|  |
| --- |
| California State University East Bay |
| Hadoop Map-Reduce |
| Ami Pandya  Parth Patel  Computer Science |

**Abstract**

Hadoop is one of the free java based programming framework, which will help to processing large amount of data set from distributed computing systems. Hadoop provides distributed file system called HDFS (Hadoop Distributed File Systems), which stores data across the nodes of the cluster. Mainly Hadoop provides parallel paradigm implementation Map Reduce and is widely used for parallel computation with low response time. Hadoop is used to process extremely large amount of data without bothering of the structure of data. Large data, which means petabytes of Data. The main objective of the project is to measure the performance of the cluster system using Hadoop and implement MapReduce Algorithm.

To measure the performance, we have set up Hadoop cluster with single node or multiple nodes. We have used amazon web server to create Hadoop environment. HDFS performance will be calculated by writing and reading with small and large data sets. We will implement the MapReduce algorithm to extract data efficiently and fast.

**Introduction to Apache Hadoop**

In recent years, data is increasing tremendously in size. Therefore, it is necessary to manipulate data effectively and efficiently. Alongside that, the retrieval of data from such large dataset was also one concerned for some of the big companies such as Facebook, LinkedIn, and Yahoo.

Hadoop has been introduced as an open-source and java-based programming framework that supports the processing of large data sets in a distributed computing environment. Google File System (GFS) papers inspired it. [1] Apache developed Hadoop is now a top level Apache project, being built and used by a community of contributors from all over the world, since it’s easy to install, configure and can be run on many platforms supporting Java. Major players including Google, Yahoo and IBM, largely for applications involving search engines and advertising currently use the Hadoop framework.

The significant commitments of this work are a Hadoop execution assessment on composing and full comprehension about the MapReduce idea and additionally the dispersion of procedures.

**Key Concepts of Hadoop:**

1. HDFS (Hadoop Distributed File System)
2. MapReduce (Processing /Computation layer)

**HDFS (Hadoop Distributed File System)**

Hadoop accompanies a distributed filesystem called HDFS, which remains for Hadoop Distributed Filesystem. HDFS is a filesystem intended for putting away extensive files with spilling information access patterns, running on clusters on thing equipment. The HDFS is a distributed, adaptable, and compact file-system written in Java for the Hadoop framework. A Hadoop cluster has ostensibly a solitary namenode in addition to a bunch of datanodes, in spite of the fact that excess choices are accessible for the namenode because of its criticality. Each datanode serves up squares of information over the system utilizing a piece convention particular to HDFS. The file system utilizes TCP/IP attachments for correspondence. Customers use remote procedure call (RPC) to impart between one another.

**HDFS Architecture**

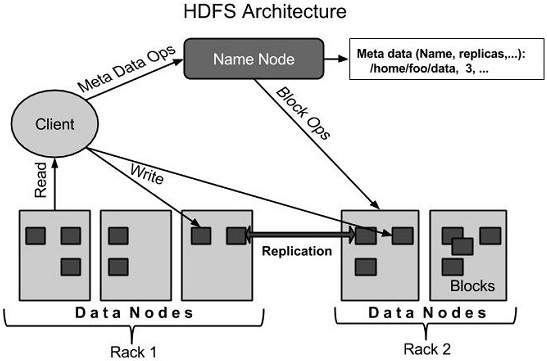


Figure 1: HDFS Architecture Retrieved from http://www.tutorialspoint.com/hadoop/hadoop\_hdfs\_overview.htm

HDFS works on master-slave architecture. HDFS architecture consists of many components such as NameNode, DataNode, JobTracker and TaskTracker is used several clients.

**Client**

Client is a software, which communicate with the name node.

**NameNode**

NameNode is consider as master server, which contains LINUX operating system. It performs the following task:

* Managing files.
* Client Interaction.
* File operation such as renaming, opening and closing.

**DataNode**

DataNode is consider as slave server in the system. They are managed by anyone of system NameNode. It performs following task

* Read and write operation of file as per client requirement.
* Managing the blocks.

**JobTracker**

Job tracker is the master process, which runs on separate node. It performs the following task.

* Client job is handled.
* It determines location of the data.
* Task allocation to the chosen Task tracker nodes.
* Task tracker is monitored by namenode. Task tracker notifies the name node with results of the job assigned.
* It provides the information to the client.

Job Tracker is only the point of failure in whole Hadoop system.

**Task Tracker**

Task tracker is slave process, which accepts the task from the Job tracker. Task tracker runs on separate JVM because during the failure it does not affect the system. It calculates and captures the output. It notifies the job tracker regarding its status and result of the executed task.

**MapReduce:**

MapReduce is a framework for processing [parallelizable](https://en.wikipedia.org/wiki/Parallel_computing) problems across huge datasets using a large number of computers (nodes), collectively referred to as a [cluster](https://en.wikipedia.org/wiki/Computer_cluster) (if all nodes are on the same local network and use similar hardware) or a [grid](https://en.wikipedia.org/wiki/Grid_Computing) (if the nodes are shared across geographically and administratively distributed systems, and use more heterogenous hardware). Processing can occur on data stored either in a [filesystem](https://en.wikipedia.org/wiki/Filesystem) (unstructured) or in a [database](https://en.wikipedia.org/wiki/Database) (structured). MapReduce can take advantage of the locality of data, processing it near the place it is stored in order to reduce the distance over which it must be transmitted.

* **"Map" step:** Each worker node applies the "map()" function to the local data, and writes the output to a temporary storage. A master node ensures that only one copy of redundant input data is processed.
* **"Shuffle" step:** Worker nodes redistribute data based on the output keys (produced by the "map()" function), such that all data belonging to one key is located on the same worker node.
* **"Reduce" step:** Worker nodes now process each group of output data, per key, in parallel.

MapReduce allows for distributed processing of the map and reduction operations. Provided that each mapping operation is independent of the others, all maps can be performed in parallel – though in practice this is limited by the number of independent data sources and/or the number of CPUs near each source. Similarly, a set of 'reducers' can perform the reduction phase, provided that all outputs of the map operation that share the same key are presented to the same reducer at the same time, or that the reduction function is [associative](https://en.wikipedia.org/wiki/Associative_property). While this process can often appear inefficient compared to algorithms that are more sequential (because multiple rather than one instance of the reduction process must be run), MapReduce can be applied to significantly larger datasets than "commodity" servers can handle – a large [server farm](https://en.wikipedia.org/wiki/Server_farm) can use MapReduce to sort a [petabyte](https://en.wikipedia.org/wiki/Petabyte) of data in only a few hours.[[12]](https://en.wikipedia.org/wiki/MapReduce#cite_note-12) The parallelism also offers some possibility of recovering from partial failure of servers or storage during the operation: if one mapper or reducer fails, the work can be rescheduled – assuming the input data is still available.

**MapReduce Architecture**

The computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user of the MapReduce library expresses the computation as two functions: Map and Reduce.

Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The Map Reduce library groups together all intermediate values associated with the same intermediate key I and passes them to the Reduce function.

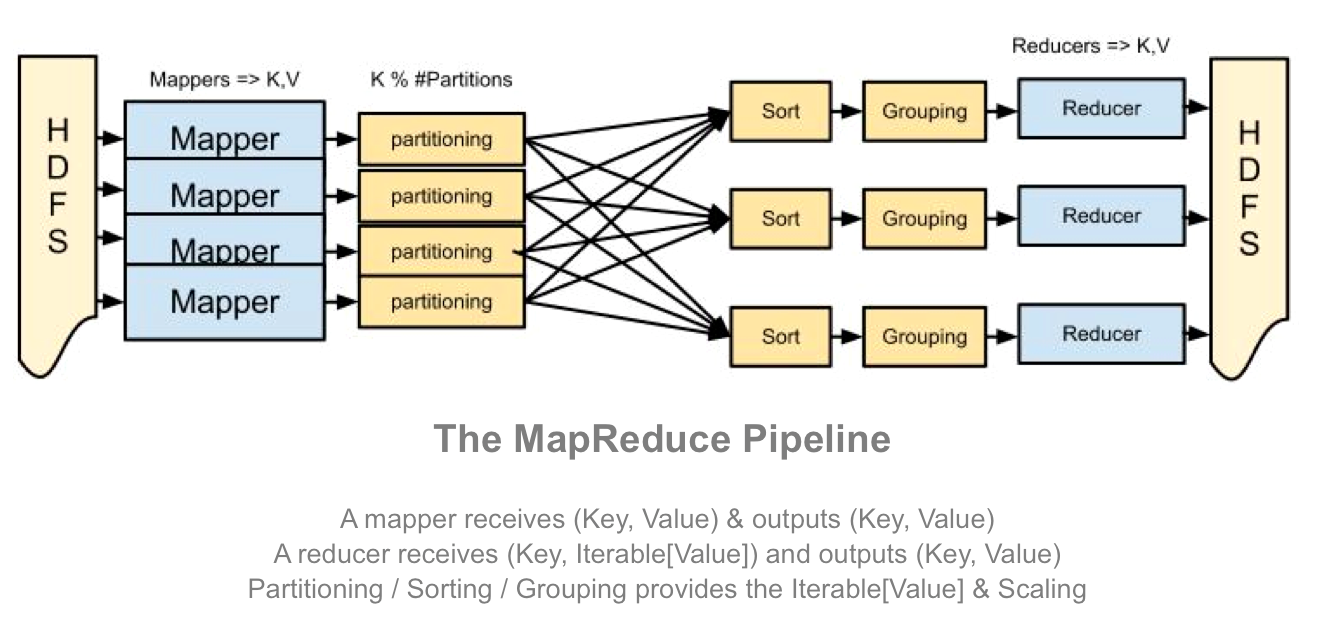
****

Fig 2: MapReduce Architecture

Reduce, also written by the user, accepts an intermediate key I and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically, just zero or one output value is produced per Reduce invocation. The intermediate values are supplied to the users reduce function via an iterator. This allows us to handle lists of values that are too large to in memory.

Hadoop divides the input files stored on HDFS into splits and assigns every split to a different mapper, trying to assign every split to the mapper where the split physically resides. MapReduce basically breaks the data flow into two phases: Map phase & Reduce phase.

**Task Scheduling in Hadoop:**

*Job Tracker:* is the master of the system which manages the jobs and resources in the cluster (Task Trackers). The Job Tracker tries to schedule each map as close to the actual data being processed i.e. on the Task Tracker which is running on the same Datanode as the underlying block.

*Task Trackers:* are the slaves which are deployed on each machine. They are responsible for running the map and reduce tasks as instructed by the Job Tracker.

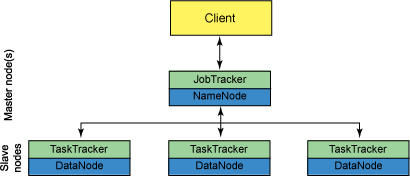
****

Fig 3: Job Tracker and Task Tracker

*Job Tracker:* Every task tracker sends a heartbeat message to Job tracker encompassing a request for a map or a reduce task to run.

*Map Task Scheduling:* Job tracker satisfies requests for map tasks via attempting to schedule mappers in the vicinity of their input splits (i.e., it considers locality)

*Reduce Task Scheduling:* However, Job tracker simply assigns the next yet-to-run reduce task to a requesting Task tracker regardless of Task tracker’s network location and its implied effect on the reducer’s shuffle time (i.e., it does not consider locality)

*MapReduce in Hadoop comes with a choice of schedulers:*

Default scheduler the FIFO scheduler which schedule jobs in order of submission. There is also a multi-user scheduler called the Fair scheduler which aims to give every user a fair share of the cluster capacity over time.

**Fault Tolerance**

**There are two Common Types of Failures in HDFS:**

* Name Node failures
* Data Node failures

**Data Node Failure**

* Each Data Node sends a heartbeat message to the Namenode periodically.
* If the Namenode does not receive a heartbeat from a particular data node for 10 minutes, then it considers that data node to be dead/out of service.
* Name Node initiates replication of blocks which were hosted on that data node to be hosted on some other data node.

**Name Node Failure**

* Single Name Node per cluster.
* If Name Node becomes unavailable, the cluster as a whole would be unavailable.
* Namenode has to be restarted
* Brought up on a separate machine.

**Step by Step Tutorial:**

**1: Install Java**

sudo apt-get install oracle-java7-installer

**2: Add entry of master and slaves in hosts file**

sudo nano /etc/hosts

MASTER-IP master

SLAVE01-IP slave-01

SLAVE02-IP slave-02

**3: Configure SSH**

3.1: Install Open SSH Server-Client

sudo apt-get install openssh-server openssh-client

3.2: Configure password-less SSH

Copy the contents of “$HOME/.ssh/id\_rsa.pub” of master to “$HOME/.ssh/authorized\_keys”all the slaves.

3.3: Check by SSH to slaves

ssh slave-01

ssh slave-02

**4: Download Hadoop**

<http://archive.cloudera.com/cdh/3/hadoop-0.20.2-cdh3u6.tar.gz>

**5: Install Hadoop**

5.1: Untar Tar ball

tar xzf hadoop-0.20.2-cdh3u6.tar.gz

5.2: Go to HADOOP\_HOME\_DIR

cd hadoop-0.20.2-cdh3u6/

**6: Setup Configuration**

6.1: Edit configuration file conf/hadoop-env.sh and set JAVA\_HOME

export JAVA\_HOME=path to be the root of your Java installation(eg: /usr/lib/jvm/jdk1.7.0\_65)

6.2: Edit configuration file conf/core-site.xml and add following entries

<configuration>

<property>

<name>fs.default.name</name>

<value>hdfs://master:9000</value>

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>/home/hadoop\_admin/hdata/hadoop-${user.name}</value>

</property>

</configuration>

6.3: Edit configuration file conf/hdfs-site.xml and add following entries

<configuration>

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

</configuration>

6.4: Edit configuration file conf/mapred-site.xml and add following entries

<configuration>

<property>

<name>mapred.job.tracker</name>

<value>master:9001</value>

</property>

</configuration>

6.5: Edit configuration file conf/masters and add entry of secondary-master

slave-01

6.6: Edit configuration file conf/slaves and add entry of slaves

slave-01

slave-02

6.7: Set environment variables Update ~/.bashrc and set or update the HADOOP\_HOME and PATH shell variables as follows

nano ~/.bashrc

export HADOOP\_HOME=/home/hadoop/hadoop-0.20.2-cdh3u6

export PATH=$PATH:$HADOOP\_HOME/bin

*Hadoop is setup on master*

**7: Setup Hadoop on slaves**

7.1: Repeat the step-1 and step-2 on all the slaves

7.2: Create tar ball of configured Hadoop-setup and copy to all the slaves

tar czf hadoop.tar.gz hadoop-0.20.2-cdh3u6

scp hadoop.tar.gz slave01:~

scp hadoop.tar.gz slave02:~

7.3: Untar configured Hadoop-setup on all the slaves

tar xzf hadoop.tar.gz

**8: Start The Cluster**

8.1: Format the name node

$bin/hadoop namenode –format

**8.2: Start Map-Reduce services**

$bin/start-mapred.sh

**Run the word count test:**

hadoop jar hadoop-examples-0.20.2-cdh3u6.jar wordcount inp out

**Test Results:**

**Write:**

*File size = 1.2 GB, Block size = 64 MB*

Throughput (mb/second) 88.61

Average I/O rate (mb/second) 88.61

Execution time (seconds) 18.65

nrFilesize = 1, Replication = 2

Throughput (mb/second) 70.72

Average I/O rate (mb/second) 70.72

Execution time (seconds) 19.373

nrFilesize = 3, Replication = 1

Throughput (mb/second) 85.50

Average I/O rate (mb/second) 85.50

Execution time (seconds) 45.31

*File size = 1.2 GB, Block size = 128 MB*

nrFilesize = 1, Replication = 1

Throughput (mb/second) 77.13

Average I/O rate (mb/second) 77.13

Execution time (seconds) 18.23

nrFilesize = 1, Replication = 2

Throughput (mb/second) 125.091

Average I/O rate (mb/second) 125.091

Execution time (seconds) 18.40

nrFilesize = 3, Replication = 1

Throughput (mb/second) 99.69

Average I/O rate (mb/second) 99.24

Execution time (seconds) 40.47

**Read**

*File size = 1.2 GB, Block size = 64 MB*

nrFilesize = 1, Replication = 1

Throughput (mb/second) 400.66

Average I/O rate (mb/second) 400.66

Execution time (seconds) 6.417

nrFilesize = 1, Replication = 2

Throughput (mb/second) 378.78

Average I/O rate (mb/second) 378.78

Execution time (seconds) 6.34

nrFilesize = 3, Replication = 1

Throughput (mb/second) 93.399

Average I/O rate (mb/second) 93.44

Execution time (seconds) 42.39

*File size = 1.2 GB, Block size = 128 MB*

nrFilesize = 1, Replication = 1

Throughput (mb/second) 701.34

Average I/O rate (mb/second) 701.34

Execution time (seconds) 4.28

nrFilesize = 1, Replication = 2

Throughput (mb/second) 580.55

Average I/O rate (mb/second) 580.55

Execution time (seconds) 4.128

nrFilesize = 3, Replication = 1

Throughput (mb/second) 81.25

Average I/O rate (mb/second) 81.28

Execution time (seconds) 47.62

**Frequently Asked Questions:**

**1: What is Apache Hadoop MapReduce?**

**Ans:** Hadoop MapReduce is a product framework for distributed preparing of substantial data sets on process clusters of ware equipment. It is a sub-task of the Apache Hadoop venture. The framework deals with planning errands, checking them and re-executing any failed task.

**2:** **What is HDFS? How it is different from traditional file systems?**

**Ans:** HDFS, the Hadoop Distributed File System, is responsible for storing huge data on the cluster. This is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS is designed to support very large files. Applications that are compatible with HDFS are those that deal with large data sets. These applications write their data only once but they read it one or more times and require these reads to be satisfied at streaming speeds. HDFS supports write-once-read-many semantics on files.

**3: What sort of data can Hadoop handle?**

**Ans**: Hadoop can handle the organized, semi organized and unstructured data.

**4: Does High level programming support in Hadoop?**

**Ans**: Yes, High level inquiries based on the Hadoop Hive, pig and SPARK.

## 5: How Job Tracker schedules a task?

**Ans:** The Task Trackers convey heartbeat messages to the Job Tracker, generally like clockwork, to console the Job Tracker that it is still alive. These message likewise advise the Job Tracker of the quantity of accessible openings, so the Job Tracker can stay up and coming with where in the cluster work can be appointed. At the point when the Job Tracker tries to discover some place to plan an errand inside of the Map Reduce operations, it first searches for a void space on the same server that has the Data Node containing the data, and if not, it searches for an unfilled opening on a machine in the same rack.

## 6: How Name Node Handles data node failures?

**Ans:** Once Name node realize that data node is out of service then system replace the backup data node to failure data node and continue execution.

**7: Does Map Reduce programming model provide a way for reducers to communicate with each other? In a Map Reduce job can a reducer communicate with another reducer?**

**Ans**: Nope, Map Reduce programming model does not allow reducers to communicate with each other. Reducers run in isolation.

**Analysis:**

Through this Project we got the opportunity to have a more critical take a gander at the usefulness of Apache Hadoop. We realize that to keep up a framework that gives exact results in a small amount of second, it takes top to bottom information of the details, programming and equipment. This is the point at which we can acknowledge how Hadoop makes it truly simple for us. It gives the file system and gives us different feature like Hive, Pig, Hbase and MapReduce.

The investigation we performed include the working of Apache Hadoop usage and examining its engineering against that of other real framework and file system. Additionally we built up a "Word Count".

**Application Implementation:**

In this part, first we list out the tools used for developing a simple application. Then will describe on how it is implemented.

**Development tools:**

* OS used: Ubuntu 14.04LTS
* Platform: Amazon web servers
* Hadoop 2.7.1
* JAVA jdk1.7.0\_75
* Hypervisor enable local machine
* Multi clusters on Amazon web server (hadoop installed)

**Study Experiences:**

In the wake of going to a quarter of Distributed Systems Course it was vital that we had hands on experience on how the systems really function. We could accomplish this objective by adding to this undertaking. Apache Hadoop has the features of the HDFS and MapReduce which are the core component of the Hadoop Architecture. Here, we learnt, how to implement the Single node and Multi node cluster Hadoop in the Ubuntu operating system. We installed hadoop 2.7.1 on the Ubuntu 14.04 LTS using amazon web server. After performing the Word Count example on the Hadoop, we got to know about the actual working of the HDFS and MapReduce. Multi node cluster takes less time for the processing compare to the Single node. Apache Hadoop is a free open source framework which is used by yahoo, Facebook and many giant companies. Task scheduling done by the Job tracker and Task tracker. Job Tracker assigns works to the related Task Tracker. Namenode and Datanode are the part of the Hadoop Distributed File System. In MapReduce, Mapper, combiner and reducer perform the task by the value and key as a component. One of the major studies that we acquired was through developing a single node hadoop using amazon web server platform. There were many cases when the deployment failed or the build was unsuccessful. But rectifying all of them by careful analysis and inquiry among peers had been a fruitful experience. We come to know about how different cluster communicates with each other in the real time as well as in virtual environment. Finally, it can be inferred that Apache Hadoop is backed by impressive infrastructure, and if we are using one of the technologies they support, we have access to many features with plenty of powerful languages.

**Teamwork Details**

Essentially the task was entirely testing keeping in mind the end goal to see how the genuine hadoop map-reduce actualized and it required an aggregate cooperation so as to complete with great exertion. The flow of the project and one of the primary part was completed in gathering by both the colleague together. The primary concern about this was as we gave aggregate endeavors on to it, it provided the best result for the same. When we were done choosing how the real flow of the project will go we separated the work into half and made the having so as to undertake profound comprehension of all the key parts that are been required for the execution of the same. One part that took a shot at the real stream while the other part than worked upon the sub segments that were required as a spine for the task. In this manner, it did required out collective effort in order to present this well enough and we did a good teamwork for the same

**Future Development:**

Since the Hadoop scheduler is pluggable, you ought to see new schedulers created for one of a kind cluster organizations. Two in-procedure schedulers (from the Hadoop issues list) incorporate the versatile scheduler and the learning scheduler. The learning schedulers intended to keep up a level of use when given a various arrangement of workloads. At present, this scheduler usage concentrates on CPU load midpoints, yet use of system and plate I/O is arranged. The versatile scheduler concentrates on adaptively changing assets for a given employment in view of its execution and client characterized business objectives. It's likewise barely fortuitous event that YARN makes Hadoop significantly more cross-compatible with other Apache ventures for rubbing big data. Utilize one, and it gets to be far less demanding to utilize the rest. Such a rising tide for Hadoop would lift the greater part of Apache's related boats.

**Snapshots:**

Creating 3 instance on Amazon Web Servers.

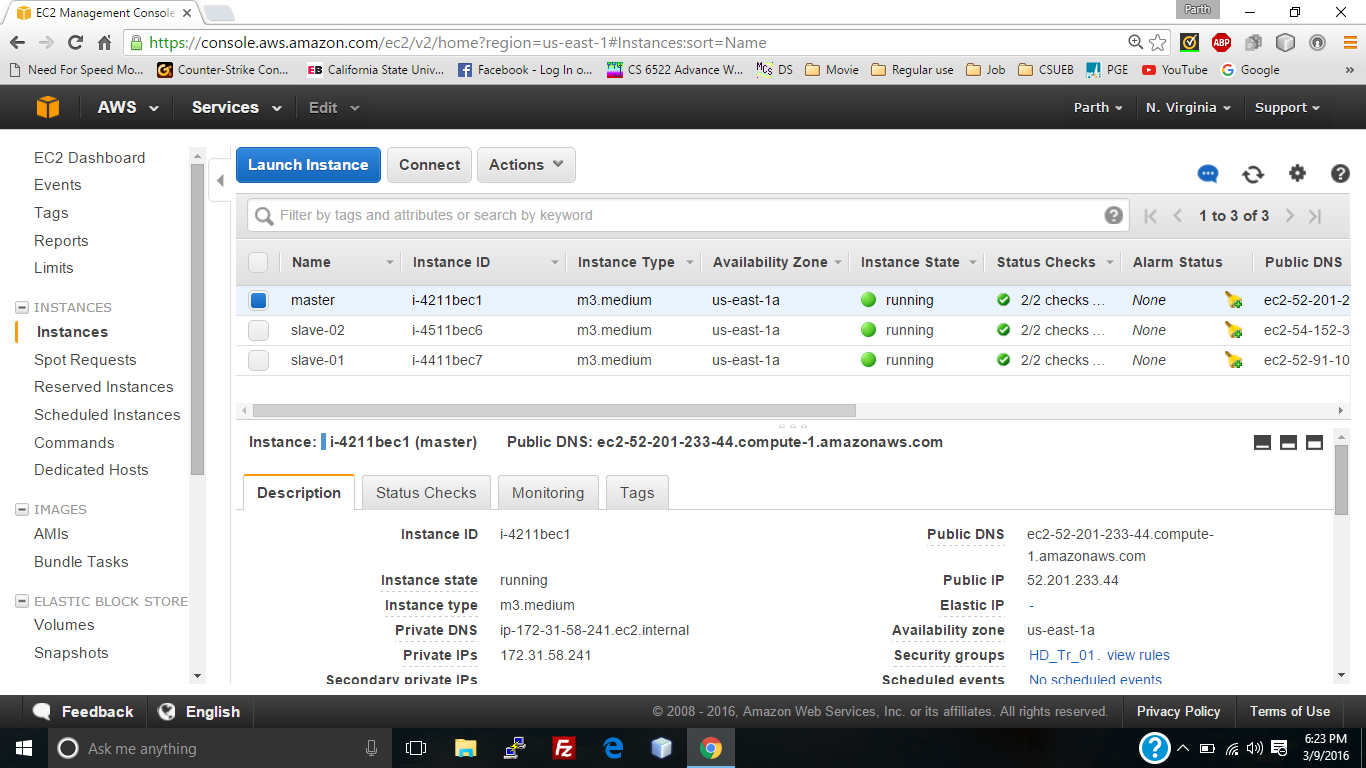


Fig 4

Connect to Master node:

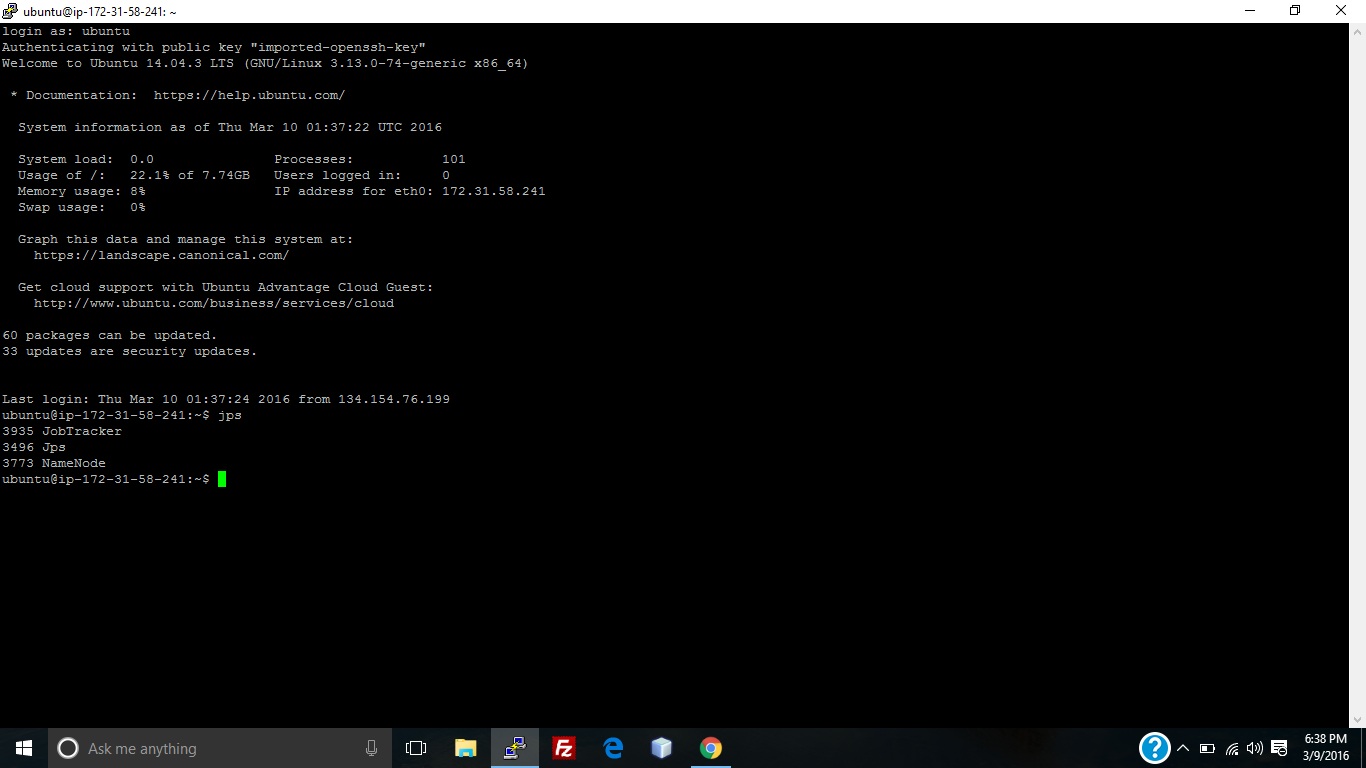


Fig 5

Connect to Slave-01:

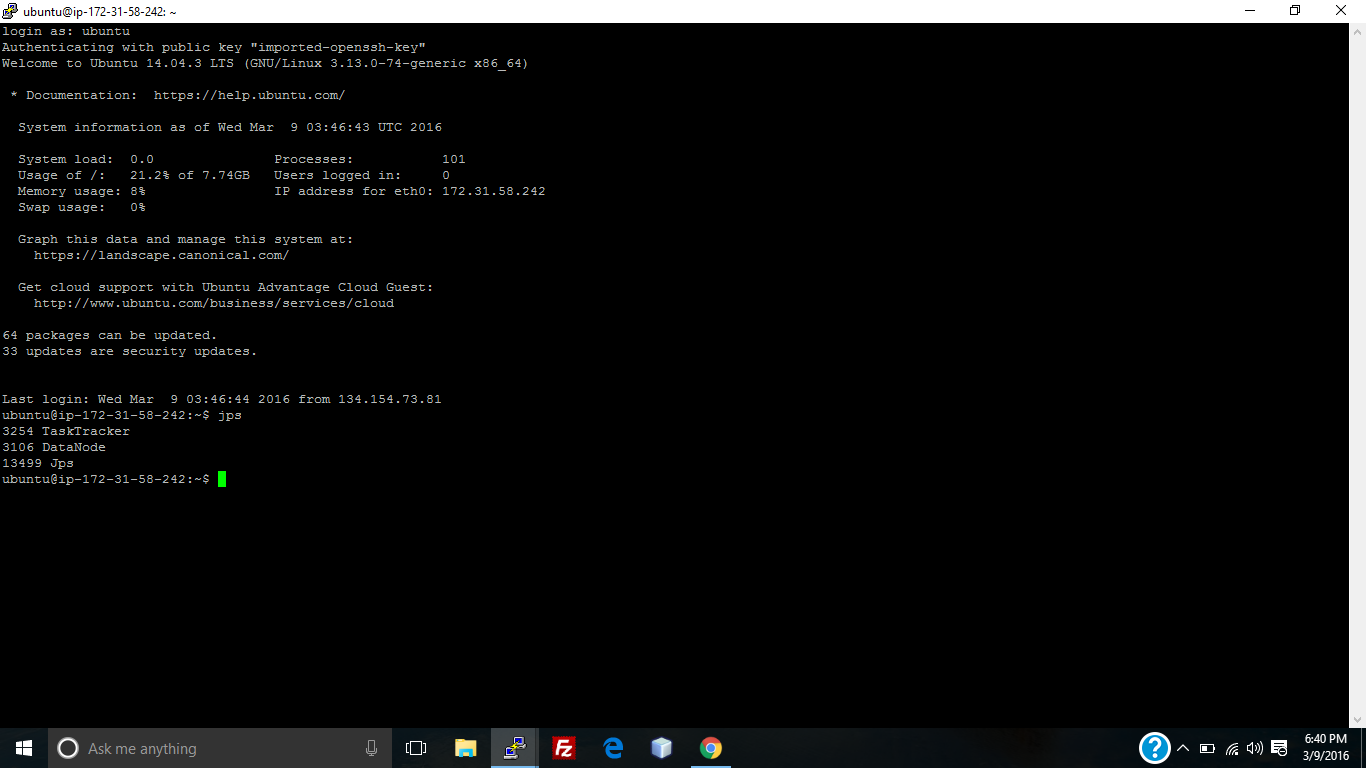


Fig - 6

Connect to Slave-02

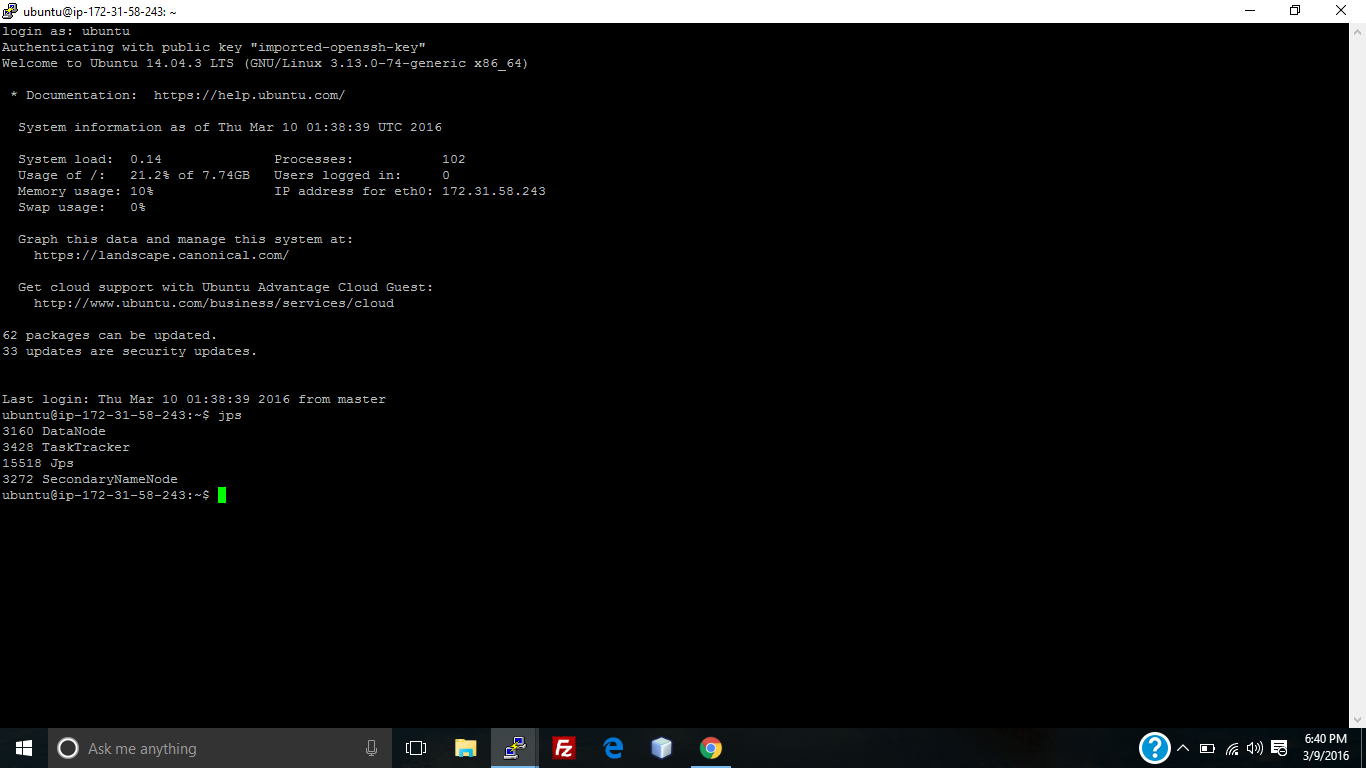


Fig 7

NameNode details:

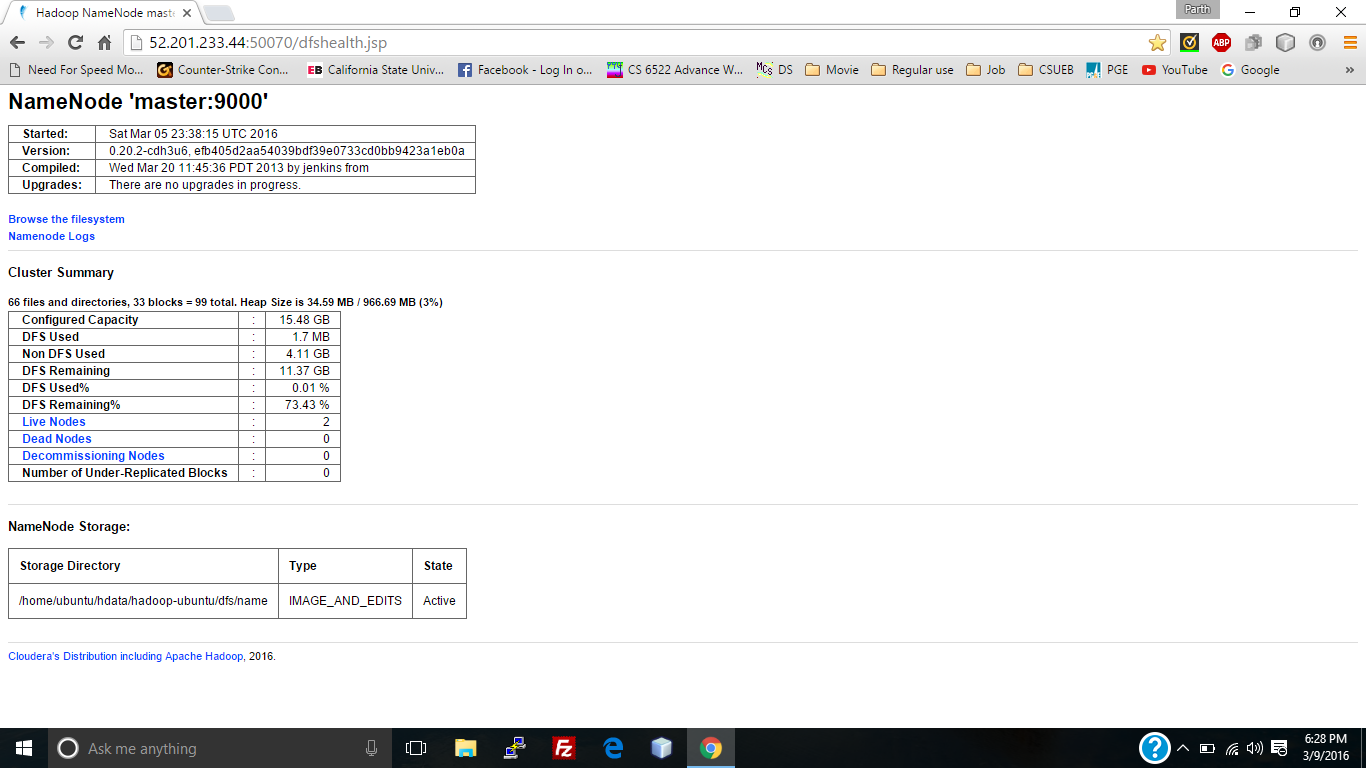


Fig - 8

Input files in HDFS

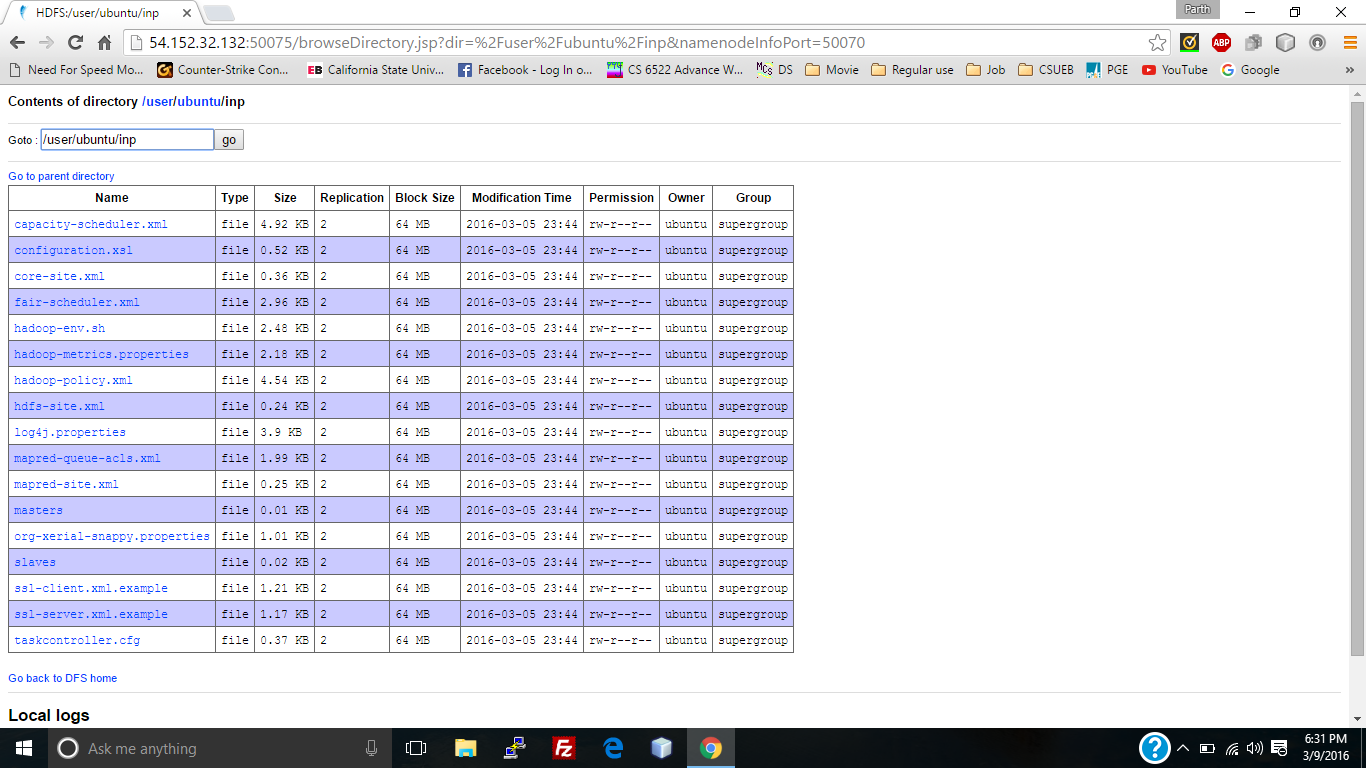


Fig - 9

Run Map-Reduce:

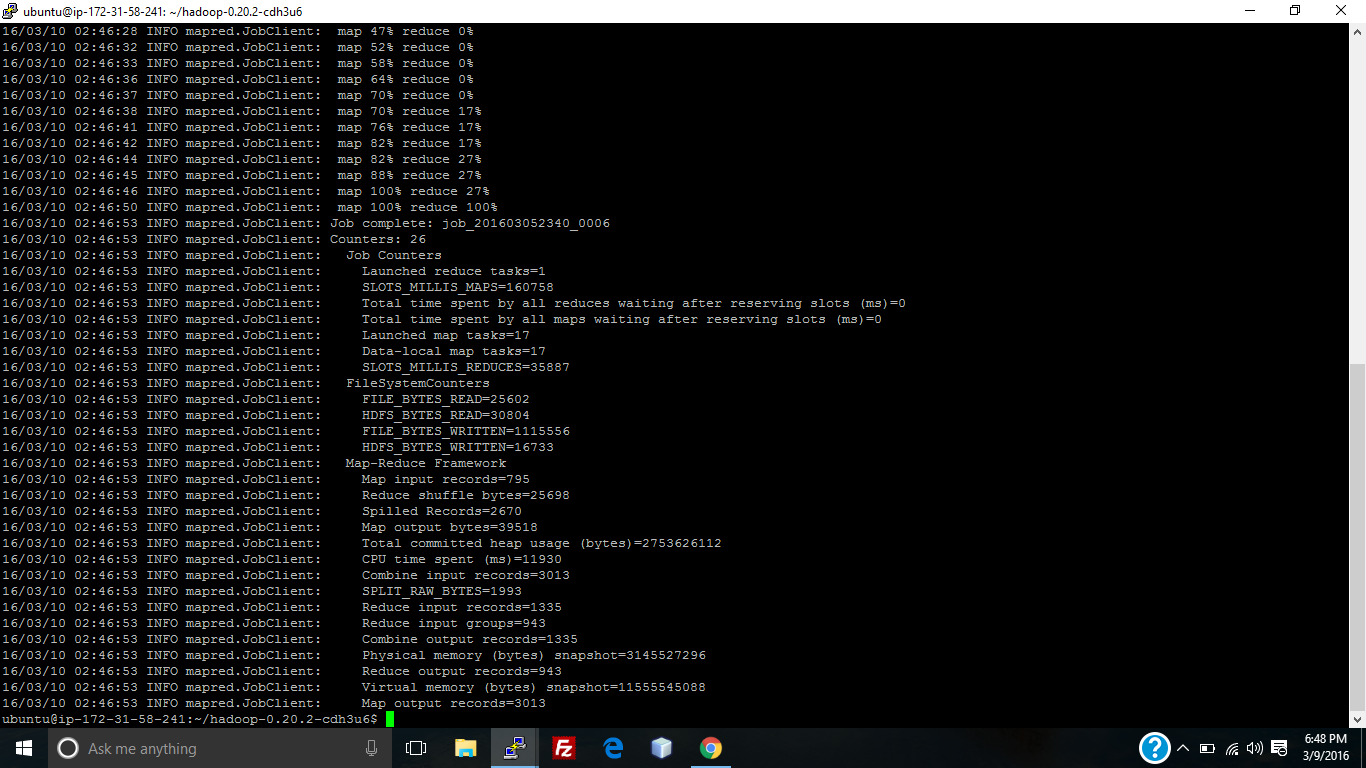


Fig - 10

Job Tracker before map-reduce run:

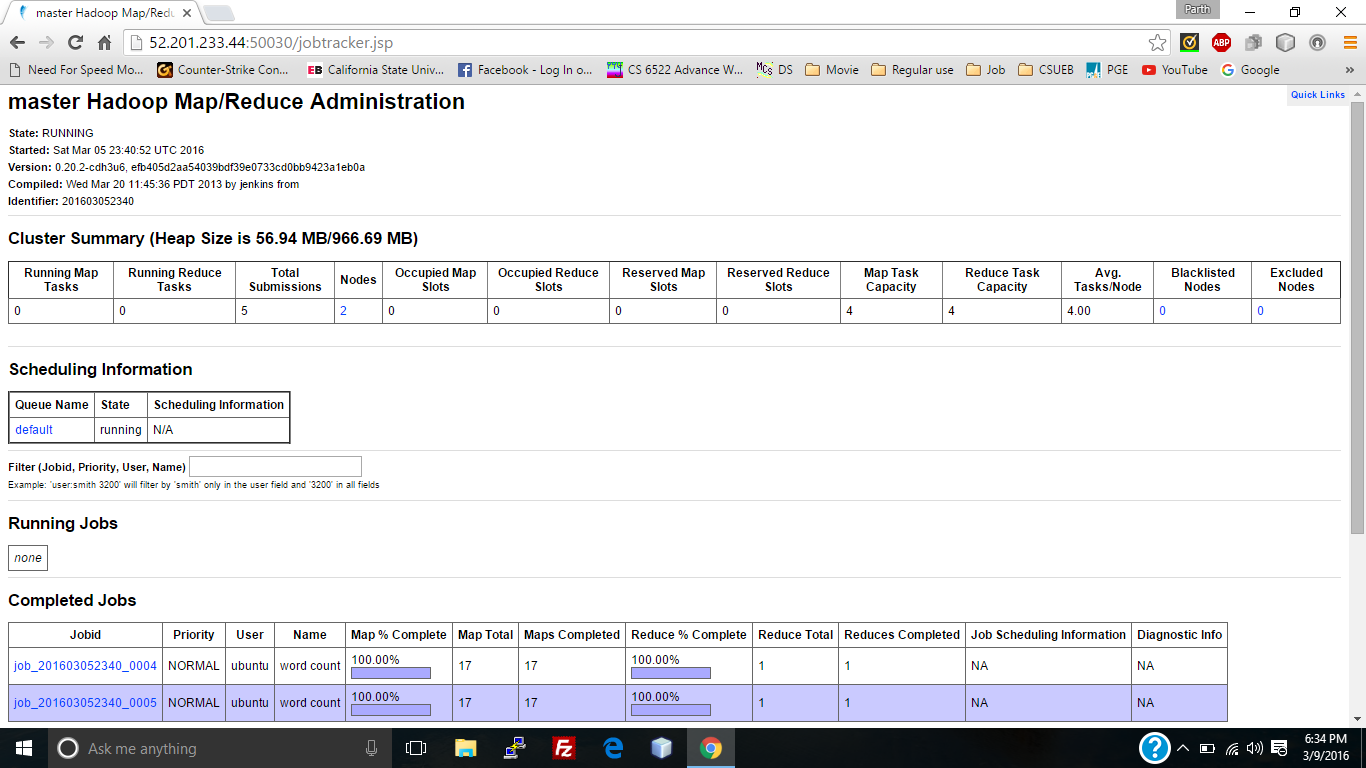


Fig - 11

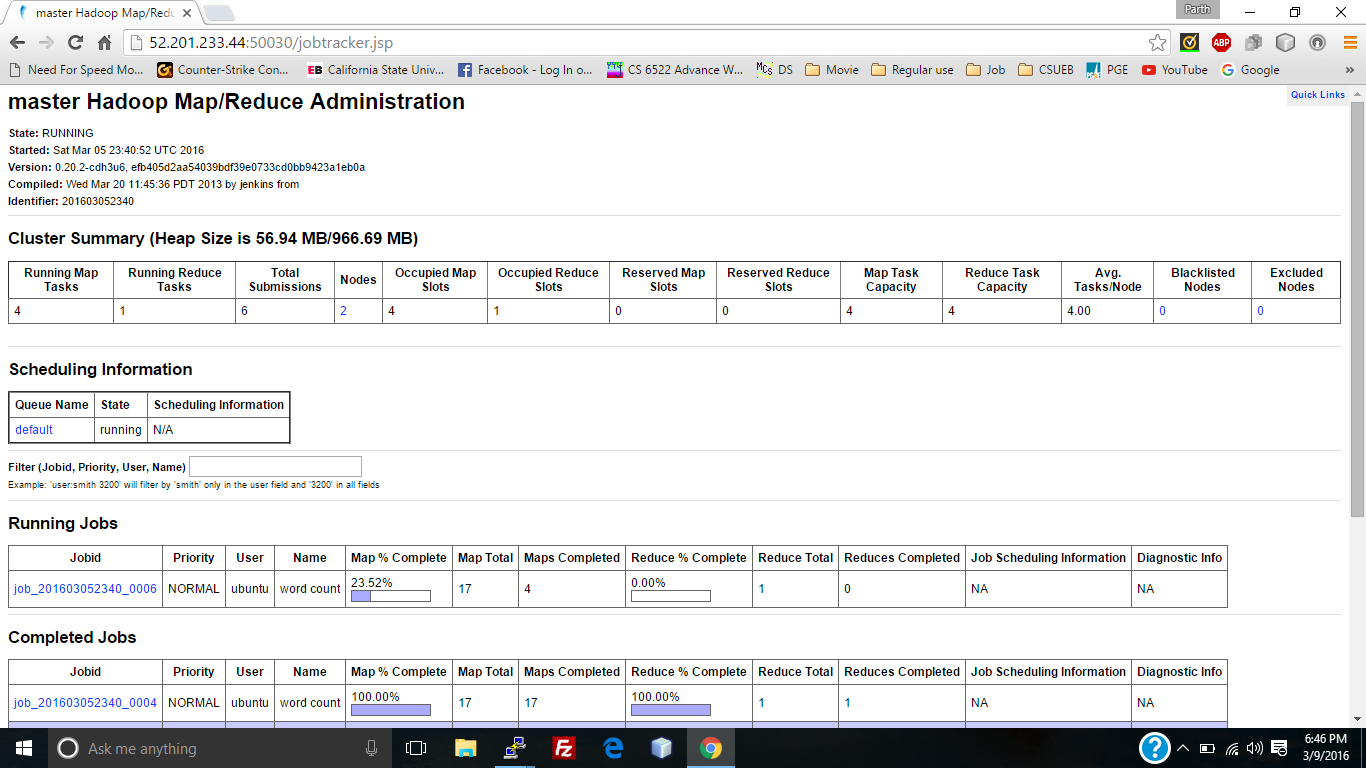


Fig 12

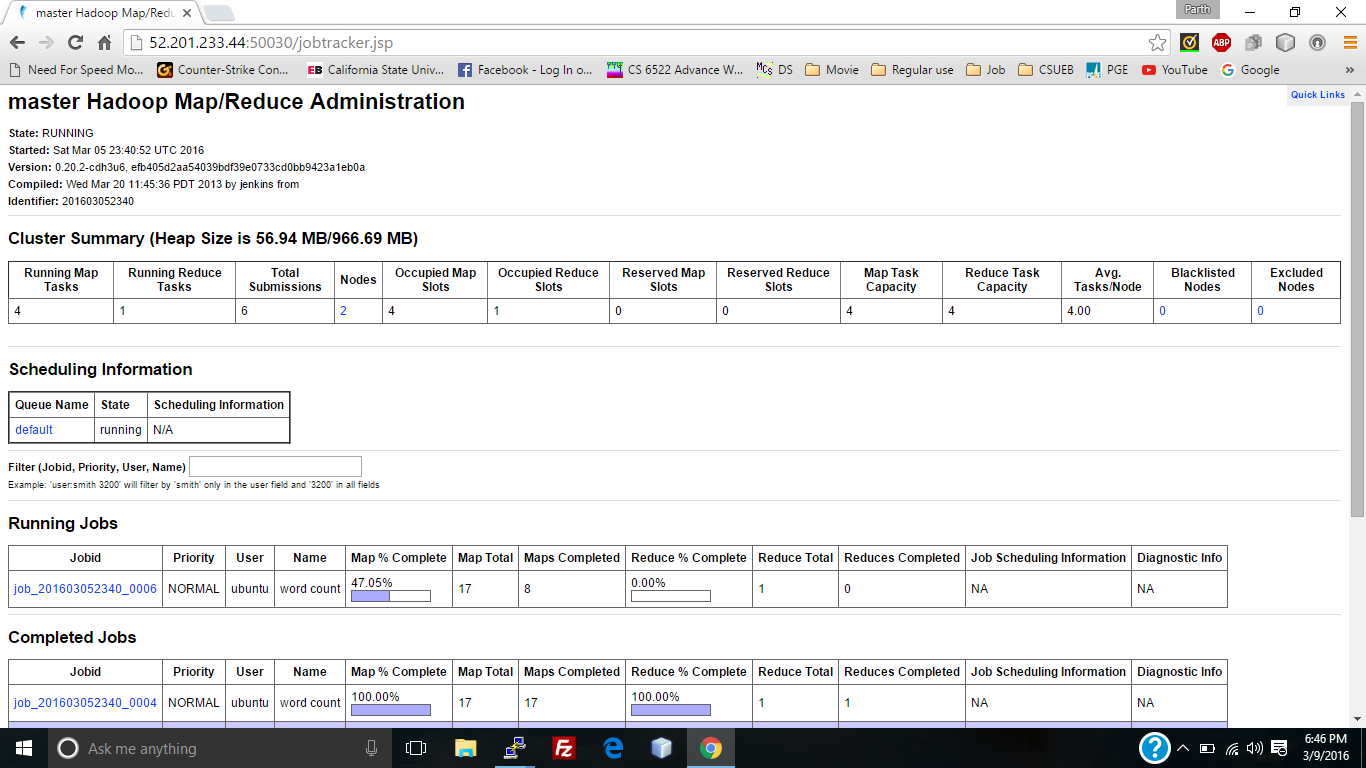


Fig 13

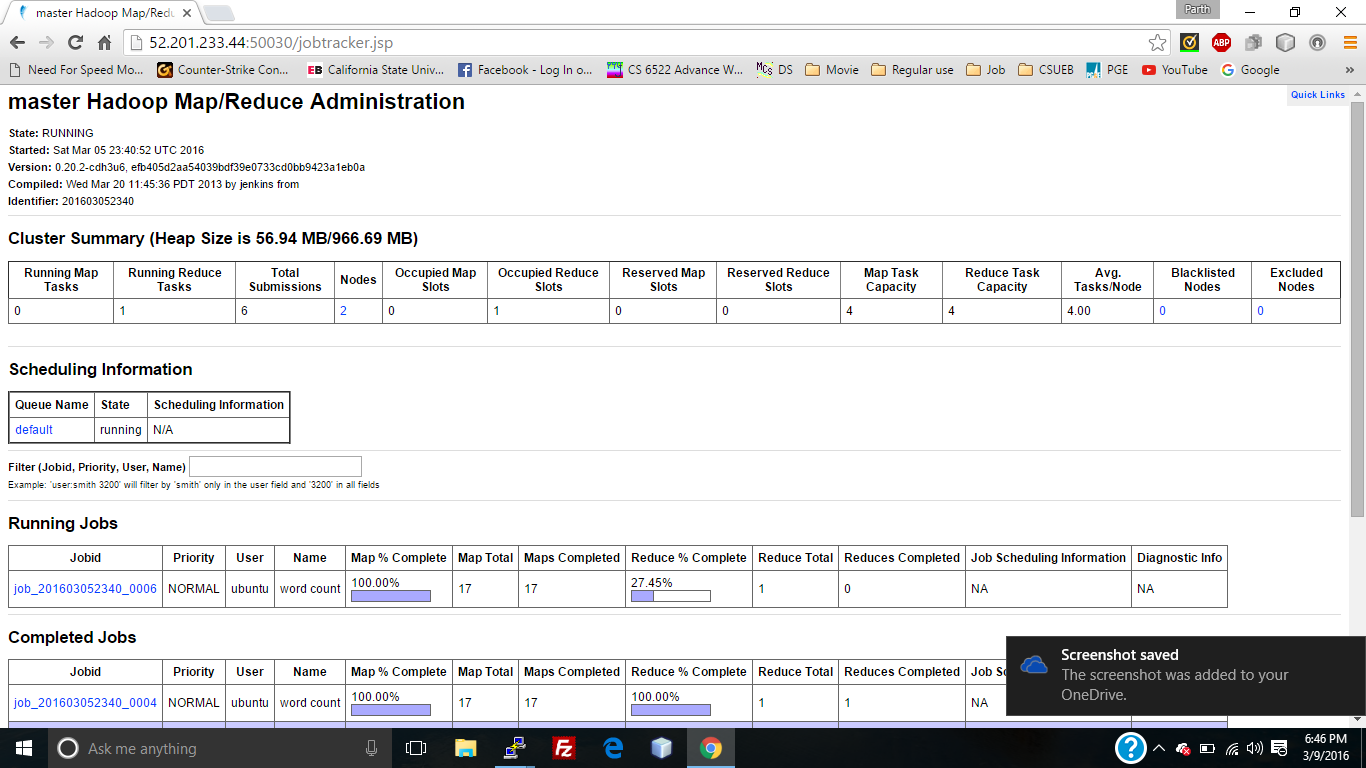


Fig 14

Job Tracker after map-reduce completion:

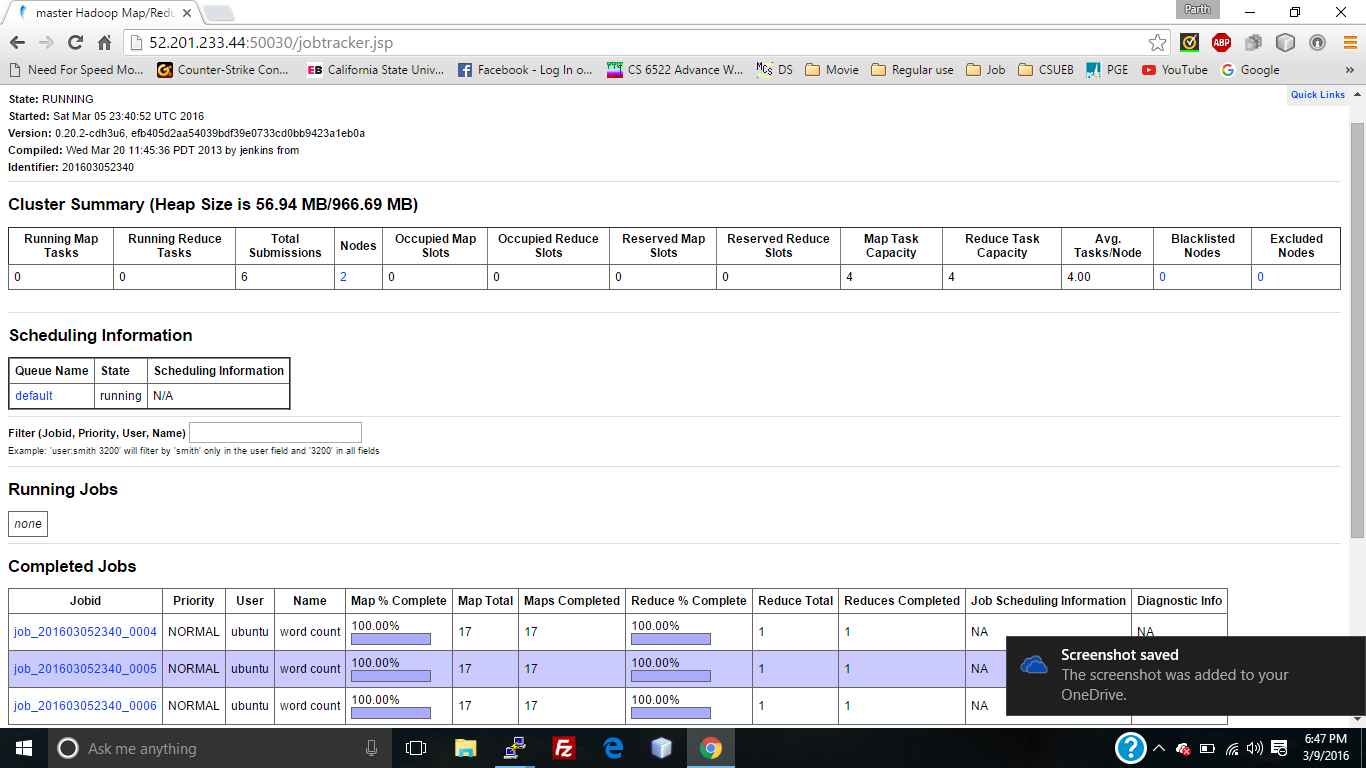


Fig 15

Job detail:

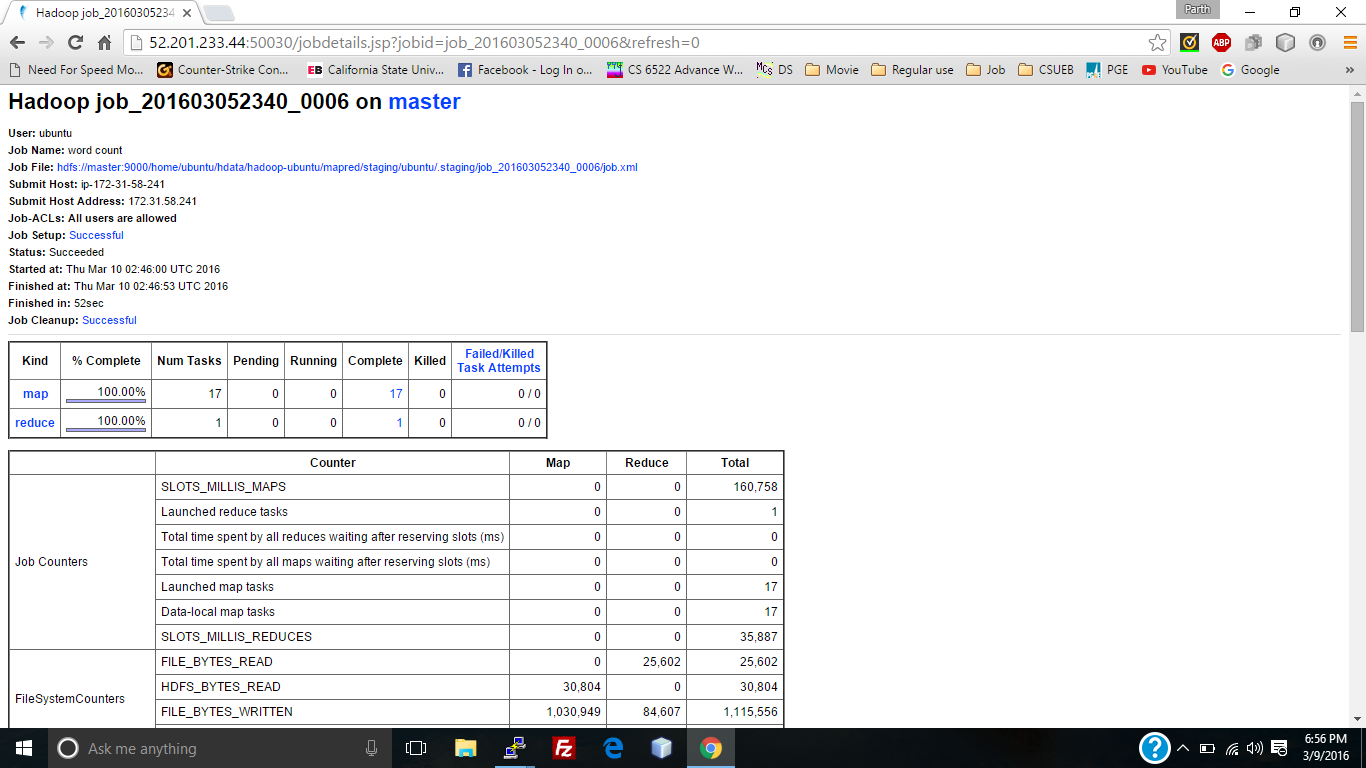


Fig 16

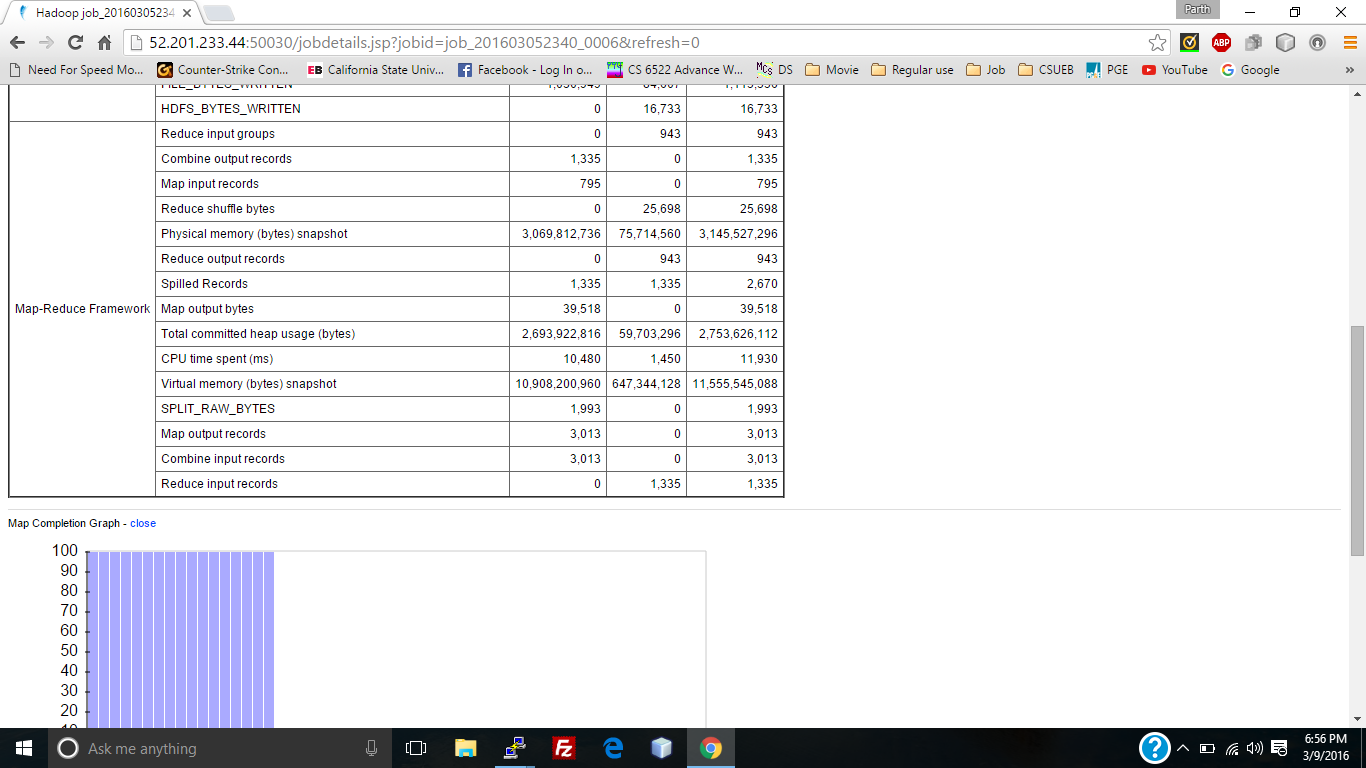


Fig 17



Fig 18

Output:

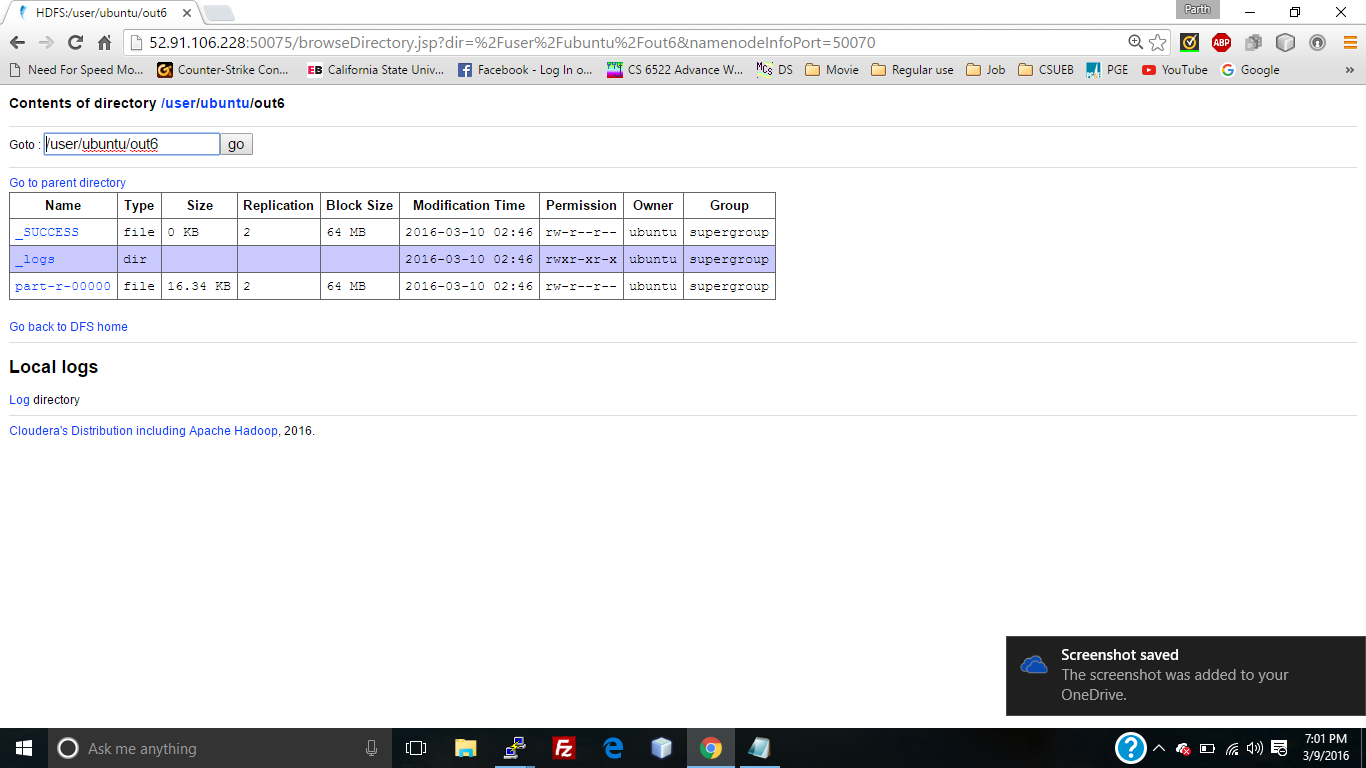


Fig 19

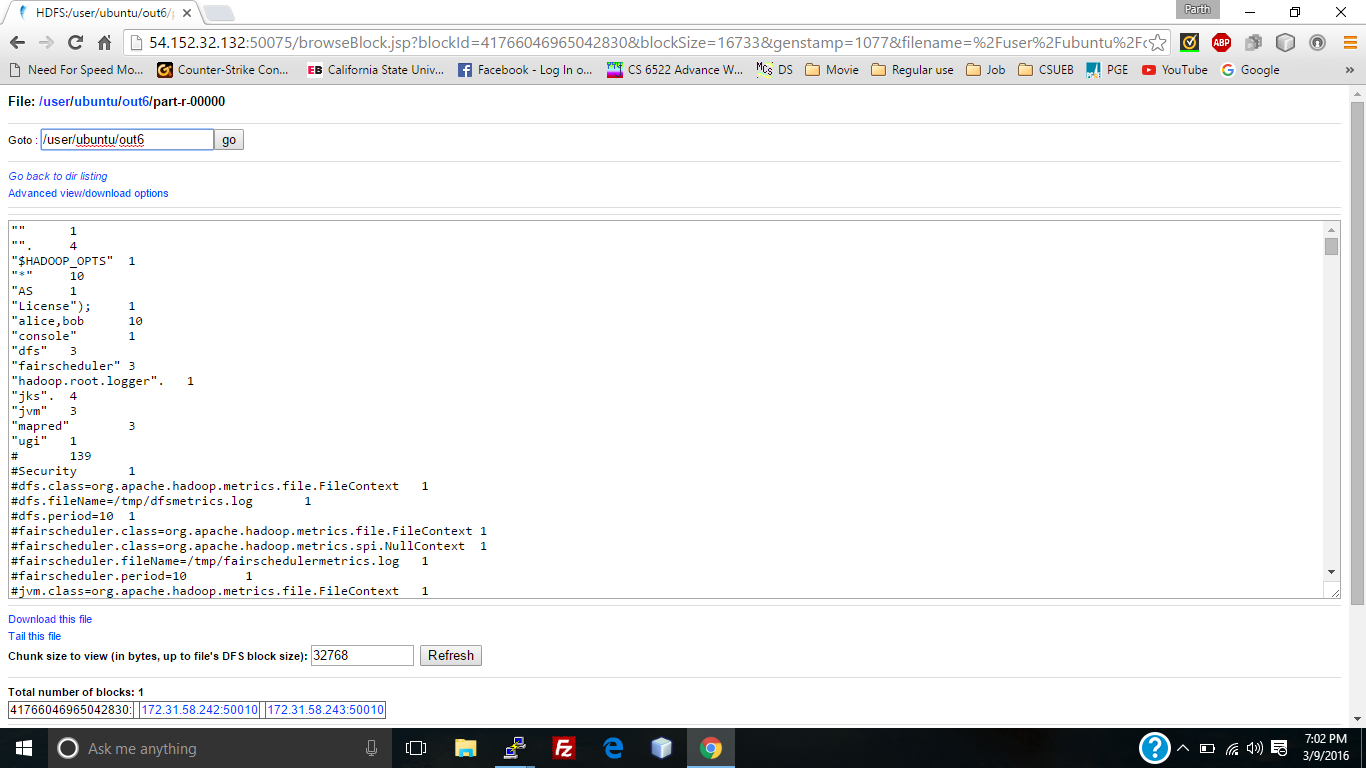


Fig 20

**References:**

* <https://en.wikipedia.org/wiki/Apache_Hadoop>
* <https://hadoop.apache.org/>
* <http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/>
* Hadoop: The Definitive Guide by Tom White
* <http://docs.hortonworks.com/HDPDocuments/HDP1/HDP-1.2.4/bk_getting-started-guide/content/ch_hdp1_getting_started_chp2_1.html>
* <http://www.ibm.com/developerworks/library/os-hadoop-scheduling/>
* <https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html>
* <http://www.bogotobogo.com/Hadoop/BigData_hadoop_Install_on_ubuntu_single_node_cluster.php>
* <https://aws.amazon.com>
* http://blogs.data-flair.com/install-hadoop-1-x-on-multi-node-cluster/