UNIVERSITY OF TEXAS ARLINGTON | DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROJECT 3 -REPORT DISTRIBUTED

SYSTEMS CSE-5306

Instructor – DR. JIA RAO

Table of Contents:

Date: 16th December, 2017

Assignment-1 (40pts): Set up a realistic big data processing environment. Deploy Apache YARN in a Linux environment. YARN should be deployed in the pseudo-distributed mode, in which the daemons are run on a single node as separate process. Download the stable Hadoop release 2.7.2. See the installation guide for reference. Note that you need to configure YARN to specify the replication factor and the block size of HDFS, the location of YARN daemons, and other YARN configurations. Once YARN is up and running, install the Intel Big Data benchmarks HiBench in YARN. 1) Show the successful completion of at least one MapReduce and one Spark job from the HiBench benchmarks; 2) explore to change the number of mappers, reducers for the MapReduce job, and the number of Spark executors for the Spark job, and study the performance of the jobs with different settings. Explain in your report how you changed the job configuration. Plot the trend of job performance with different number of mappers, reducers and executors.

Solution 1:

HADOOP 2.7.4 INSTALLING ON UBUNTU 14.04 (SINGLE-NODE CLUSTER) PSEUDO DISTRIBUTED MODE

Step 1: Install Java version 8(for Hadoop 2.7.4)

Step 2: Adding a Dedicated Hadoop User

abc@laptop:~$ sudo addgroup hadoop
Adding group ‘hadoop’
abc@laptop:~$ sudo adduser --ingroup hadoop hduser

Step 3: Installing SSH:

ssh has two main components:

1. ssh : The command we use to connect to remote machines - the client.
2. sshd : The daemon that is running on the server and allows clients to connect to the server.

The ssh is pre-enabled on Linux, but in order to start sshd daemon, we need to install sshfirst.

Use this command to do that:

abc@laptop:~$ sudo apt-get install ssh

This will install ssh on our machine. If we get something similar to the following, we can think it is setup properly:

abc@laptop:~$ which ssh

abc@laptop:~$ which sshd

Step 4: Create and Setup SSH Certificates

Hadoop requires SSH access to manage its nodes, i.e. remote machines plus our local machine. For our single-node setup of Hadoop, we therefore need to configure SSH access to localhost.

So, we need to have SSH up and running on our machine and configured it to allow SSH public key authentication.

Hadoop uses SSH (to access its nodes) which would normally require the user to enter a password. However, this requirement can be eliminated by creating and setting up SSH certificates using the following commands. If asked for a filename just leave it blank and press the enter key to continue.

abc@laptop:~$ su hduser

Password:

hduser@laptop:~$ ssh-keygen -t rsa -P ""

hduser@laptop: /home/abc$ cat $HOME/ .ssh/id_rsa.pub >> $HOME/ .ssh/authorized_keys

The above command adds the newly created key to the list of authorized keys so that Hadoop can use ssh without prompting for a password.

We can check if ssh works:

hduser@laptop :/home/k$ ssh localhost

Step 5: Installing HADOOP

hduser@laptop :~$ wget http://mirrors.sonic.net/apache/hadoop/common/hadoop-2.7.4 /hadoop-2.7.4.tar.gz

hduser@laptop :~$ tar xvzf hadoop-2.7.4.tar.gz

We want to move the Hadoop installation to the **/usr/local/hadoop** directory using the following command:

hduser@laptop:~/hadoop-2.7.4$ sudo mv * /usr/local/Hadoop
[sudo] password for hduser:
hduser is not in the sudoers file. This incident will be reported.

This error can be resolved by logging in as a root user, and then add **hduser** to **sudo**:

hduser@laptop:~/hadoop-2.7.4$ su abc

Password:

Now, the **hduser** has root priviledge, we can move the Hadoop installation to the **/usr/local/hadoop** directory without any problem:

abc@laptop:/home/hduser$ sudo su hduser

hduser@laptop:~/hadoop-2.7.4$ sudo mv * /usr/local/hadoop

hduser@laptop:~/hadoop-2.7.4$ sudo chown –R hduser: hadoop /usr/local/Hadoop

Step 6:
Setup Configuration Files

The following files will have to be modified to complete the Hadoop setup:

1. ~/.bashrc
2. /usr/local/hadoop/etc/hadoop/hadoop-env.sh
3. /usr/local/hadoop/etc/hadoop/core-site.xml
4. /usr/local/hadoop/etc/hadoop/mapred-site.xml.template
5. /usr/local/hadoop/etc/hadoop/hdfs-site.xml

6.1. ~/.bashrc:

Before editing the **.bashrc** file in our home directory, we need to find the path where Java has been installed to set the **JAVA_HOME** environment variable using the following command:

hduser@laptop:~/sudo nano ~/ .bashrc

Copy the following:

```
#HADOOP VARIABLES START
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
#HADOOP VARIABLES END
```

6.2: **/usr/local/hadoop/etc/hadoop/hadoop-env.sh**

We need to set **JAVA_HOME** by modifying **hadoop-env.sh** file.

```
hduser@laptop:~$ vi /usr/local/hadoop/etc/hadoop/hadoop-env.sh
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-i386
```

Adding the above statement in the **hadoop-env.sh** file ensures that the value of JAVA_HOME variable will be available to Hadoop whenever it is started up.

6.3. /usr/local/hadoop/etc/hadoop/core-site.xml:

The **/usr/local/hadoop/etc/hadoop/core-site.xml** file contains configuration properties that Hadoop uses when starting up.
This file can be used to override the default settings that Hadoop starts with.

```
hduser@laptop:~$ sudo mkdir –p /app/hadoop/tmp
```

```
hduser@laptop:~$ sudo chown hduser:hadoop /app/hadoop/tmp
```

Open the file and enter the following in between the <configuration></configuration> tag:

```
hduser@laptop:~$ vi /usr/local/hadoop/etc/hadoop/core-site.xml

<configuration>
 <property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
  <description>A base for other temporary directories.</description>
 </property>

 <property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:54310</value>
```

```
 <description>The name of the default file system.  A URI whose
 scheme and authority determine the FileSystem implementation.  The
 uri's scheme determines the config property (fs.SCHEME.impl) naming
 the FileSystem implementation class.  The uri's authority is used to
 determine the host, port, etc. for a filesystem.</description>
</property>
</configuration>
```

6.4. /usr/local/hadoop/etc/hadoop/mapred-site.xml

By default, the **/usr/local/hadoop/etc/hadoop/** folder contains **/usr/local/hadoop/etc/hadoop/mapred-site.xml.template** file which has to be renamed/copied with the name **mapred-site.xml**:

```
hduser@laptop:~$ cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template /usr/local/hadoop/etc/hadoop/mapred-site.xml
```

The **mapred-site.xml** file is used to specify which framework is being used for MapReduce.
We need to enter the following content in between the <configuration></configuration> tag:

```
<configuration>
 <property>
  <name>mapred.job.tracker</name>
  <value>localhost:54311</value>
```

```
 <description>The host and port that the MapReduce job tracker runs
 at.  If "local", then jobs are run in-process as a single map
 and reduce task.
 </description>
</property>
</configuration>
```

5. /usr/local/hadoop/etc/hadoop/hdfs-site.xml

The **/usr/local/hadoop/etc/hadoop/hdfs-site.xml** file needs to be configured for each host in the cluster that is being used.
It is used to specify the directories which will be used as the **namenode** and the **datanode** on that host.

Before editing this file, we need to create two directories which will contain the namenode and the datanode for this Hadoop installation.
This can be done using the following commands:

```
hduser@laptop:~$ sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode
hduser@laptop:~$ sudo mkdir -p /usr/local/hadoop_store/hdfs/datanode
hduser@laptop:~$ sudo chown -R hduser:hadoop /usr/local/hadoop_store
```

Open the file and enter the following content in between the <configuration></configuration> tag:

```
hduser@laptop:~$ sudo nano /usr/local/hadoop/etc/hadoop/hdfs-site.xml


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

```
  The actual number of replications can be specified when the file is created.
  The default is used if replication is not specified in create time.
 </description>
</property>
<property>
  <name>dfs.namenode.name.dir</name>
  <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
</property>
</configuration>
```

Step 7: Format the New Hadoop Filesystem

Now, the Hadoop file system needs to be formatted so that we can start to use it. The format command should be issued with write permission since it
creates **current** directory under **/usr/local/hadoop_store/hdfs/namenode** folder:

```
hduser@laptop:~$ hadoop namenode -format
```

Step 8: Starting Hadoop

Now it's time to start the newly installed single node cluster.
We can use **start-all.sh** or (**start-dfs.sh** and **start-yarn.sh**)

```
abc@laptop:~$ cd /usr/local/hadoop/sbin
```

```
abc@laptop:/usr/local/hadoop/sbin$ ls
abc@laptop:/usr/local/hadoop/sbin$ sudo su hduser
hduser@laptop:/usr/local/hadoop/sbin$ start-all.sh
hduser@laptop:~$ start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
15/04/18 16:43:13 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your plat
form... using builtin-java classes where applicable
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-namenode-laptop.out
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-laptop.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hduser-secondaryna
menode-laptop.out
15/04/18 16:43:58 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your plat
form... using builtin-java classes where applicable
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-hduser-resourcemanager-laptop.out
localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-nodemanager-laptop.
out
```

Step 9: We can check if it's really up and running:

```
hduser@laptop:/usr/local/hadoop/sbin$ jps
9026 NodeManager
7348 NameNode
9766 Jps
8887 ResourceManager
7507 DataNode
```

```
$ pwd
/usr/local/hadoop/sbin

$ ls
distribute-exclude.sh  httpfs.sh              start-all.sh        start-yarn.cmd    stop-dfs.cmd        yarn-dae
mon.sh
hadoop-daemon.sh       mr-jobhistory-daemon.sh  start-balancer.sh   start-yarn.sh     stop-dfs.sh
yarn-daemons.sh
hadoop-daemons.sh      refresh-namenodes.sh     start-dfs.cmd       stop-all.cmd      stop-secure-dns.sh
hdfs-config.cmd        slaves.sh                start-dfs.sh        stop-all.sh       stop-yarn.cmd
hdfs-config.sh         start-all.cmd            start-secure-dns.sh  stop-balancer.sh  stop-yarn.sh
```

Step 10: Stopping Hadoop

We run **stop-all.sh** or (**stop-dfs.sh** and **stop-yarn.sh**) to stop all the daemons running on our machine:

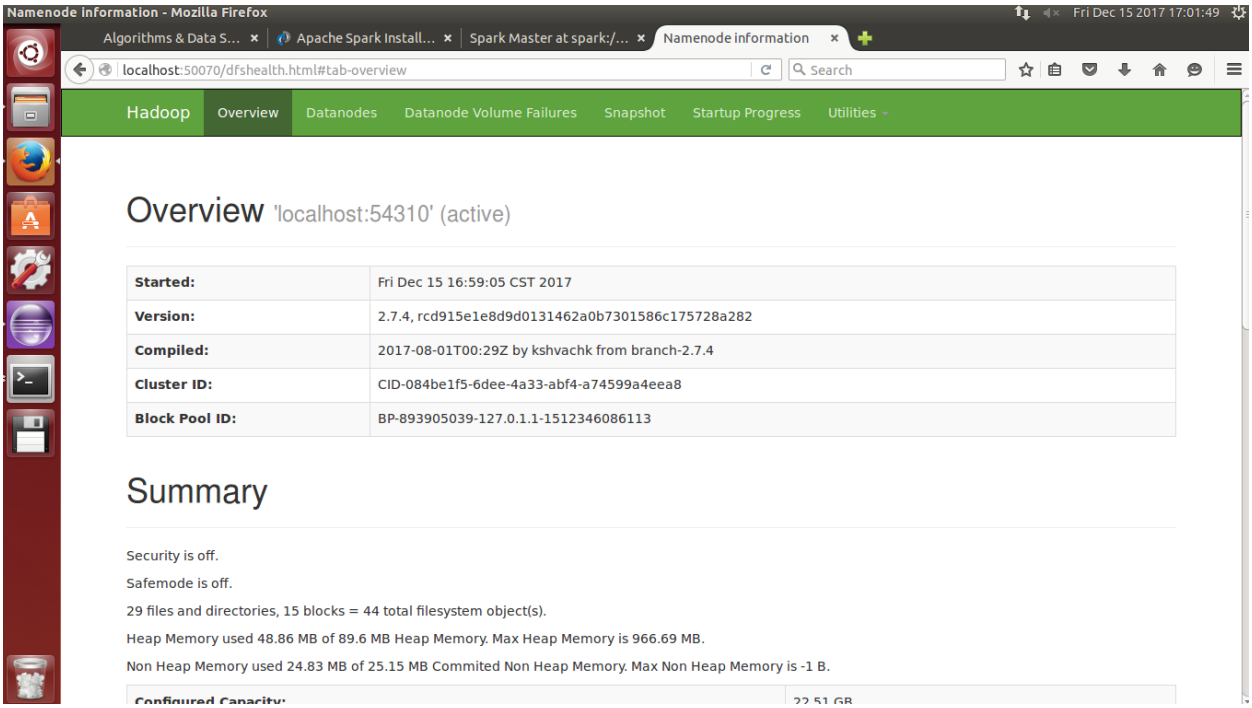
```
hduser@laptop:/usr/local/hadoop/sbin$ pwd
/usr/local/hadoop/sbin
hduser@laptop:/usr/local/hadoop/sbin$
```

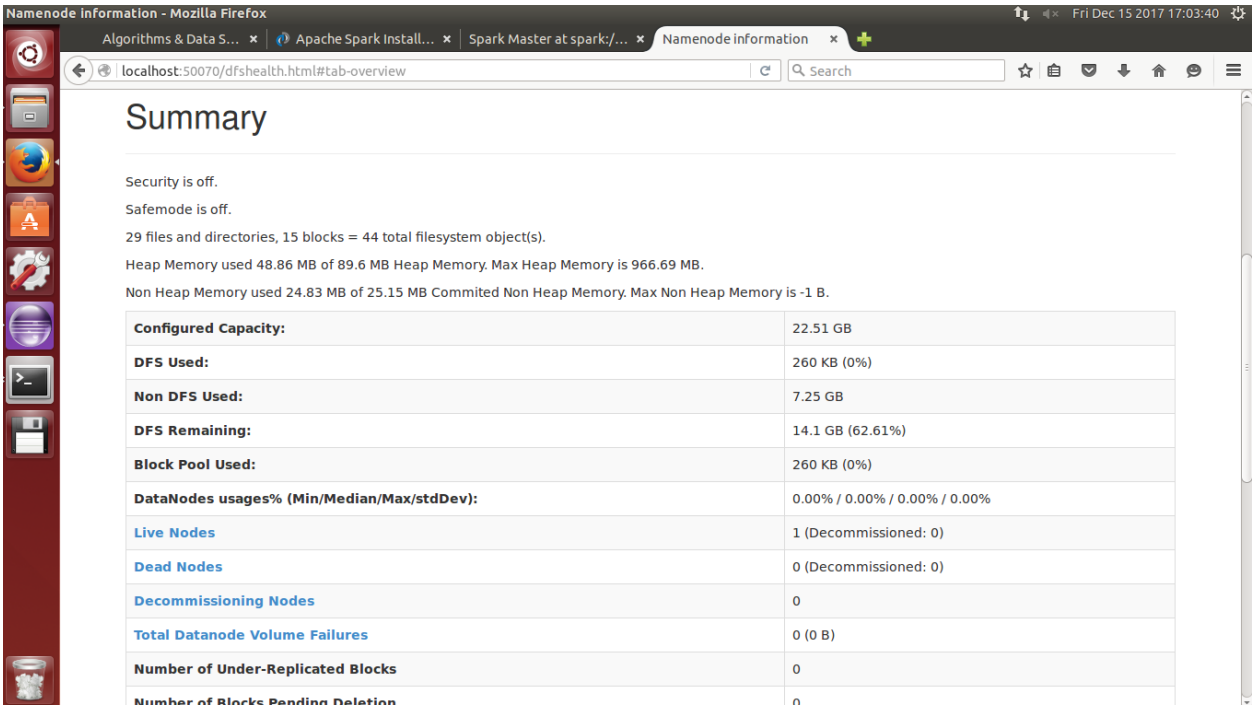
Step 11: Hadoop Web Interfaces:

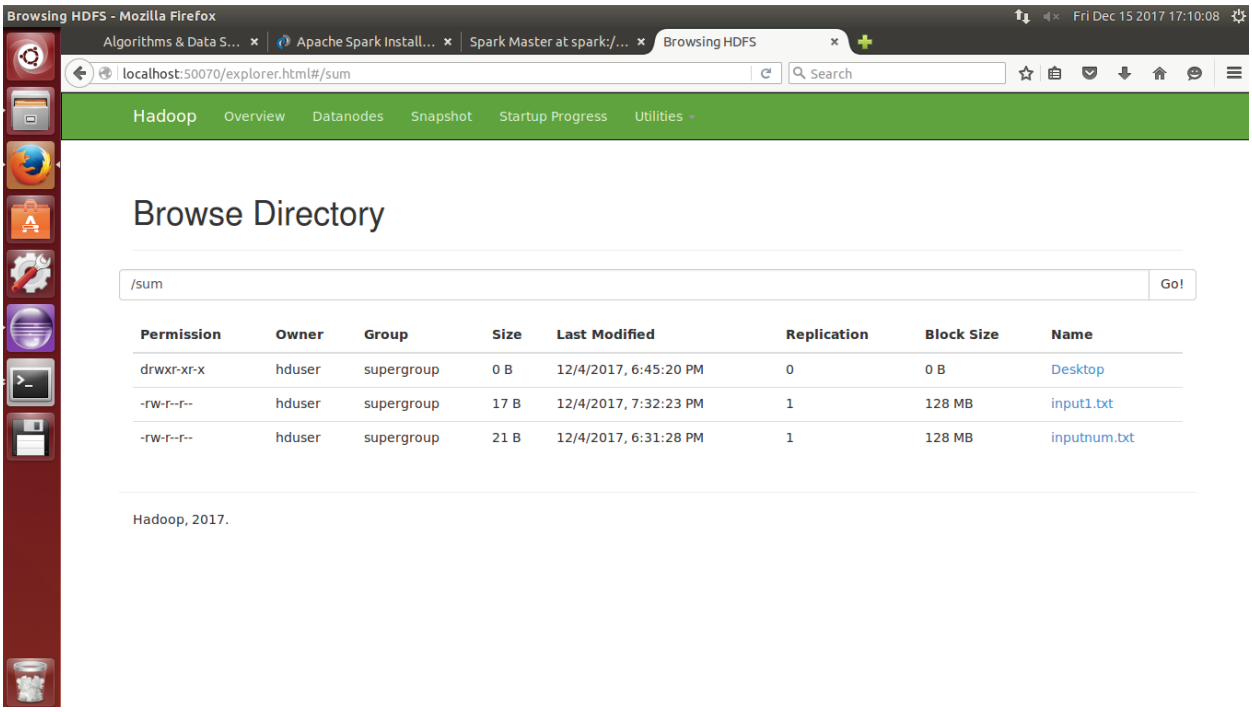
Let's start the Hadoop again and see its Web UI:

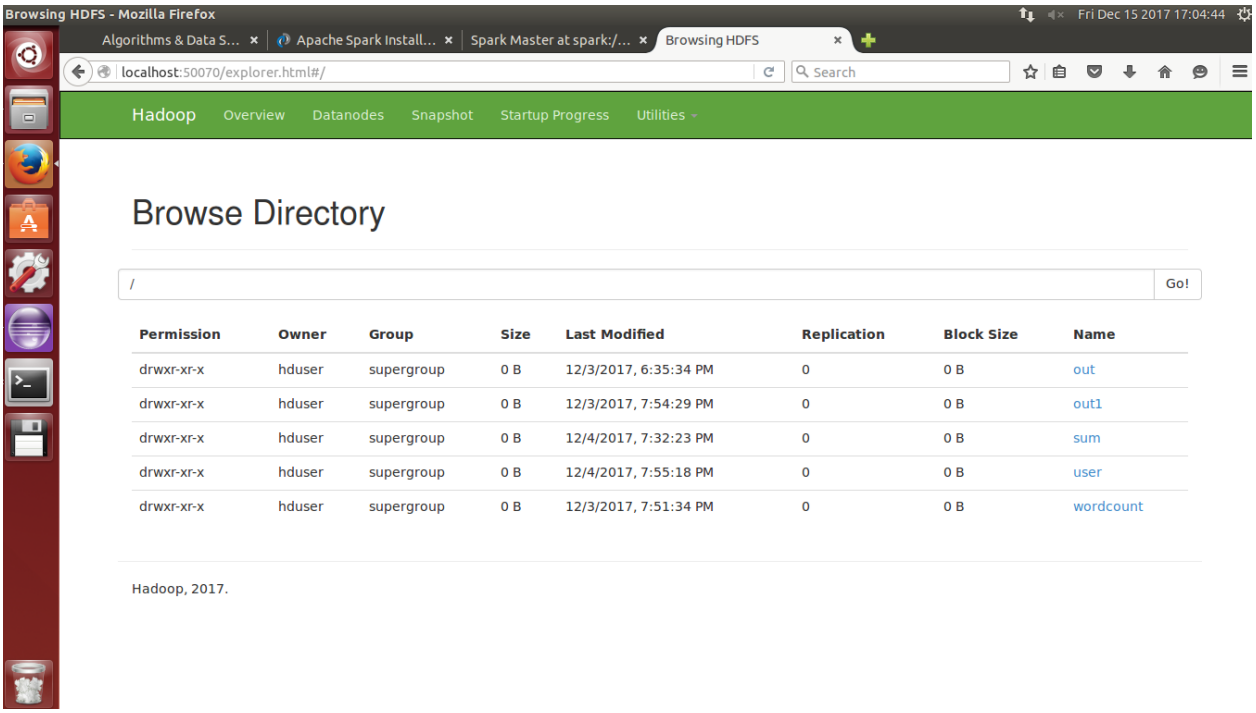
```
hduser@laptop:/usr/local/hadoop/sbin$ start-all.sh
```

http://localhost:50070/ - web UI of the NameNode daemon


Namenode information - Mozilla Firefox
Fri Dec 15 2017 17:01:49
Algorithms & Data S...
Apache Spark Install...
Spark Master at spark:/...
Namenode information
localhost:50070/dfshealth.html#tab-overview
Search
Hadoop
Overview
Datanodes
Datanode Volume Failures
Snapshot
Startup Progress
Utilities
Overview 'localhost:54310' (active)
Started: Fri Dec 15 16:59:05 CST 2017
Version: 2.7.4, rcd915e1e8d9d0131462a0b7301586c175728a282
Compiled: 2017-08-01T00:29Z by kshvachk from branch-2.7.4
Cluster ID: CID-084be1f5-6dee-4a33-abf4-a74599a4eea8
Block Pool ID: BP-893905039-127.0.1.1-1512346086113
Summary
Security is off.
Safemode is off.
29 files and directories, 15 blocks = 44 total filesystem object(s).
Heap Memory used 48.86 MB of 89.6 MB Heap Memory. Max Heap Memory is 966.69 MB.
Non Heap Memory used 24.83 MB of 25.15 MB Commited Non Heap Memory. Max Non Heap Memory is -1 B.
Configured Capacity: 22.51 GB


Namenode information - Mozilla Firefox
Fri Dec 15 2017 17:03:40
Algorithms & Data S...
Apache Spark Install...
Spark Master at spark:/...
Namenode information
localhost:50070/dfshealth.html#tab-overview
Search
Summary
Security is off.
Safemode is off.
29 files and directories, 15 blocks = 44 total filesystem object(s).
Heap Memory used 48.86 MB of 89.6 MB Heap Memory. Max Heap Memory is 966.69 MB.
Non Heap Memory used 24.83 MB of 25.15 MB Commited Non Heap Memory. Max Non Heap Memory is -1 B.
Configured Capacity: 22.51 GB
DFS Used: 260 KB (0%)
Non DFS Used: 7.25 GB
DFS Remaining: 14.1 GB (62.61%)
Block Pool Used: 260 KB (0%)
DataNodes usages% (Min/Median/Max/stdDev): 0.00% / 0.00% / 0.00% / 0.00%
Live Nodes 1 (Decommissioned: 0)
Dead Nodes 0 (Decommissioned: 0)
Decommissioning Nodes 0
Total Datanode Volume Failures 0 (0 B)
Number of Under-Replicated Blocks 0
Number of Blocks Pending Deletion 0





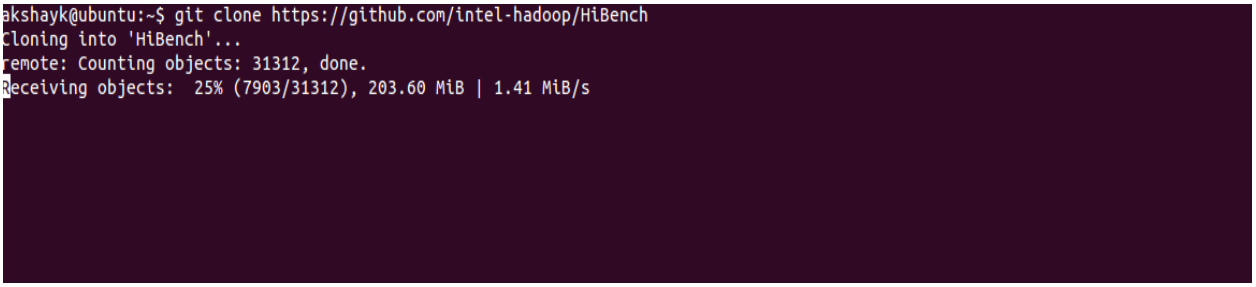
Step 12: To configure YARN the relevant file is /etc/hadoop/yarn-site.xml. For a single-node installation of YARN you'll want to add the following to that file:

```
<configuration>
  <property>
    <name>yarn.scheduler .minimum-allocation-mb</name>
    <value>128</value>
    <description>Minimum limit of memory to allocate to each container request at the Resource Manager. </description>
  </property>
  <property>
    <name> yarn.scheduler .maximum-allocation-mb</name>
    <value>2048</value>
    <description>Maximum limit of memory to allocate to each container request at the Resource Manager. </description>
  </property>
  <property>
    <name>yarn.scheduler .minimum-allocation-vcores</name>
    <value>1</value>
    <description>The minimum allocation for every container request at the RM, in terms of virtual CPU cores. Requests lower than this won't take effect, and the specified value will get allocated the minimum. </description>
  </property>
  <property>
    <name> yarn. scheduler .maximum-allocation-vcores</name>
    <value>2</value>
    <description>The maximum allocation for every container request at the RM, in terms of virtual CPU cores. Requests higher than this won't take effect, and will get capped to this value </description>
  </property>
  <property>
    <name> yarn.nodemanager .resource.memory-mb</name>
    <value>4096</value>
    <description>Physical memory, in MB, to be made available to running containers</description>
  </property>
  <property>
    <name>yarn.nodemanager.resource.cpu-vcores</name>
    <value>4</value>
    <description>Number of CPU cores that can be allocated for containers .</description>
  </property>
```

```
</configuration>
```

INSTALLING AND CONFIGURING HIBENCH ON HADOOP AND YARN BASE

1. **Clone the repository from the Github Source:**



2. **Install Maven on the local system:**

   cd /opt/

   wget http://www-eu.apache.org/dist/maven/maven-3/3.0.4/binaries/apache-maven-3.0.4-bin.tar.gz

   sudo tar -xvzf apache-maven-3.0.4-bin.tar.gz

   sudo mv apache-maven-3.3.9 maven

   Next, you will need to setup the environment variables such as M2_HOME, M2, MAVEN_OPTS, and PATH. You can do this by creating a mavenenv.sh file inside of the /etc/profile.d/

   sudo nano /etc/profile.d/mavenenv.sh

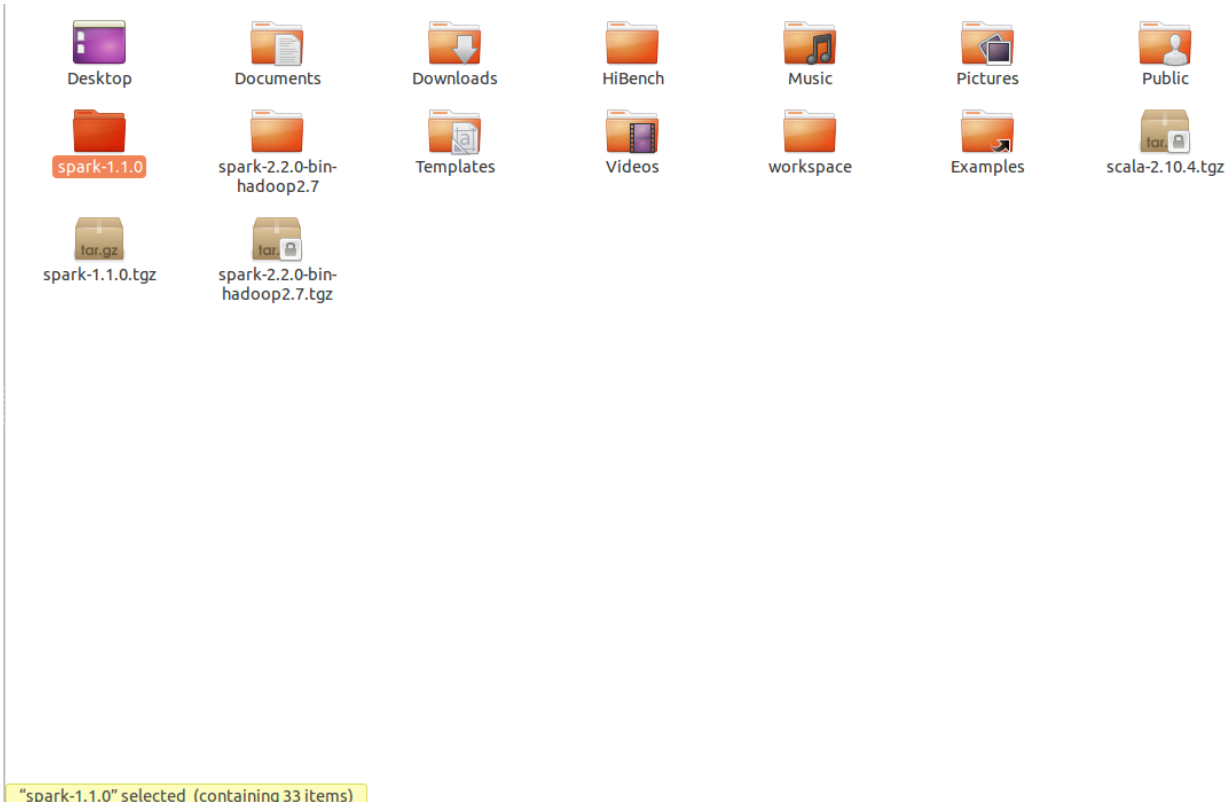
   export M2_HOME=/opt/maven
   export PATH=${M2_HOME}/bin:${PATH}

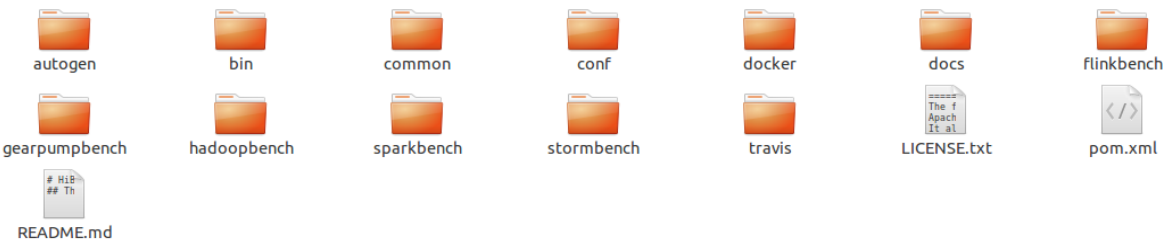
Save and close the file, update its permissions, then load the environment variables with the following command:

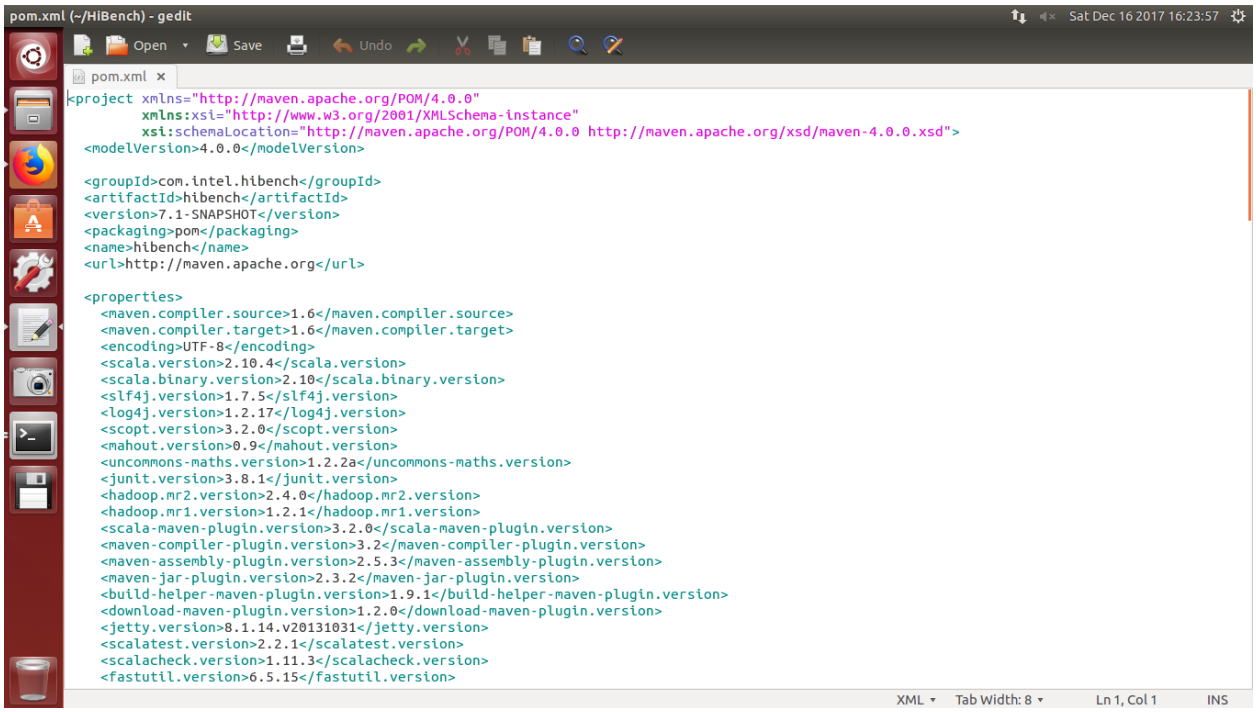
sudo chmod +x /etc/profile.d/mavenenv.sh
sudo source /etc/profile.d/mavenenv.sh

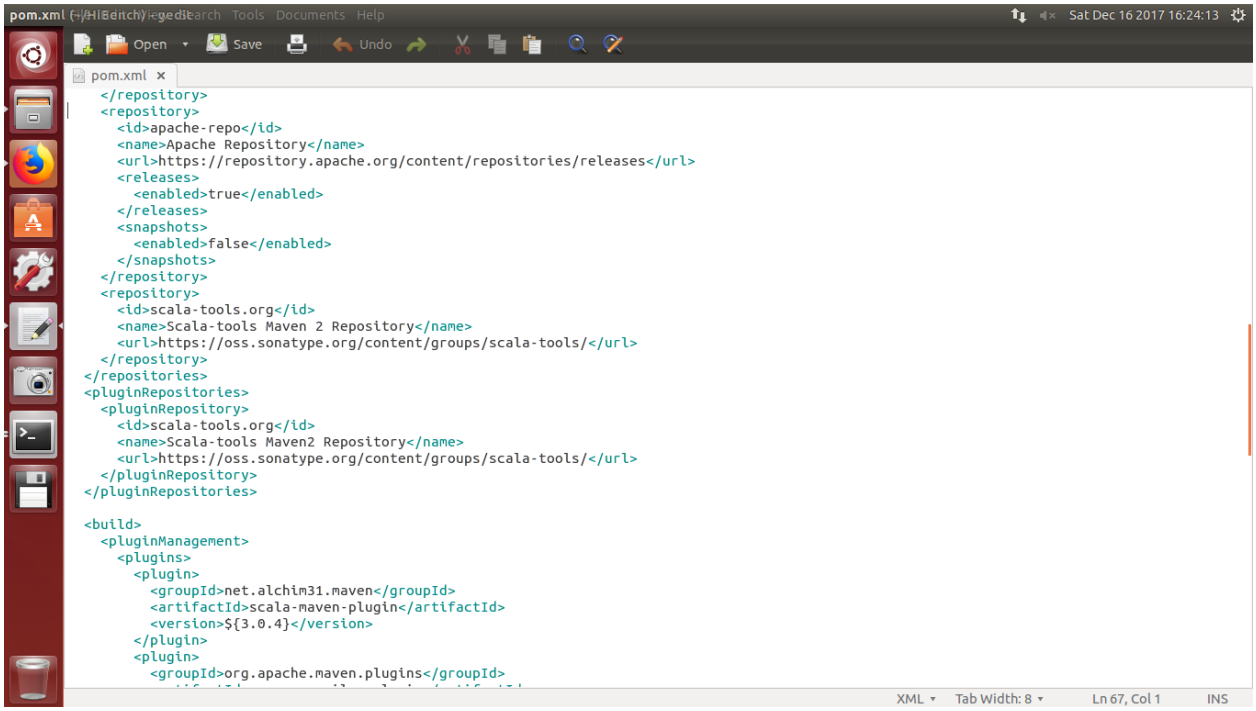
Once everything has been successfully configured, check the version of the Apache Maven
mvn --version

3. Make the additions in the pom.xml file of Hibench:



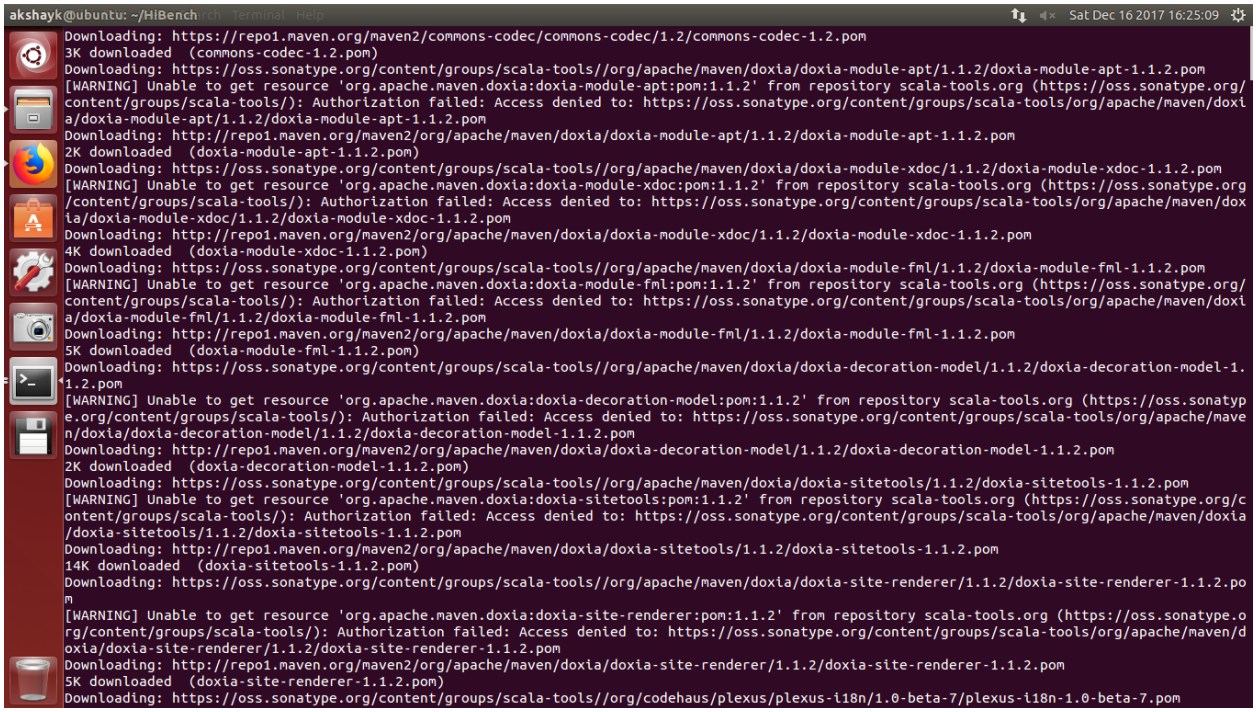

autogen
bin
common
conf
docker
docs
flinkbench
gearpumpbench
hadoopbench
sparkbench
stormbench
travis
LICENSE.txt
pom.xml
README.md

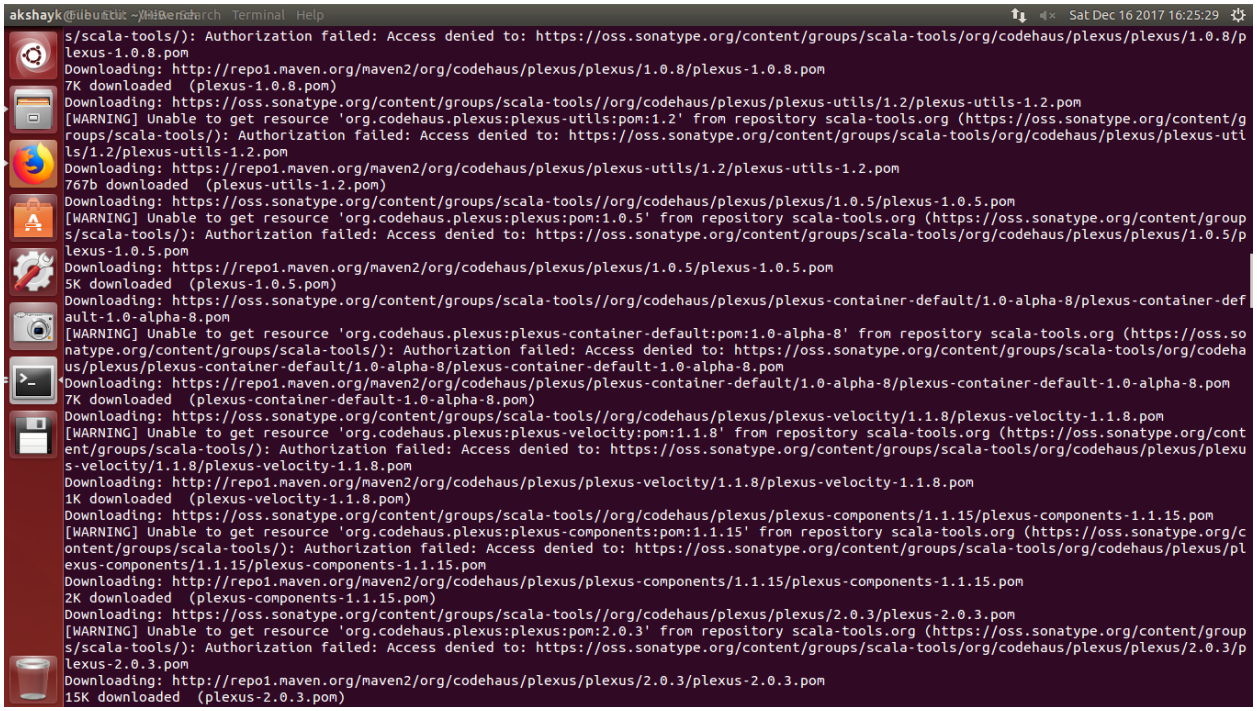

pom.xml (~/HiBench) - gedit
Sat Dec 16 2017 16:23:57
Open
Save
Undo
pom.xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.intel.hibench</groupId>
<artifactId>hibench</artifactId>
<version>7.1-SNAPSHOT</version>
<packaging>pom</packaging>
<name>hibench</name>
<url>http://maven.apache.org</url>
<properties>
<maven.compiler.source>1.6</maven.compiler.source>
<maven.compiler.target>1.6</maven.compiler.target>
<encoding>UTF-8</encoding>
<scala.version>2.10.4</scala.version>
<scala.binary.version>2.10</scala.binary.version>
<slf4j.version>1.7.5</slf4j.version>
<log4j.version>1.2.17</log4j.version>
<scopt.version>3.2.0</scopt.version>
<mahout.version>0.9</mahout.version>
<uncommons-maths.version>1.2.2a</uncommons-maths.version>
<junit.version>3.8.1</junit.version>
<hadoop.mr2.version>2.4.0</hadoop.mr2.version>
<hadoop.mr1.version>1.2.1</hadoop.mr1.version>
<scala-maven-plugin.version>3.2.0</scala-maven-plugin.version>
<maven-compiler-plugin.version>3.2</maven-compiler-plugin.version>
<maven-assembly-plugin.version>2.5.3</maven-assembly-plugin.version>
<maven-jar-plugin.version>2.3.2</maven-jar-plugin.version>
<build-helper-maven-plugin.version>1.9.1</build-helper-maven-plugin.version>
<download-maven-plugin.version>1.2.0</download-maven-plugin.version>
<jetty.version>8.1.14.v20131031</jetty.version>
<scalatest.version>2.2.1</scalatest.version>
<scalacheck.version>1.11.3</scalacheck.version>
<fastutil.version>6.5.15</fastutil.version>
XML
Tab Width: 8
Ln 1, Col 1
INS

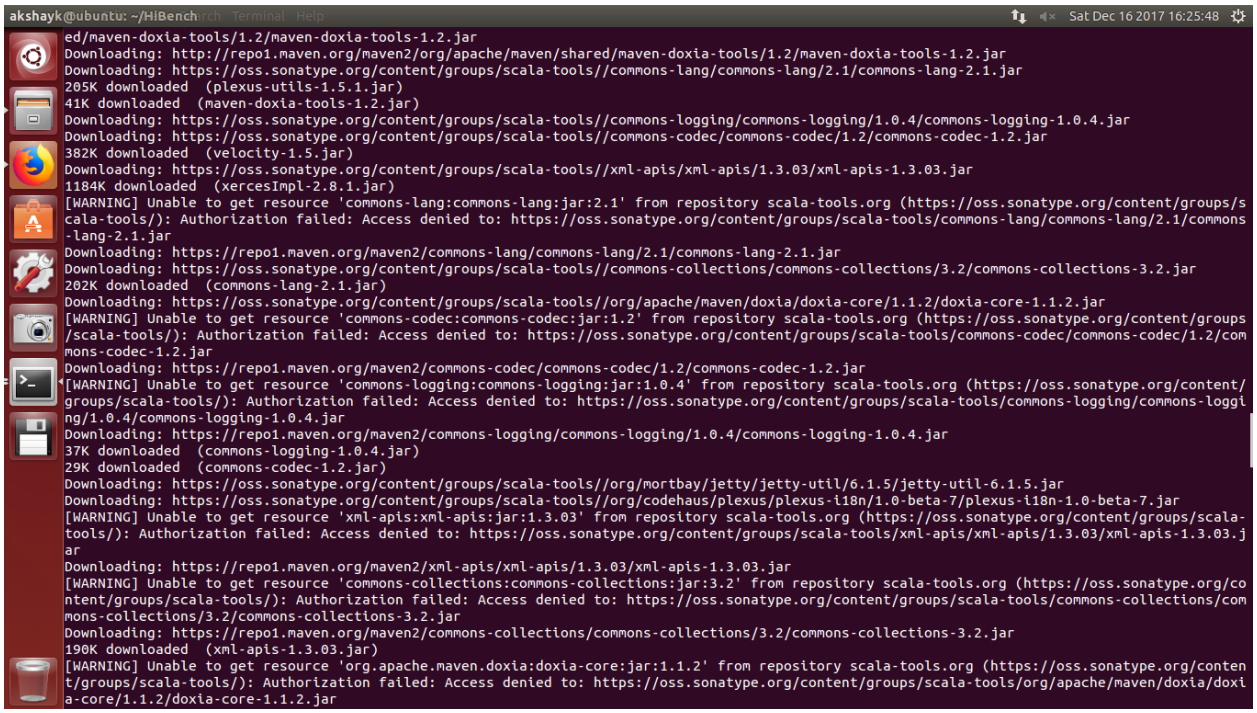


4. DEPLOY AND BUILD HI BENCH=

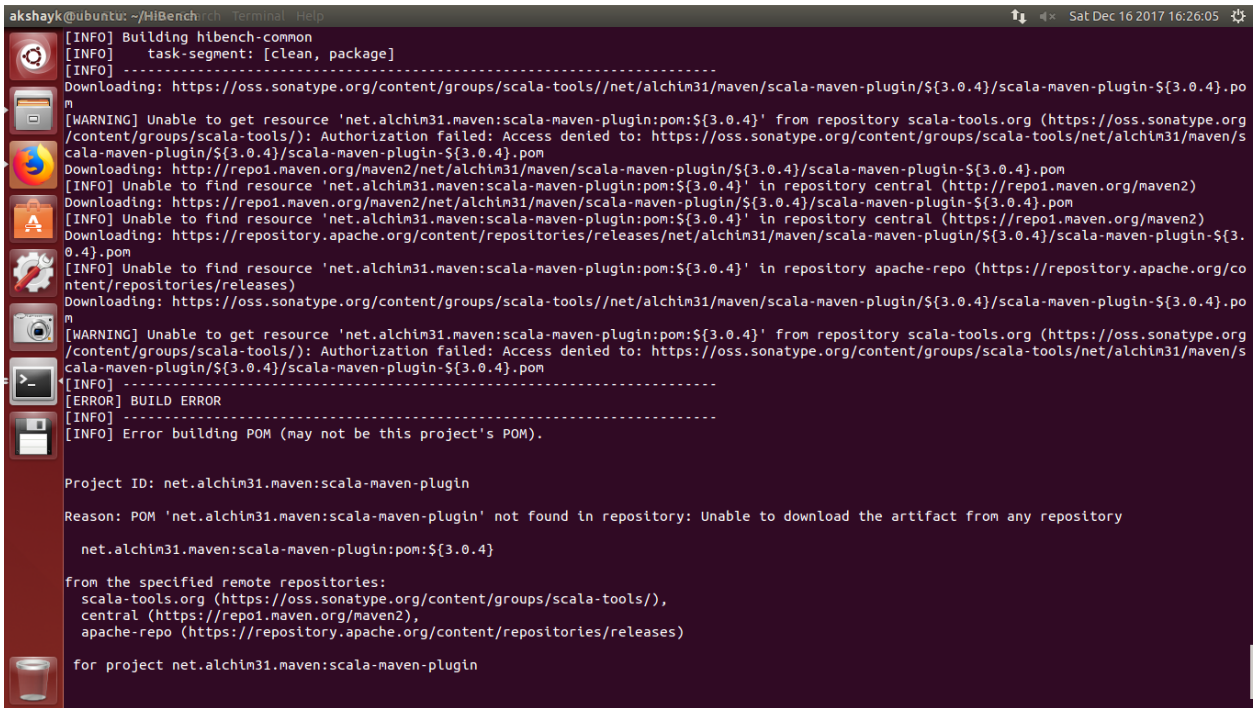
mvn -Phadoopbench -Dspark=2.2 -Dscala=2.10 clean package







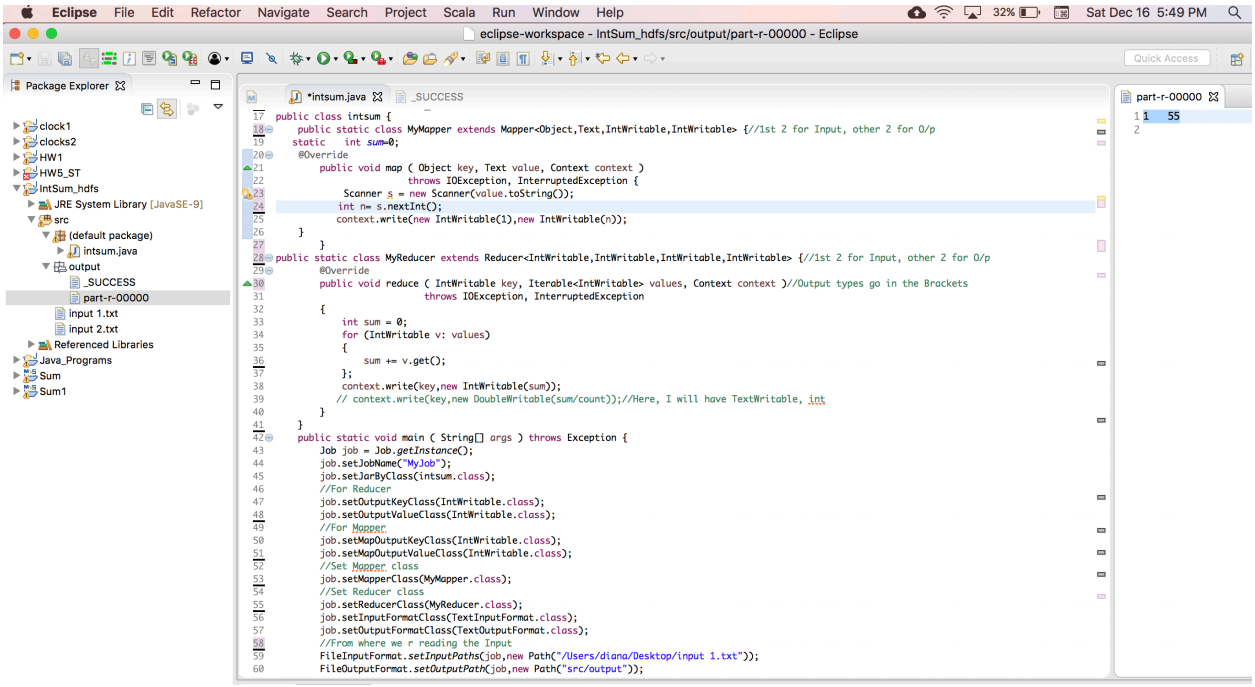
However, an error is repetitively cropping up.



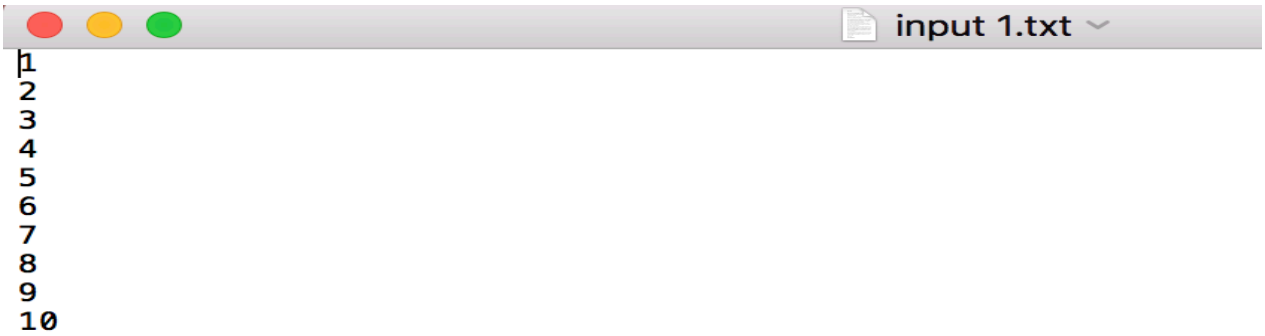
Despite reconfiguring the /conf folder an error is being thrown up as it is not able to find the mirror repositories.
So in our project we have installed Hi-Bench but are not able to deploy it to the DFS framework.

Assignment-2 (20pts): Implement a simple MapReduce job – *IntSum* and execute it in the YARN environment. The job adds all integer numbers of an input file and outputs the sum of these numbers. Learn from the examples included in the YARN distribution, such as wordcount or grep, on how to write a MapReduce job. The source code of the examples can be found at hadoop-2.7.2/share/hadoop/mapreduce/sources/hadoop-mapreduce-examples-2.7.2-sources.jar

Solution 2: (Program + Output)

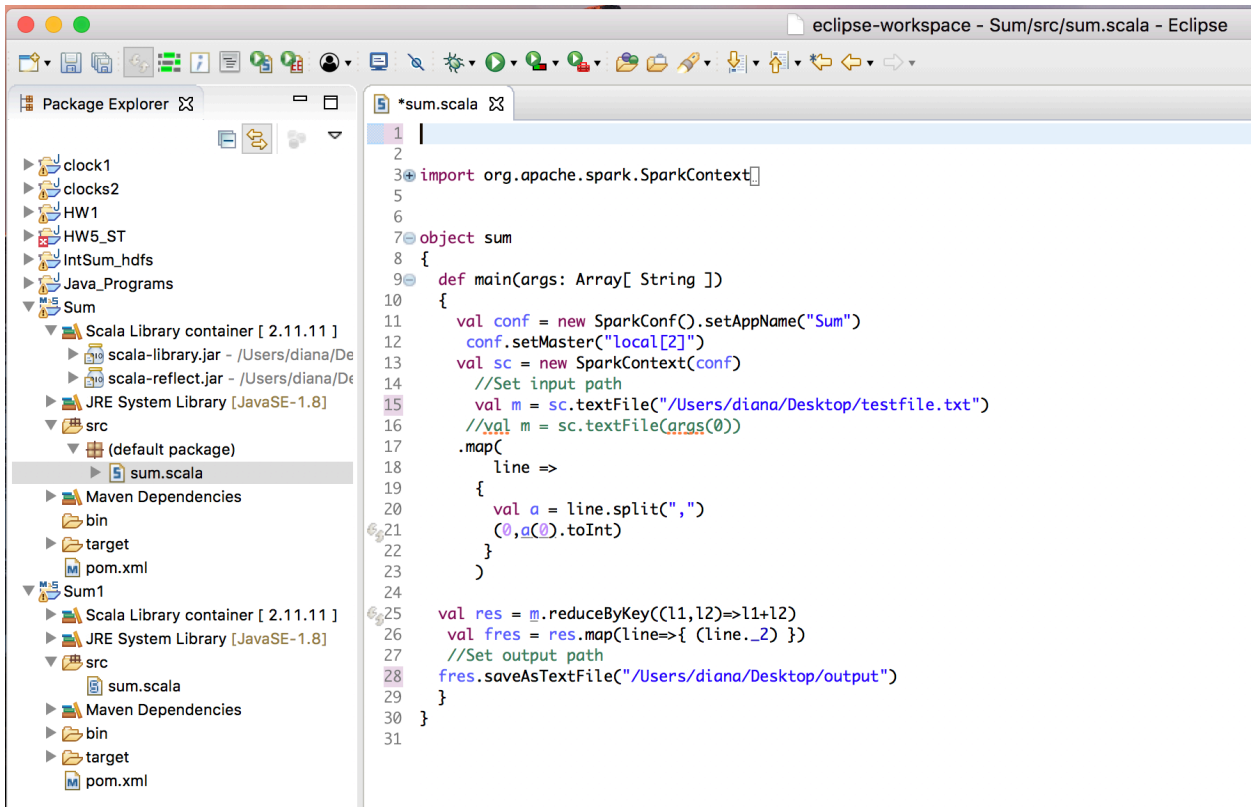


INPUT FILE:



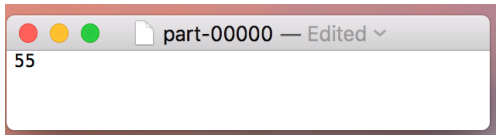
Assignment-3 (20pts): Re-implement the *IntSum* job in Spark and execute it in the YARN environment. Quantitatively compare the performance of the MapReduce and Spark implementations using the same input file.

Solution 3:



INPUT FILE:

OUTPUT FILE:



Assignment-4 (20pts): Read the papers listed below and other online articles if needed, and answer the following questions based on your own understandings.

Solution:

1.What are the key differences between Hadoop and Spark, and their respective advantages?

	Spark	**Hadoop**
Introduction	It is an open source big data framework. It provides faster and more general-purpose data processing engine. Spark is basically designed for fast computation. It also covers wide range of workloads f or example batch, interactive, iterative and streaming.	It is also an open source framework for writing applications. It also processes structured and unstructured data that are stored in **HDFS**. Hadoop MapReduce is designed in a way to process a large volume of data on a cluster of commodity hardware. MapReduce can process data in batch mode.
Speed	Spark is lightning fast cluster computing tool. Apache Spark runs applications up to 100x faster in memory and 10x faster on disk than Hadoop. Because of reducing the number of read/write cycle to disk and storing intermediate data in-memory Spark makes it possible.	MapReduce reads and writes from disk, as a result, it slows down the processing speed.
Difficulty	Spark is easy to program as it has tons of high-level operators with	In MapReduce, developers need to hand code each and every operation which makes it very difficult to work.

Easy to Manage	Spark is capable of performing batch, interactive and Machine Learning and Streaming all in the same cluster. As a result, makes it a complete data analytics engine. Thus, no need to manage different component for each need. Installing Spark on a cluster will be enough to handle all the requirements.	As MapReduce only provides the batch engine. Hence, we are dependent on different engines. For example- Storm, Giraph, Impala, etc. for other requirements. So, it is very difficult to manage many components.
Real-time analysis	It can process real time data i.e. data coming from the real-time event streams at the rate of millions of events per second, e.g. Twitter data for instance or Facebook sharing/posting. Spark's strength is the ability to process live streams efficiently.	MapReduce fails when it comes to real-time data processing as it was designed to perform batch processing on voluminous amounts of data.
Latency	Spark provides low-latency computing.	MapReduce is a high latency computing framework.
Interactive mode	Spark can process data interactively	MapReduce doesn't have an interactive mode.

Streaming	Spark can process real time data through Spark Streaming	With MapReduce, you can only process data in batch mode.
Ease of use	Spark is easier to use. Since, its abstraction (RDD) enables a user to process data using high-level operators. It also provides rich APIs in Java, Scala**,** Python, and R.	MapReduce is complex. As a result, we need to handle low-level APIs to process the data, which requires lots of hand coding.
Recovery	RDDs allows recovery of partitions on failed nodes by re-computation of the DAG while also supporting a more similar recovery style to Hadoop by way of checkpointing, to reduce the dependencies of an RDDs.	MapReduce is naturally resilient to system faults or failures. So, it is a highly fault-tolerant system.
Scheduler	Due to in-memory computation spark acts its own flow scheduler.	MapReduce needs an external job scheduler for example, Oozie to schedule complex flows.
Fault tolerance	Spark is fault-tolerant. As a result, there is no need to restart the application from scratch in case of any failure.	Like Apache Spark, MapReduce is also fault-tolerant, so there is no need to restart the application from scratch in case of any failure.
Security	Spark is little less secure in comparison to MapReduce	Apache Hadoop MapReduce is more secure because of

	because it supports the only authentication through shared secret password authentication.	Kerberos and it also supports Access Control Lists (ACLs) which are a traditional file permission model.
Cost	As spark requires a lot of RAM to run in-memory. Thus, increases the cluster, and also its cost.	MapReduce is a cheaper option available while comparing it in terms of cost.
Language Developed	Spark is developed in Scala.	Hadoop MapReduce is developed in Java.
Category	It is data analytics engine. Hence, it is a choice for Data Scientist.	It is basic data processing engine.
OS support	Spark supports cross-platform.	Hadoop MapReduce also supports cross-platform.
SQL support	It enables the user to run SQL queries using Spark SQL.	It enables users to run SQL queries using Apache Hive.
Scalability	Spark is highly scalable. Thus, we can add n number of nodes in the cluster. Also, a largest known Spark Cluster is of 8000 nodes.	MapReduce is also highly scalable we can keep adding n number of nodes in the cluster. Also, a largest known Hadoop cluster is of 14000 nodes.
The line of code	Apache Spark is developed in merely 20000 lines of codes.	Hadoop 2.0 has 1,20,000 lines of codes
Machine Learning	Spark has its own set of machine learning ie MLlib.	Hadoop requires machine learning tool for example Apache Mahout**.**
Caching	Spark can cache data in memory for further iterations. As a result, it enhances the system performance.	MapReduce cannot cache the data in memory for future requirements. So, the processing speed is not that high as that of Spark.
Hardware Requirements	Spark needs mid to high-level hardware.	MapReduce runs very well on commodity hardware.
Community	Spark is one of the most active project at Apache. Since, it has a very strong community.	MapReduce community has been shifted to Spark.

Q 2: Discuss how to recover a failed task in Spark and Hadoop, respectively.
Solution:
Solution: There are two properties below which can determine how many failures or attempts of a task could be acceptable. mapred.map.max.attempts for Map tasks and a property mapred.reduce.max.attempts for reduce tasks. By default, if any task fails four times (or whatever you configure in those properties), the whole job would be considered as failed.

Bibliography:

1. https://data-flair.training/blogs/apache-spark-vs-hadoop-mapreduce/