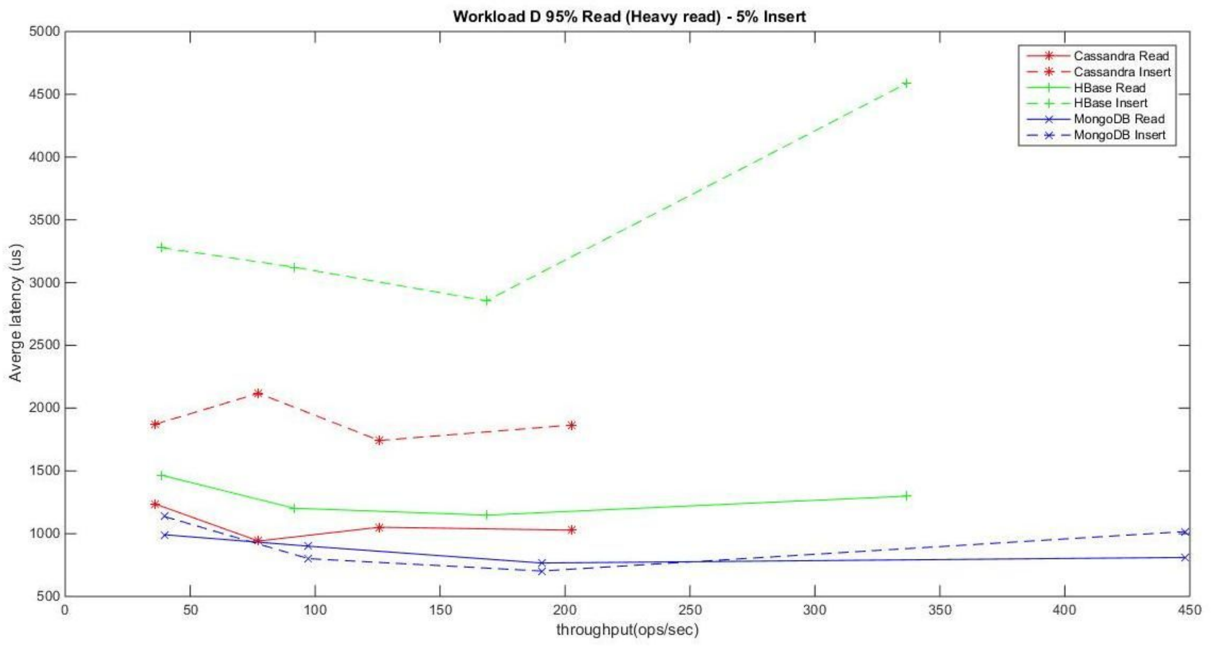
100% READS (READ INTENSIVE)

In the case of 100% reads, Cassandra is found to perform almost as good as MongoDB. But after crossing a certain threshold of throughput, its performance drops.



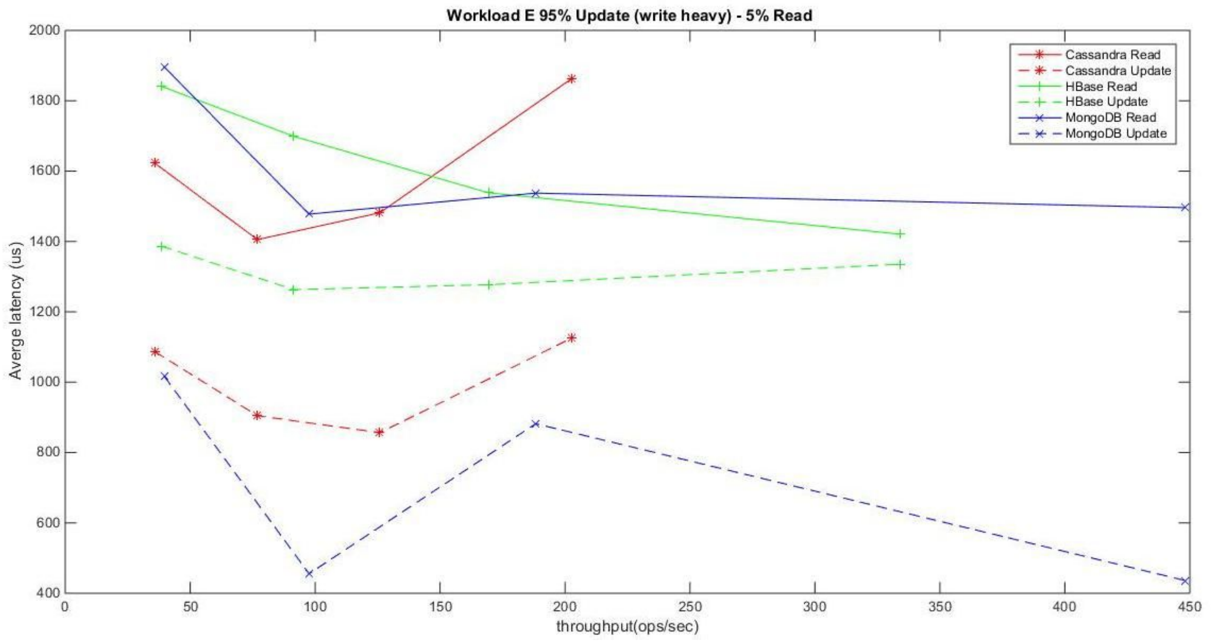
95% READ (HEAVY READ) & 5% INSERT

The dotted lines indicate Inserts while the thick lines indicate Reads. Clearly it is seen that the performance of MongoDB is better than Cassandra and Hbase for both Reads and Inserts. We can clearly see that the Insert operations than the Read and Update operations. However, for MongoDB the Insert operations work almost as fast as the Read indicating that MongoDB is best especially for write operations.



95% Updates & 5% READS

The dotted lines indicate Updates while the solid lines indicates Reads. Clearly the performance of updates in the case of MongoDB is much higher than that of the other databases. When the throughput is low, Cassandra performs well in the case of reads. But when the throughput increases the performance of Cassandra sharply declines.



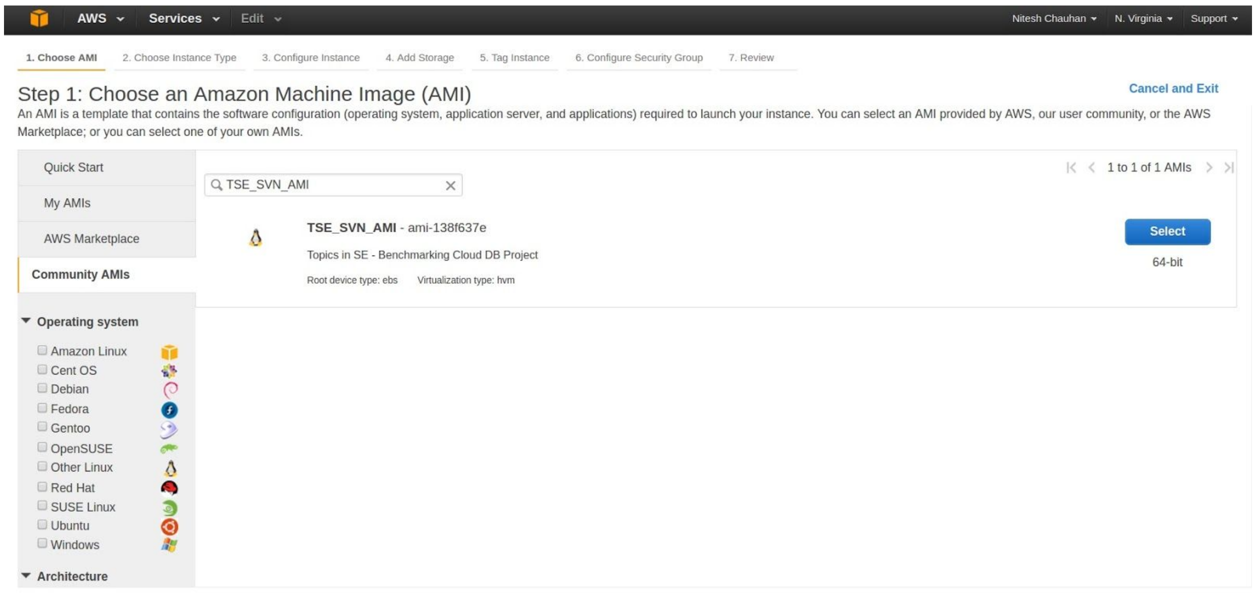
.

Discussions & Conclusions

It is seen that MongoDB performs better than Cassandra and HBase. This could be attributed to the fact that MongoDB uses Multi - granularity locking. Multi -granularity locking allows one to lock operations at the global, database, or collection level. Since an intent shared lock and an intent exclusive lock intend to read or write at a finer granularity, operations are non blocking and thus we see high throughput and low response times.

From the analysis section, we can also observe that both MongoDB and Cassandra perform well on reads where the hot data set fits in memory. Both also emphasize join-less data models (and encourage denormalization instead), and both provide indexes on documents or rows, although MongoDB's indexes are currently more flexible.

Deliverables



Step 1: Choose an Amazon Machine Image (AMI)

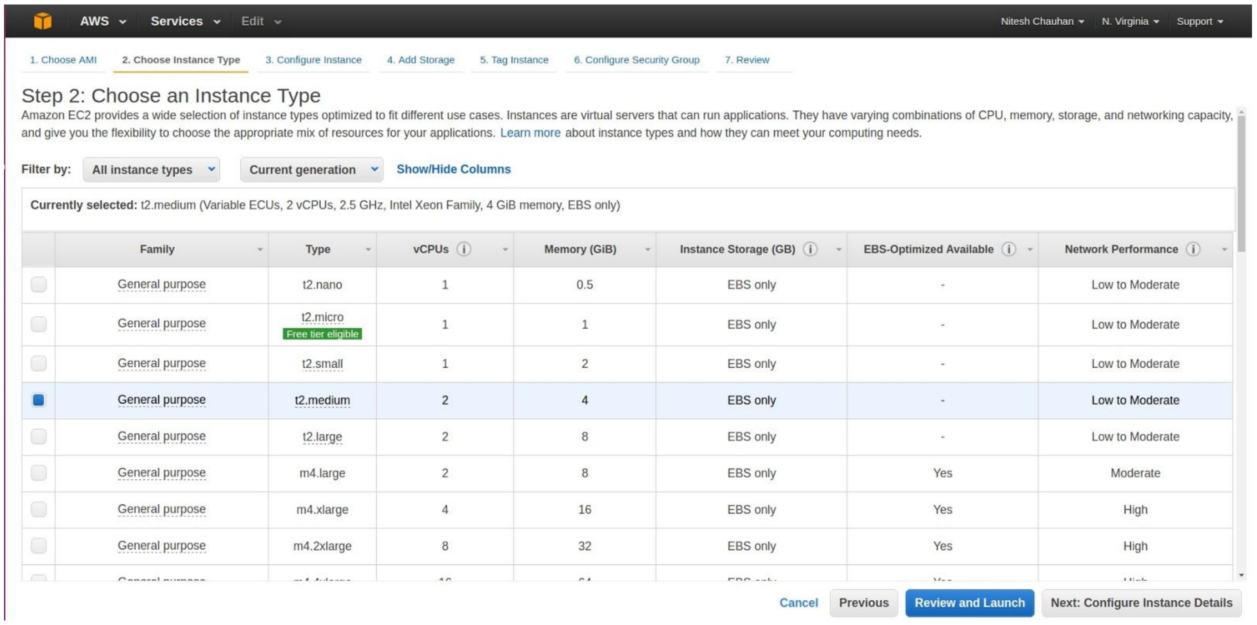
An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

TSE_SVN_AMI

TSE_SVN_AMI - ami-138f637e

Topics in SE - Benchmarking Cloud DB Project

Root device type: ebs Virtualization type: hvm



Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. Learn more about instance types and how they can meet your computing needs.

Filter by: All instance types | Current generation | Show/Hide Columns

Currently selected: t2.medium (Variable ECUs, 2 vCPUs, 2.5 GHz, Intel Xeon Family, 4 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
☐	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate
☐	General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate
☐	General purpose	t2.small	1	2	EBS only	-	Low to Moderate
■	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate
☐	General purpose	t2.large	2	8	EBS only	-	Low to Moderate
☐	General purpose	m4.large	2	8	EBS only	Yes	Moderate
☐	General purpose	m4.xlarge	4	16	EBS only	Yes	High
☐	General purpose	m4.2xlarge	8	32	EBS only	Yes	High

Our project is available on github.
The following is the link to our repository

https://github.com/niteshch/BenchmarkingNoSQLDB

The README gives us detailed instructions on how to evaluate the performance of Cassandra, HBase and MongoDB under various workloads in real time by accessing the AMI.

References:

1. Kamat, Govind. "YCSB, the Open Standard for NoSQL Benchmarking, Joins Cloudera Labs - Cloudera Engineering Blog." Cloudera Engineering Blog. 2015.
2. "Brianfrankcooper/YCSB." *GitHub*.