Frog-B-Data

Developer's Manual

Version 4.0

2 May, 2016

Revision Signatures

By signing the following document, the team member is acknowledging that he has read the entire document thoroughly and has verified that the information within this document is, to the best of his knowledge, is accurate, relevant and free of typographical errors.

Name	**Signature**	**Date**
Sushant Ahuja		
Cassio Lisandro Caposso Cristovao		
Sameep Mohta		

Revision History

The following table shows the revisions made to this document.

Version	**Changes**	**Date**
1.0	Initial Draft	20 April, 2016
2.0	Apache Spark	25 April, 2016
3.0	Apache Hadoop	29 April, 2016
4.0	Formatting and screenshots	2 May, 2016

Table of Contents

1 Introduction

1.1 Purpose

The purpose of this document is to provide the developers with all the necessary tools and installation guides to setup and continue the development of Frog-B-Data, senior capstone project. This document contains a detailed breakdown of creating a cluster of nodes on Linux machines, Hadoop and Spark installation, and running sample codes on standalone as well as a cluster of nodes. Additionally, we provide detailed description on building the recommender on two separate frameworks.

1.2 Overview

This document includes the following four sections.

Section 2 - Pre-Requisites: Gives a detailed description of the required operating system and pre-installed software on machines before beginning of the project.

Section 3 - Hadoop: This section gives a very detailed description on the introduction to building Maven projects on Eclipse Mars, creating a cluster of network working on multiple nodes, installation of Hadoop on standalone and cluster of machines, deploying apps on single node and cluster, recommender and K-Means clustering using Mahout.

Section 4 - Spark: This section gives a very detailed description on the introduction to building Maven projects on Eclipse Mars, creating a cluster of network working on multiple nodes, installation of Spark on standalone and cluster of machines, deploying apps on single node and cluster, Python development in Spark, recommender and K-Means clustering using MLlib.

Section 5 - Glossary of Terms: Lists all the technical terms that are mentioned in this document with their definitions.

2 Pre-Requisites

2.1 Ubuntu 15.04

All machines involved in this project must be installed with Linux operating systems, preferably Ubuntu 15.04 which can be downloaded from the following link:

http://releases.ubuntu.com/15.04/ubuntu-15.04-desktop-amd64.iso

2.2 Java 8 download

Java is the integral part of Frog-B-data project and all machines must have the latest versions of java installed using the following command line instructions:

```
sudo apt-get update
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

After entering these commands, verify that java is installed on your machine using the following command:

```
java –version
```

2.3 Eclipse Mars

This project requires an IDE to develop the applications on Hadoop and Spark. Download the Eclipse Mars from the following link:

https://www.eclipse.org/downloads/download.php?file=/oomph/epp/mars/R2/eclipse-inst-linux64.tar.gz&mirror_id=1135

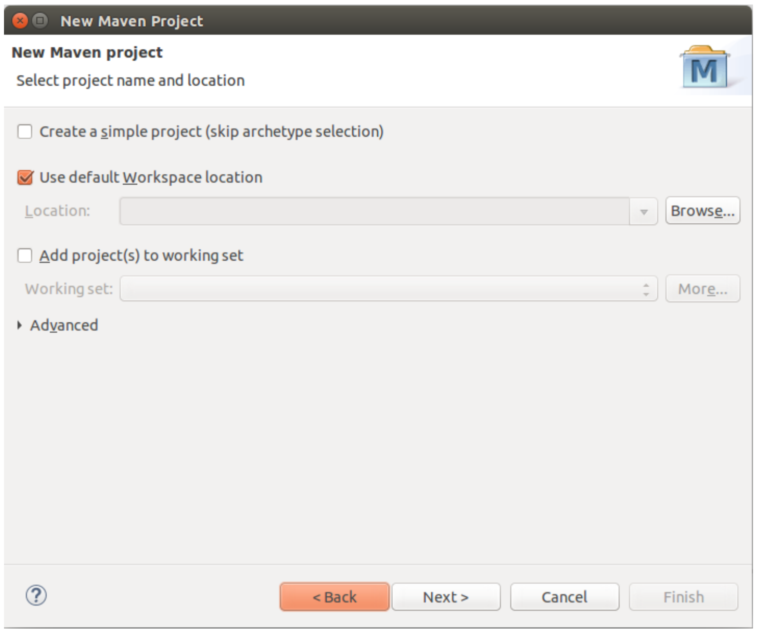
Extract the downloaded file in your current directory, and go inside the newly created directory called 'eclipse-installer', and install eclipse by double clicking on 'eclipse-inst'. In the next prompt window, choose the first option 'Eclipse IDE for Java Developers.' Choose your installation folder and click Install.

3 Hadoop

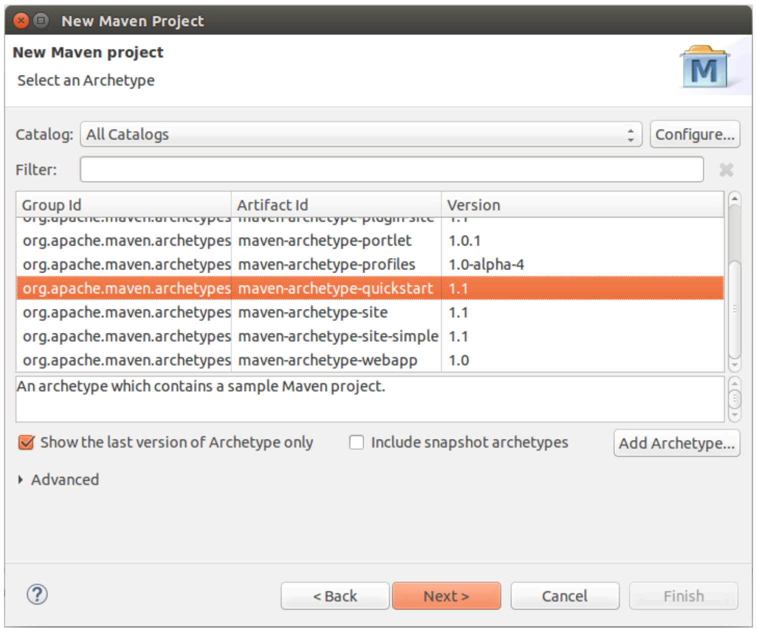
3.1 Creating and managing Maven Projects in Eclipse Mars

3.1.1 Open Eclipse Mars and create a new Maven project

- Create a Maven Project: File->New->Project->Maven->Maven project


New Maven Project
New Maven project
Select project name and location
Create a simple project (skip archetype selection)
Use default Workspace location
Location:
Browse...
Add project(s) to working set
Working set:
More...
Advanced
< Back
Next >
Cancel
Finish

- Use the default Maven Version and click next



Catalog: All Catalogs

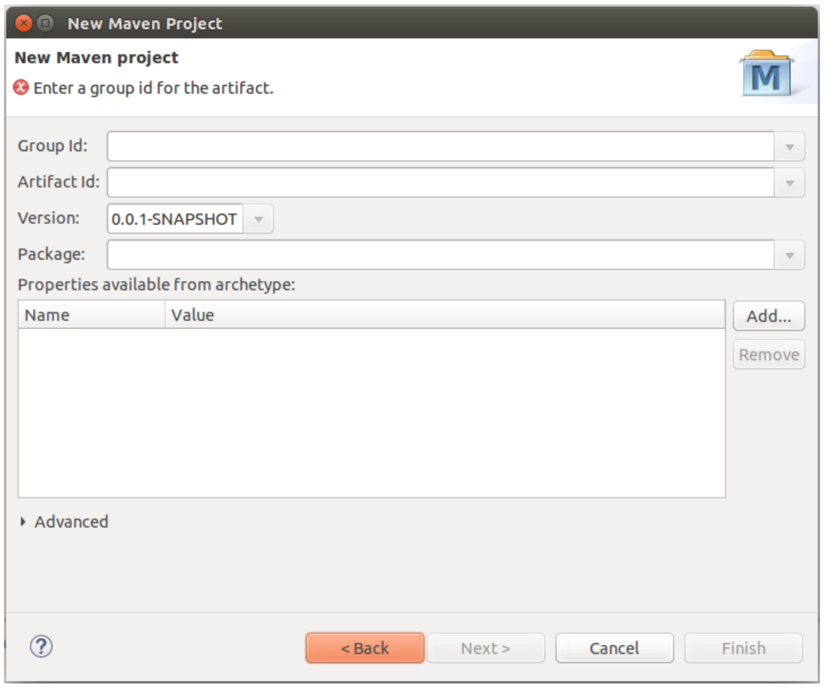
Filter:

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1
org.apache.maven.archetypes	maven-archetype-site-simple	1.1
org.apache.maven.archetypes	maven-archetype-webapp	1.0

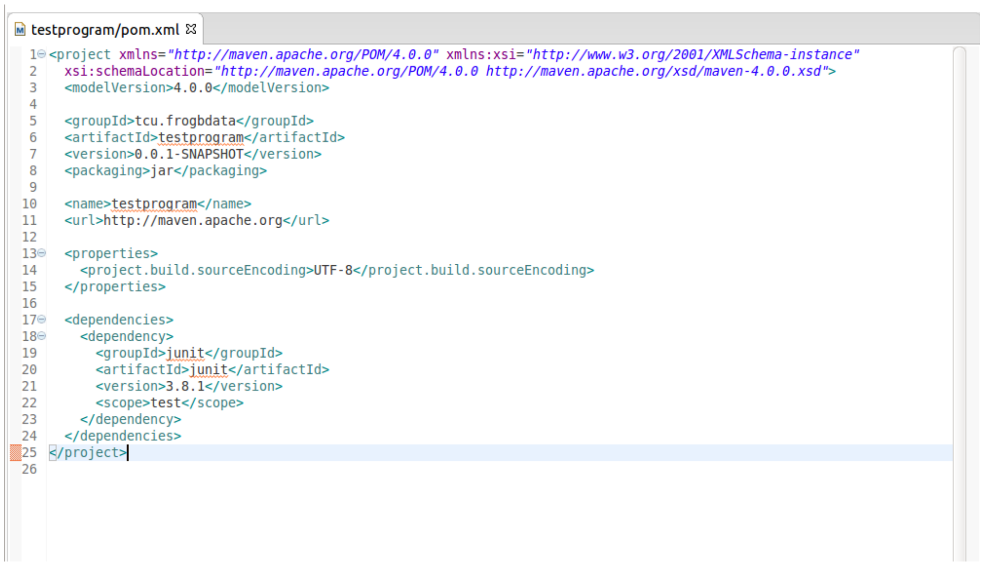
An archetype which contains a sample Maven project.

- Enter the group Id and the artifact Id and the appropriate package name will be generated for you.

  Group ID: identifies your project uniquely among all the projects.
  Artifact ID: is the name of the jar without version.



- POM file
  - POM stands for Project Object Model.
  - It is an XML representation of a Maven project held in a file named 'pom.xml'.
  - This file contains all the information about a project which includes all the plugins required to build the project and the dependencies of a project. Maven manages the list of all the programs and projects that a project depends on through the POM file.
  - By default, your POM file should look like this:

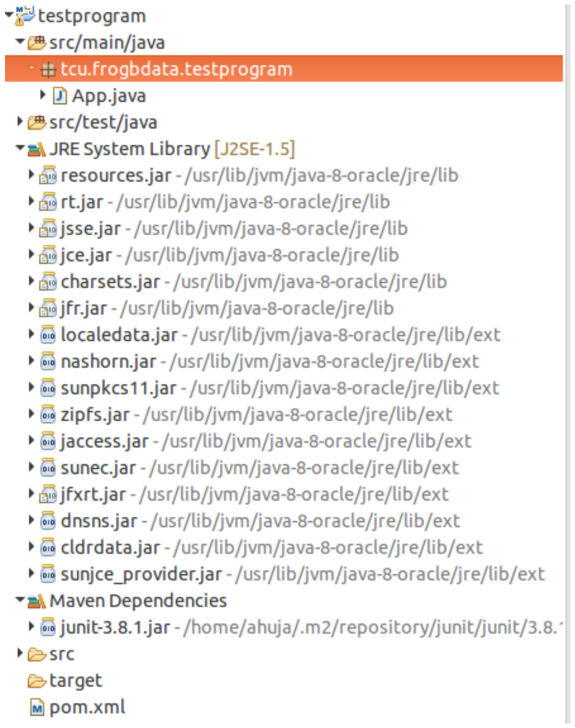


testprogram/pom.xml

```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>tcu.frogbdata</groupId>
  <artifactId>testprogram</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>testprogram</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

- Project Structure
  - Directories

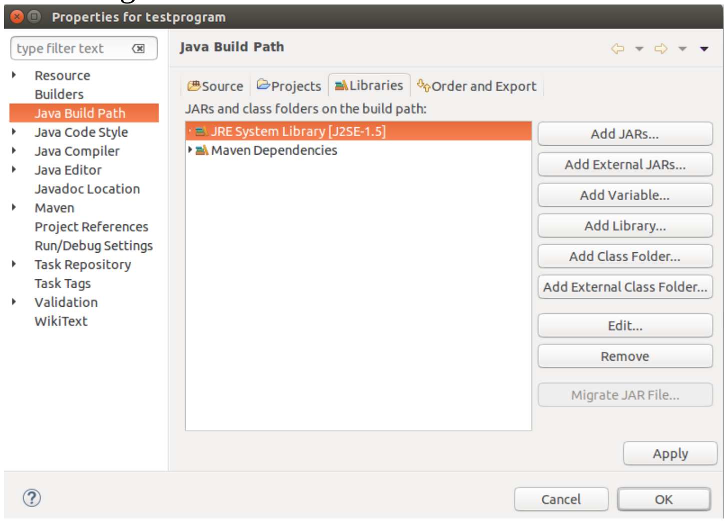
    This is how a typical project directory looks as soon as you make a Maven project. As you keep adding dependencies in POM file, the list here would get bigger.



```
testprogram
  src/main/java
    tcu.frogbdata.testprogram
      App.java
  src/test/java
  JRE System Library [J2SE-1.5]
    resources.jar - /usr/lib/jvm/java-8-oracle/jre/lib
    rt.jar - /usr/lib/jvm/java-8-oracle/jre/lib
    jsse.jar - /usr/lib/jvm/java-8-oracle/jre/lib
    jce.jar - /usr/lib/jvm/java-8-oracle/jre/lib
    charsets.jar - /usr/lib/jvm/java-8-oracle/jre/lib
    jfr.jar - /usr/lib/jvm/java-8-oracle/jre/lib
    localedata.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
    nashorn.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
    sunpkcs11.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
    zipfs.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
    jaccess.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
    sunec.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
    jfxrt.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
    dnsns.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
    cldrdata.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
    sunjce_provider.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
  Maven Dependencies
    junit-3.8.1.jar - /home/ahuja/.m2/repository/junit/junit/3.8.
  src
  target
  pom.xml
```

- Build Path

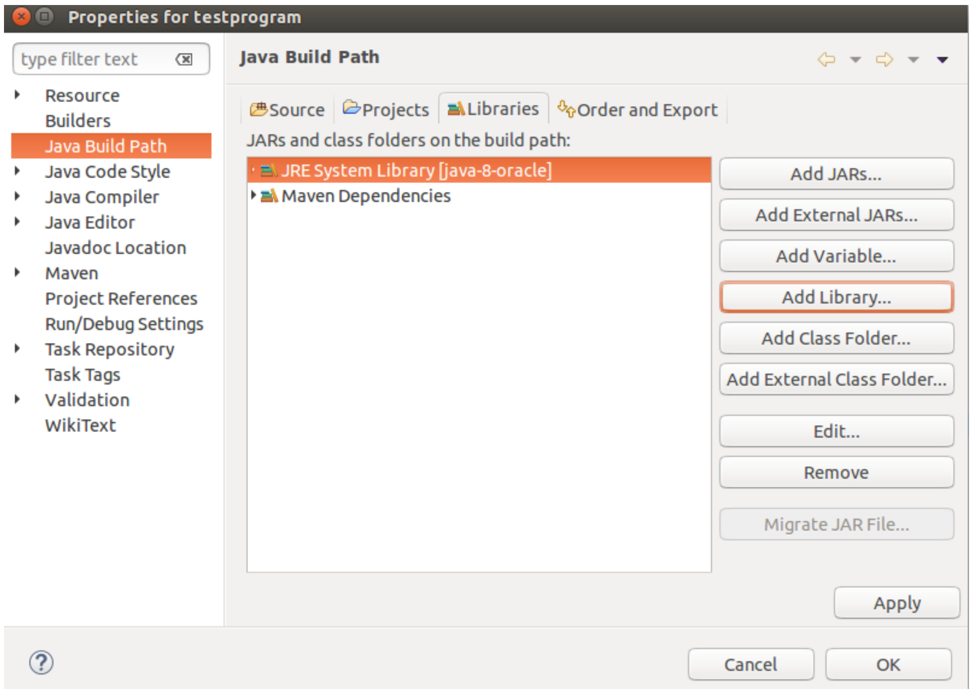
  Remove the default JRE System Library by right clicking on project, Select Build Path->Configure Build Path as shown in the following screenshot. Remove this library by pressing the remove button on the right.



JARs and class folders on the build path:

JRE System Library [J2SE-1.5]
Maven Dependencies

Now, go to Add Library then select JRE System Library from the window prompt, press ‘Next’ and then press ‘Finish’. You should see something similar as depicted in the following screenshot:



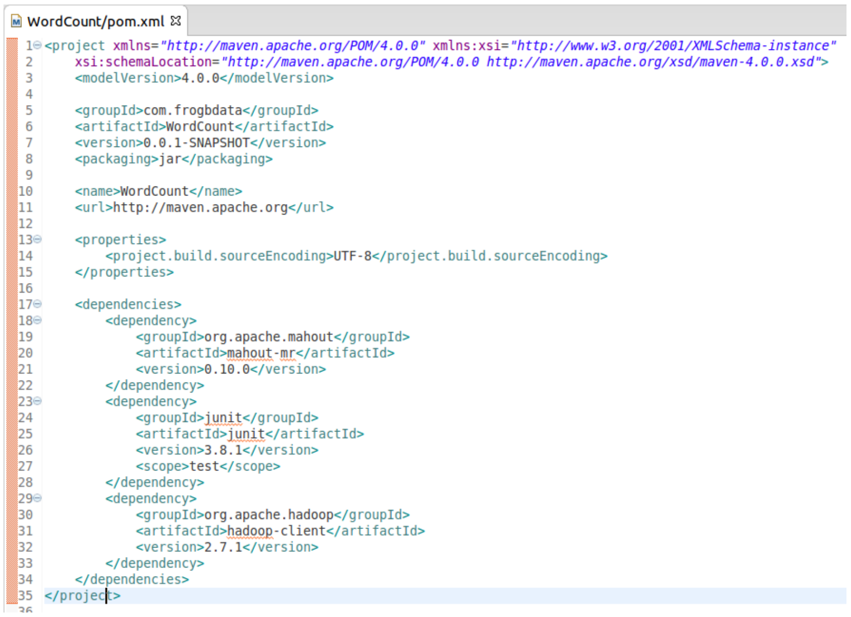
JARs and class folders on the build path:

JRE System Library [java-8-oracle]
Maven Dependencies

3.2 Writing Word Count and Matrix Multiplication programs

3.2.1 Word Count-my first MapReduce program

- Word Count-Hello World of Hadoop
  - Now you are all set to write your first MapReduce program, Word Count.
  - Word Count is called the ‘Hello World’ program of the Big Data development.
  - It is exactly what you think. We will have a text file as the input and we write MapReduce code to count the number of words in that input file.
- Setup the WC project
  - Create a new Maven project named WordCount
  - Edit the pom.xml file for Maven dependencies
    - Hadoop-client
      http://mvnrepository.com/artifact/org.apache.hadoop/hadoop-client/2.7.1
  - Edit the POM file to include the required dependencies. Add the Apache Mahout and Apache Hadoop dependency in pom.xml. Here is a screenshot of what your POM file should look like:



WordCount/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.frogbdata</groupId>
    <artifactId>WordCount</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>WordCount</name>
    <url>http://maven.apache.org</url>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.apache.mahout</groupId>
            <artifactId>mahout-mr</artifactId>
            <version>0.10.0</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-client</artifactId>
            <version>2.7.1</version>
        </dependency>
    </dependencies>
</project>
```

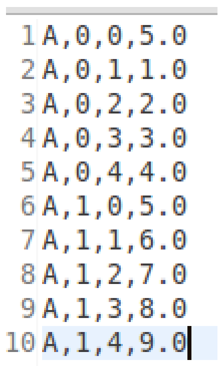
  - Fix the build path
    - Now fix the build path as mentioned earlier in the manual.
- Word Count Algorithm
  - The Map phase in our word count code filters the input data and gives the value of 1 to each word of the text file.

        - The Reduce phase then takes the input from the map phase (filtered data) and then counts the number of time each word occurs (sorting by key).
- Format of input file
    - The input file can be normal text file with words in it. These words can be in any language and in any format.
- Format of output file
    - The output file will have each word and its count in each line and will be sorted alphabetically.
- Congratulations on running your first MapReduce Program

3.2.2 Matrix Multiplication

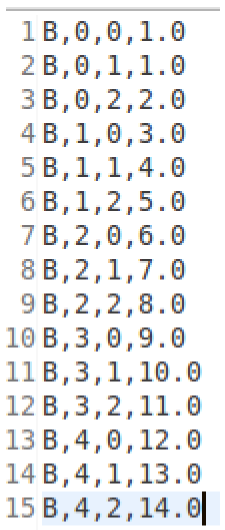
- Matrix Multiplication
    - Next we build matrix multiplication MapReduce Program to multiply two matrices.
- Setup the MM project
    - Create a new Maven project named MatrixMultiply
    - Edit the pom.xml file for Maven dependencies
        - The POM file will be the same as we had for word count as we need Apache Mahout and Apache Hadoop as the dependencies.
    - Fix the build path
        - Fix the build path as we mentioned earlier in the manual.
- Format of input files
    - We will have 2 input files. Both the files will have one matrix each. Each line of the input files will have the name of the matrix, row, column, value.

      For example:



```
A,0,0,5.0
A,0,1,1.0
A,0,2,2.0
A,0,3,3.0
A,0,4,4.0
A,1,0,5.0
A,1,1,6.0
A,1,2,7.0
A,1,3,8.0
A,1,4,9.0
```

Matrix A
(2*5)



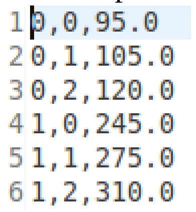
```
B,0,0,1.0
B,0,1,1.0
B,0,2,2.0
B,1,0,3.0
B,1,1,4.0
B,1,2,5.0
B,2,0,6.0
B,2,1,7.0
B,2,2,8.0
B,3,0,9.0
B,3,1,10.0
B,3,2,11.0
B,4,0,12.0
B,4,1,13.0
B,4,2,14.0
```

Matrix B
(5*3)

- Matrix Multiply Algorithm

  - There are two Map functions, one for each matrix. They filter and sort the data according to key and send this sorted data to the reduce phase.
  - The reduce function performs the summary operation of multiplying the appropriate values.
- Format of output file
  - The output will have the final matrix in the form of: row, column, value

    Your output file should look something like this:



```
0,0,95.0
0,1,105.0
0,2,120.0
1,0,245.0
1,1,275.0
1,2,310.0
```

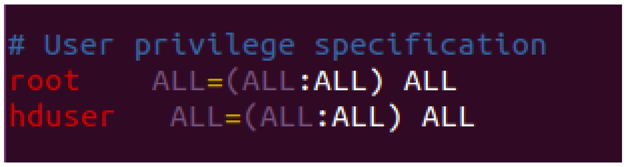
    Result Matrix
    (2*3)
- Congratulations on running your second MapReduce Program

3.3 Install Hadoop on Single Node

3.3.1 Installation instructions

- The following commands will help you set up a dedicated Hadoop user. In this manual, we have used a user called 'hduser', but you can use any other username. Also you would have to get root access for the dedicated user by modifying the sudoers file.

```
sudo addgroup hadoop
sudo adduser --ingroup hadoop hduser
sudo vim /etc/sudoers
```

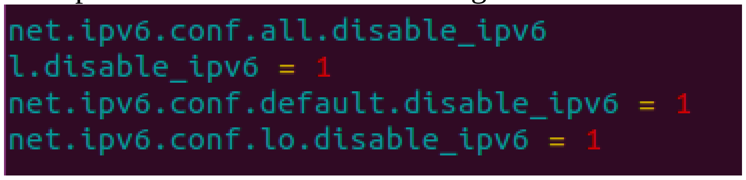


```
# User privilege specification
root    ALL=(ALL:ALL) ALL
hduser   ALL=(ALL:ALL) ALL
```

- Hadoop requires SSH access to manage its nodes i.e. the remote machines in the cluster and the local machine. The following commands would help you install SSH and configure it to allow SSH public key authentication so that you would not have to enter the password each time you try to access a remote machine.
  - If asked for a filename, just leave it blank and press enter to continue.
  - Also, you have to disable IPV6 as the Hadoop version we used does not support it. This will be done by modifying the sysctl.conf file.

```
sudo apt-get install openssh-server
ssh -keygen - t rsa -P “”
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
sudo vim /etc/sysctl.conf
```

  - Update the file with the following lines:



```
net.ipv6.conf.all.disable_ipv6
l.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

  - Once you are done with these commands, reboot the machine with the following command.

```
sudo reboot
```

  - Download Hadoop 2.7.1* from the following link:

http://supergsego.com/apache/hadoop/common/hadoop-2.7.1/hadoop-2.7.1.tar.gz

*__Note__ - During the period of our project, we worked with Hadoop 2.7.1, but we are sure that while reading this manual there will updated versions available. If you want you can go ahead and download the most recent version, but remember to change some commands wherever required.

- The following commands will help you extract the zipped Hadoop folder and move the Hadoop folder to Hadoop folder as per your convenience.

- Then you will have to assign ownership of that folder to the user you chose using the chown command.

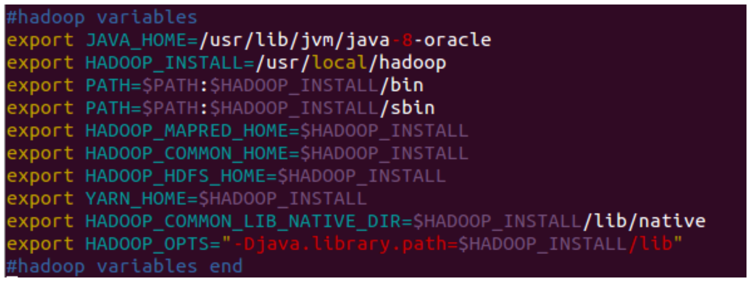
  - Further, we create the namenode and datanode folders in the Hadoop-tmp folder and assign the ownership of this temp folder to the user you chose using the chown command again.

```
tar xvcf hadoop-2.7.1.tar.gz
sudo mv hadoop-2.7.1 /usr/local/hadoop
sudo chown hduser:hadoop -R /usr/local/hadoop
sudo mkdir -p /usr/local/hadoop-tmp/hdfs/namenode
sudo mkdir -p /usr/local/hadoop-tmp/hdfs/datanode
sudo chown hduser:hadoop -R /usr/local/hadoop-tmp
```

- Now we will modify the bash file to include the new variables. The following command will open the bash file.

```
sudo vim ./bashrc
```

  - Append the bash file with the following variables:



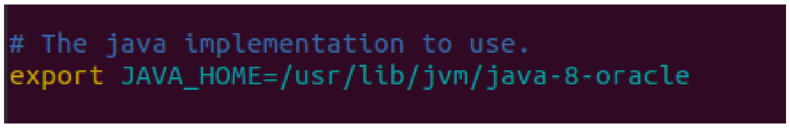
```
#hadoop variables
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
#hadoop variables end
```

- Now we will be modifying some configuration files of Hadoop. Let's open the directory which has all the Hadoop configuration files.

```
cd /usr/local/Hadoop/etc/hadoop
```

  - We need to set JAVA_HOME by modifying the hadoop-env.sh file. This variable in the hadoop-env.sh helps Hadoop finds JAVA on the machine.

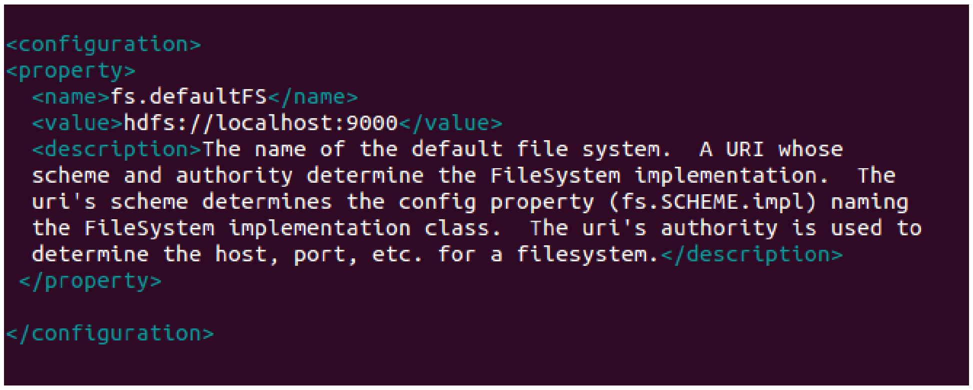
```
vim /usr/local/hadoop/etc/hadoop/hadoop-env.sh
```



```
# The java implementation to use.
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
```

- Modify the core-site.xml file. This file contains the properties that override the default core properties. The only property that we change is the url of the default file system.

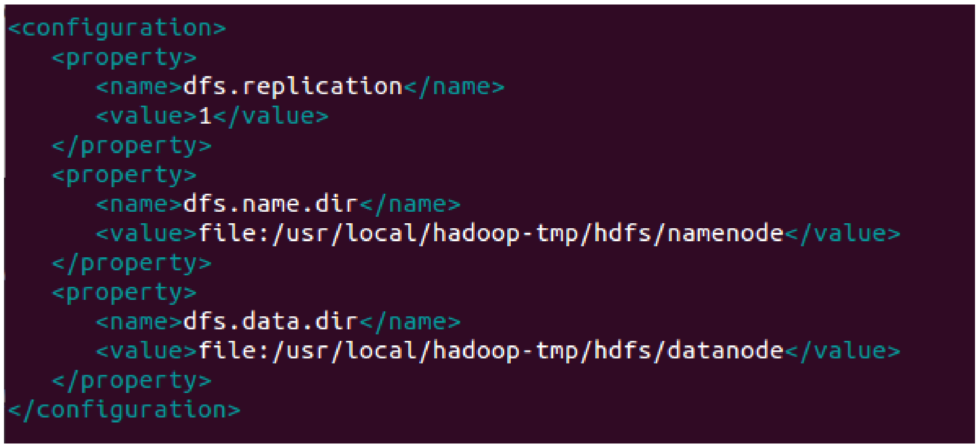
```
vim /usr/local/hadoop/etc/hadoop/core-site.xml
```



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

- Now we modify the hdfs-site.xml. We change the value of dfs.replication (default block replication) to 1 as we are running Hadoop on a single node right now.
    - We also change the value of dfs.name.dir and dfs.data.dir which specify where on the local filesystem the DFS namenode and DFS datanode blocks should be stored.

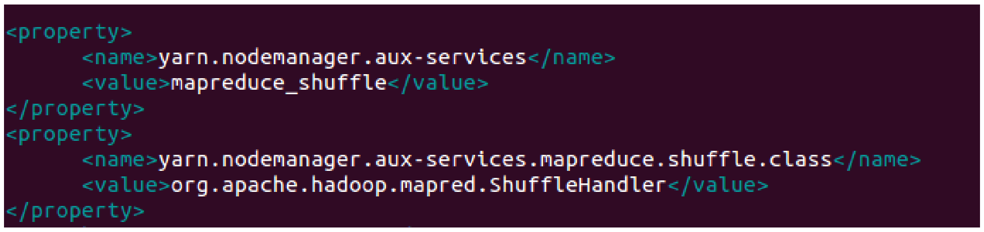
```
vim /usr/local/hadoop/etc/hadoop/hdfs-site.xml
```



```
<configuration>
   <property>
      <name>dfs.replication</name>
      <value>1</value>
   </property>
   <property>
      <name>dfs.name.dir</name>
      <value>file:/usr/local/hadoop-tmp/hdfs/namenode</value>
   </property>
   <property>
      <name>dfs.data.dir</name>
      <value>file:/usr/local/hadoop-tmp/hdfs/datanode</value>
   </property>
</configuration>
```

- Now modify the yarn-site.xml. In this we set up yarn to work with MapReduce Jobs.

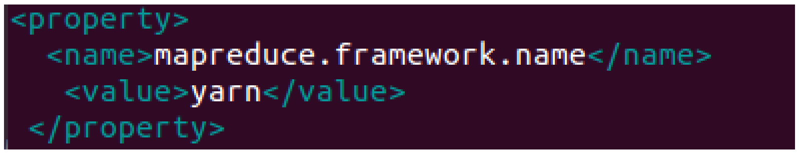
```
vim /usr/local/hadoop/etc/hadoop/yarn-site.xml
```



```
<property>
      <name>yarn.nodemanager.aux-services</name>
      <value>mapreduce_shuffle</value>
</property>
<property>
      <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
      <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```

- Modify the mapred-site.xml. We only change the value of the MapReduce framework to yarn as we yarn as our runtime framework for executing MapReduce jobs.

```
vim /usr/local/hadoop/etc/hadoop/mapred-site.xml
```



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

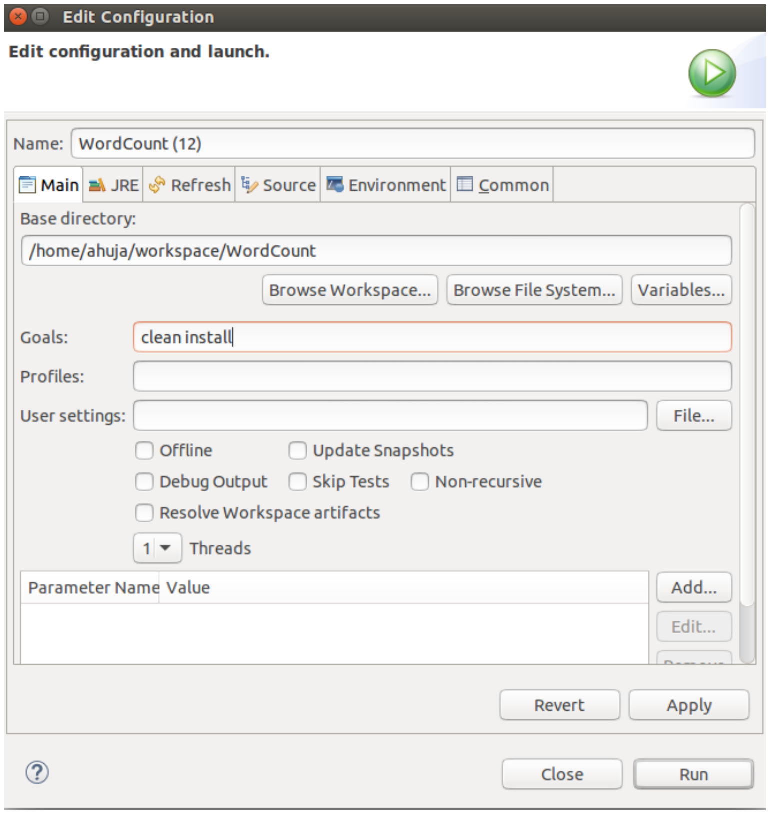
```
sudo reboot
start-dfs.sh
start-yarn.sh
```

- Note – If you were guided here from Spark single node (with HDFS) installation [Click here]() to continue.

3.4 Deploy WC and MM on single Node Hadoop

3.4.1 Export JAR file with all dependencies

- Right-click on the project -> Run As -> Maven Build
    - On the pop-up window, write 'clean install' in the Goals text box and click run. Here is a screenshot of what it should it look like.



Name: WordCount (12)

Base directory: /home/ahuja/workspace/WordCount

Goals: clean install

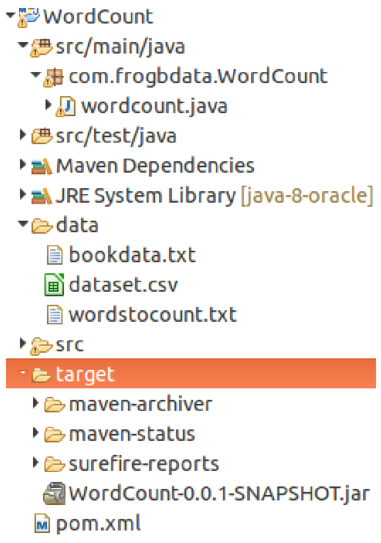
Profiles:

User settings:

☐ Offline ☐ Update Snapshots
☐ Debug Output ☐ Skip Tests ☐ Non-recursive
☐ Resolve Workspace artifacts
1 Threads

Parameter Name Value

- You should see the progress of Maven building the project on the console.
- You can find the newly created jar file of the project inside the 'target' folder of the project.



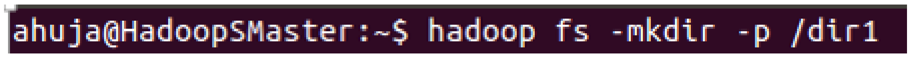
```
WordCount
  src/main/java
    com.frogbdata.WordCount
      wordcount.java
  src/test/java
  Maven Dependencies
  JRE System Library [java-8-oracle]
  data
    bookdata.txt
    dataset.csv
    wordstocount.txt
  src
  target
    maven-archiver
    maven-status
    surefire-reports
    WordCount-0.0.1-SNAPSHOT.jar
  pom.xml
```

Note: All these 4 steps have to be followed in case of both: word count and matrix multiplication.

3.4.2 Put the input file on HDFS

- Now we have to put our input file(s) on the HDFS before running the job on the Single –Node Hadoop.
  - Before putting the file on the HDFS, we need to create a directory on HDFS. The following command will help you do that.

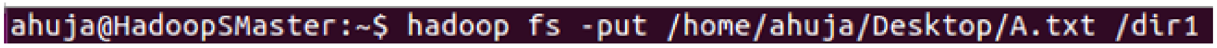
```
hadoop fs –mkdir –p directory_path
```



```
ahuja@HadoopSMaster:~$ hadoop fs -mkdir -p /dir1
```

The command to put any file on HDFS is:

```
hadoop fs -put path_to_local_file path_on_hdfs
```



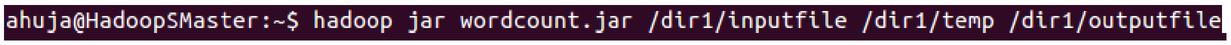
```
ahuja@HadoopSMaster:~$ hadoop fs -put /home/ahuja/Desktop/A.txt /dir1
```

In the case of word count, you will have to put only one input file as it requires only one input file, but in the case of matrix multiplication you will need to put both the files as mentioned earlier in the manual.

3.4.3 Run the job through terminal

- Running the job from the terminal is easy. The following is the generalized command to do that.

```
hadoop jar path_of_jar package_name.classname args
```



```
ahuja@HadoopSMaster:~$ hadoop jar wordcount.jar /dir1/inputfile /dir1/temp /dir1/outputfile
```

3.4.4 Access the output from HDFS and interpret results

- Accessing the output from HDFS requires you to open the Hadoop Web interface.
  Click on Utilities -> Browse the filesystem.
  Find your output directory and you have the option of downloading the output file or opening it.

3.5 Setup a YARN Cluster

3.5.1 Why cluster?

- Hadoop was never built to run on a single node. There is no point of doing that except when you are testing your MapReduce code.
- Hadoop becomes powerful and runs at its best in a cluster of nodes.
- Parallel processing is only possible in a cluster where the data can be distributed among different nodes (datanodes) which are controlled by a manager node (namenode).

3.5.2 Structure of our cluster

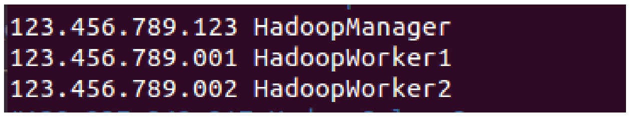
- Our Cluster had 3 nodes, each with a configuration as follows:
  - 8 GB RAM
  - 500 GB HDD
  - Ubuntu 15.04
- Manager-Worker Architecture
  - 1 Manager
  - 2 Workers

3.5.3 Install Hadoop on 2 separate machines (workers) Using 3.3

- Change the appropriate configuration files

```
sudo vim /etc/hosts
```

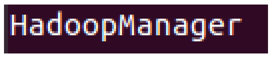
  - This is just a sample. Use appropriate names and IP addresses based on your machines. The hosts file will have to be changed on all the nodes of the cluster.



```
123.456.789.123 HadoopManager
123.456.789.001 HadoopWorker1
123.456.789.002 HadoopWorker2
```

  - Modify the masters file on all the nodes. This file tells hadoop which machine in the cluster is the manager.

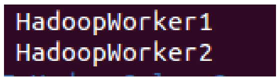
```
vim /usr/local/hadoop/etc/hadoop/masters
```



```
HadoopManager
```

  - Modify the slaves file on all the nodes. This file lists all the worker nodes of the cluster.

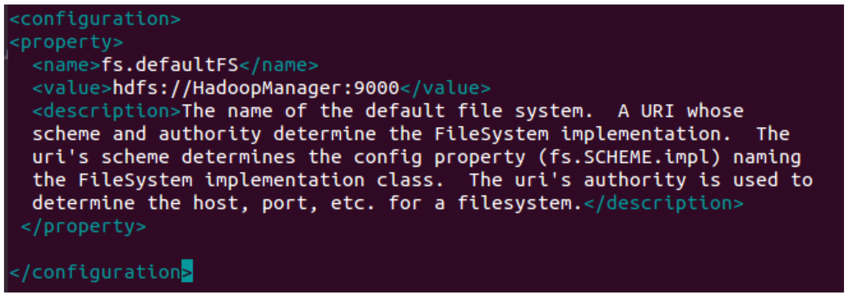
```
vim /usr/local/hadoop/etc/hadoop/slaves
```



```
HadoopWorker1
HadoopWorker2
```

  - We will have to modify the core-site.xml file to change the url of the default filesystem as it cannot be localhost now.

```
vim /usr/local/hadoop/etc/hadoop/core-site.xml
```



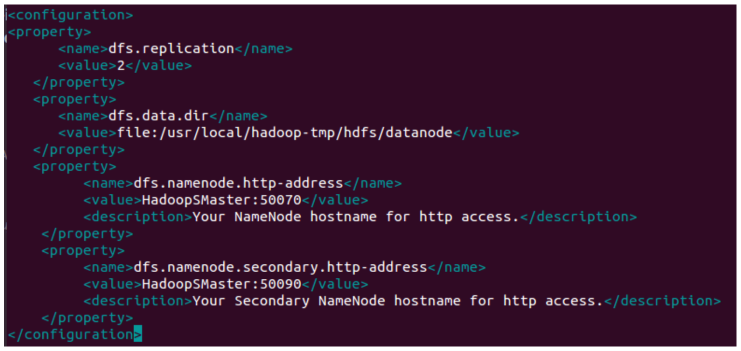
```
<configuration>
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://HadoopManager:9000</value>
  <description>The name of the default file system.  A URI whose
  scheme and authority determine the FileSystem implementation.  The
  uri's scheme determines the config property (fs.SCHEME.impl) naming
  the FileSystem implementation class.  The uri's authority is used to
  determine the host, port, etc. for a filesystem.</description>
 </property>

</configuration>
```

- We modify the hdfs-site.xml file to change the value of dfs.replication as we will have 2 data nodes (2 workers) now.

  And also we assign the https address of the namenode which will provide us with the web interface of the HDFS filesystem.

- We may or may not have specify the data.dir property depending on the node, if it is a data node we need to have that property or else not. If you want to have your manager node as a data node too, then you will need to specify this property in manager node too.

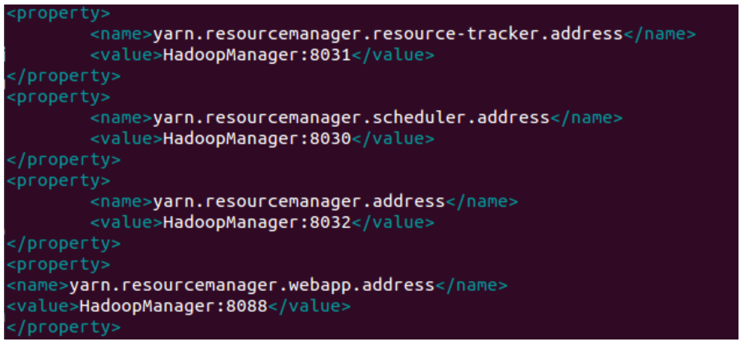
vim /usr/local/hadoop/etc/hadoop/hdfs-site.xml



```
<configuration>
<property>
      <name>dfs.replication</name>
      <value>2</value>
   </property>
   <property>
      <name>dfs.data.dir</name>
      <value>file:/usr/local/hadoop-tmp/hdfs/datanode</value>
   </property>
   <property>
         <name>dfs.namenode.http-address</name>
         <value>HadoopSMaster:50070</value>
         <description>Your NameNode hostname for http access.</description>
    </property>
    <property>
         <name>dfs.namenode.secondary.http-address</name>
         <value>HadoopSMaster:50090</value>
         <description>Your Secondary NameNode hostname for http access.</description>
    </property>
</configuration>
```

- Modify the yarn-site.xml to give URL values to resource manager of yarn for easy access from the web.

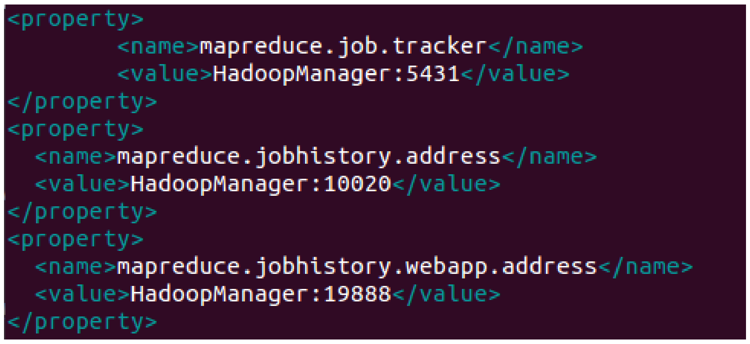
vim /usr/local/hadoop/etc/hadoop/yarn-site.xml



```
<property>
        <name>yarn.resourcemanager.resource-tracker.address</name>
        <value>HadoopManager:8031</value>
</property>
<property>
        <name>yarn.resourcemanager.scheduler.address</name>
        <value>HadoopManager:8030</value>
</property>
<property>
        <name>yarn.resourcemanager.address</name>
        <value>HadoopManager:8032</value>
</property>
<property>
<name>yarn.resourcemanager.webapp.address</name>
<value>HadoopManager:8088</value>
</property>
```

- Modify the mapred-site.xml file to give url values to the MapReduce job tracker and Job history web access.

```
vim /usr/local/hadoop/etc/hadoop/mapred-site.xml
```

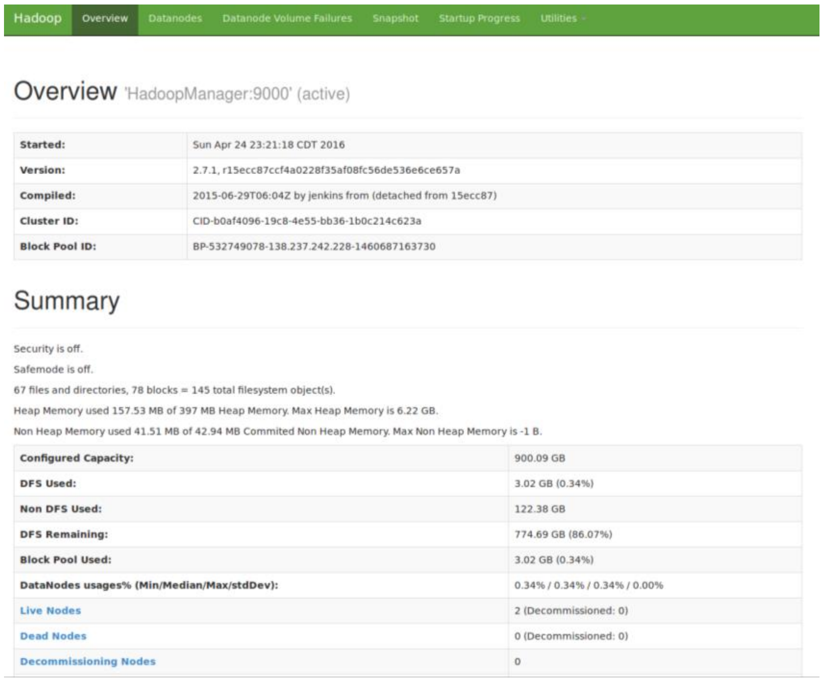


```
<property>
        <name>mapreduce.job.tracker</name>
        <value>HadoopManager:5431</value>
</property>
<property>
  <name>mapreduce.jobhistory.address</name>
  <value>HadoopManager:10020</value>
</property>
<property>
  <name>mapreduce.jobhistory.webapp.address</name>
  <value>HadoopManager:19888</value>
</property>
```

- Once you are finished with all these modifications in the configuration files, rebbot your machine and start Hadoop!

```
sudo reboot
start-dfs.sh
start-yarn.sh
```

- Here is the overview of the Web interface of HadoopManager. The url would be hadoopmanager: 50070.



Overview 'HadoopManager:9000' (active)

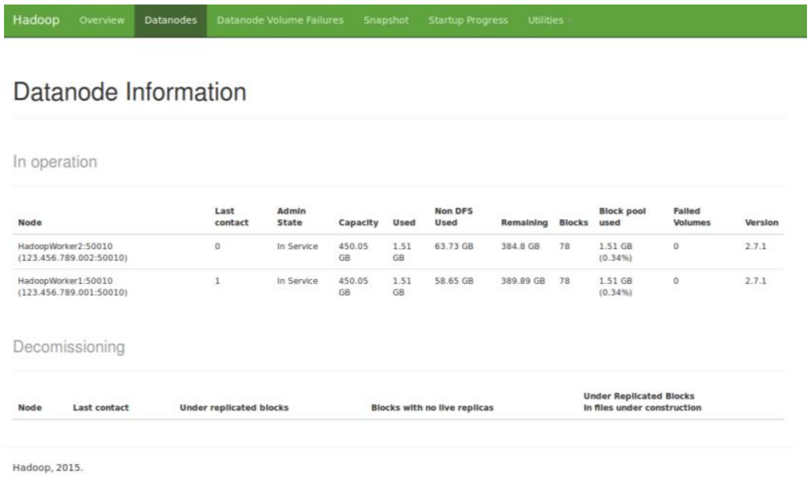
Started:	Sun Apr 24 23:21:18 CDT 2016
Version:	2.7.1, r15ecc87ccf4a0228f35af08fc56de536e6ce657a
Compiled:	2015-06-29T06:04Z by jenkins from (detached from 15ecc87)
Cluster ID:	CID-b0af4096-19c8-4e55-bb36-1b0c214c623a
Block Pool ID:	BP-532749078-138.237.242.228-1460687163730

Summary

Security is off.
Safemode is off.
67 files and directories, 78 blocks = 145 total filesystem object(s).
Heap Memory used 157.53 MB of 397 MB Heap Memory. Max Heap Memory is 6.22 GB.
Non Heap Memory used 41.51 MB of 42.94 MB Commited Non Heap Memory. Max Non Heap Memory is -1 B.

Configured Capacity:	900.09 GB
DFS Used:	3.02 GB (0.34%)
Non DFS Used:	122.38 GB
DFS Remaining:	774.69 GB (86.07%)
Block Pool Used:	3.02 GB (0.34%)
DataNodes usages% (Min/Median/Max/stdDev):	0.34% / 0.34% / 0.34% / 0.00%
Live Nodes	2 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0

Click on data nodes to check your data nodes and their current state.



Datanode Information

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
HadoopWorker2:50010 (123.456.789.002:50010)	0	In Service	450.05 GB	1.51 GB	63.73 GB	384.8 GB	78	1.51 GB (0.34%)	0	2.7.1
HadoopWorker1:50010 (123.456.789.001:50010)	1	In Service	450.05 GB	1.51 GB	58.65 GB	389.89 GB	78	1.51 GB (0.34%)	0	2.7.1

Decomissioning

Node	Last contact	Under replicated blocks	Blocks with no live replicas	Under Replicated Blocks In files under construction

Hadoop, 2015.

Click here to continue YARN cluster installation for Spark.

3.6 Deploy WC and MM on Cluster

3.6.1 Store the input files on HDFS

- The command to upload a file on the HDFS in a cluster remains the same as in case of Single-Node Hadoop.

```
hadoop fs -put path_to_local_file path_on_hdfs
```

- Run the jobs through terminal

  The command to run a MapReduce Job on the cluster also remains the same as in case of Single-Node Hadoop.

```
hadoop jar path_of_jar package_name.classname args
```

- Access the output from the HDFS

  The method of accessing the output file(s) from the HDFS remains the same as in case of Single-Node Hadoop i.e. through the web interface.

3.7 Recommender

3.7.1 What is a recommender?

- Now we come to the most interesting part of the project where we build our own recommendation system!
- We are sure that you might have come across some mind of recommendation system in your lives.
- The most popular ones are Netflix, Amazon, and Facebook etc.
- If any of you who do not what a recommender is, it is a system which recommends you objects based on your history of 'likes' and ratings.

3.7.2 Recommender types

- There are two basic types of recommendation engine algorithms:
  - User-based: The items will be suggested to the user based on what other people with similar tastes seem to like.
  - Item-based: The items will be suggested to the user based on finding similar items to the ones the user already likes, again by looking to others' apparent preferences.

3.7.3 Co-occurrence algorithm

- We used the co-occurrence algorithm with the help of Apache Mahout to build the recommendation system in Hadoop. This type of algorithm is Item-based.
- In this algorithm, we start by creating a co-occurrence matrix. It is not as hard as it sounds!
- We begin by finding some degree of similarity between any pair of items. Imagine computing a similarity for every pair of items and putting the results into a giant matrix. This is called a co-occurrence matrix.

- This matrix describes associations between items, and has nothing to do with the users. It computes the number of times each pair of items occur together in some user's list of preferences.
- For example, if there are 17 users who express liking for both items A and B, then A and B co-occur 17 times.
- Co-occurrence is like similarity, the more two items turn up together, the more related/similar they are.
- The next step is to compute a user vector for each user. This vector tells which movie has a specific user watched and which ones has he not.
- The final step to get the recommendations is to multiply the co-occurrence matrix with the user vector.

3.7.4 Implementing Co-occurrence recommender

- We used the movie lens data for our recommender. Here is a link of the movie lens data: http://grouplens.org/datasets/movielens/
- The original format of the movie lens data is:
  userId *movieId* *rating* *timestamp*
- We convert* this format to the following format:
  userId,movieId

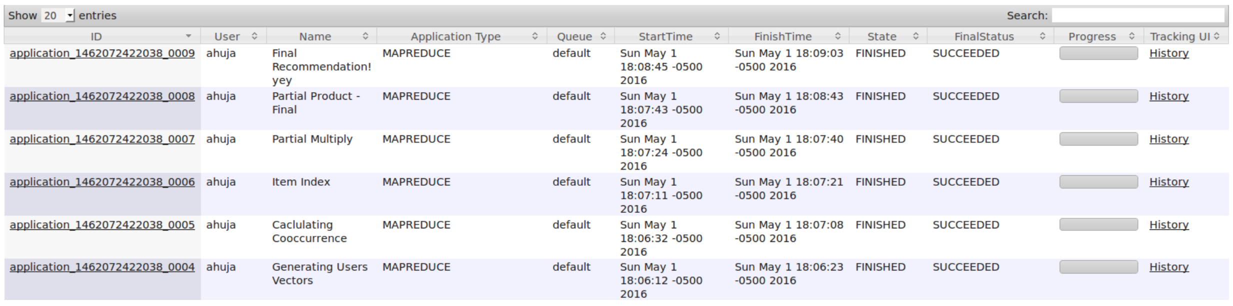
  *Note – The source code to convert from the movie lens format to our format can be found on the link:
  http://brazos.cs.tcu.edu/1516frogbdata/deliverables.html

- We do this because our recommender in Hadoop is not rating-based, rather it is item-based. It only takes into account the movies watched by the user.
- In the next step, you would have to upload the input file with the users and the movies they have watched on the HDFS using the hadoop 'put' command as mentioned earlier in the manual.
- You can find the source code of our whole recommender on the link:
  http://brazos.cs.tcu.edu/1516frogbdata/deliverables.html

- The command to run our recommender is as follows:

```
hadoop jar path_of_jar_file frogbdata.tcu.recommendereg.RecommenderJobRun
path_of_input_file_HDFS path_of_output_file_HDFS num_of_recommendations
```

- There will be 6 jobs that will run in order to give the recommendations. Here is a screenshot of what it should look like:

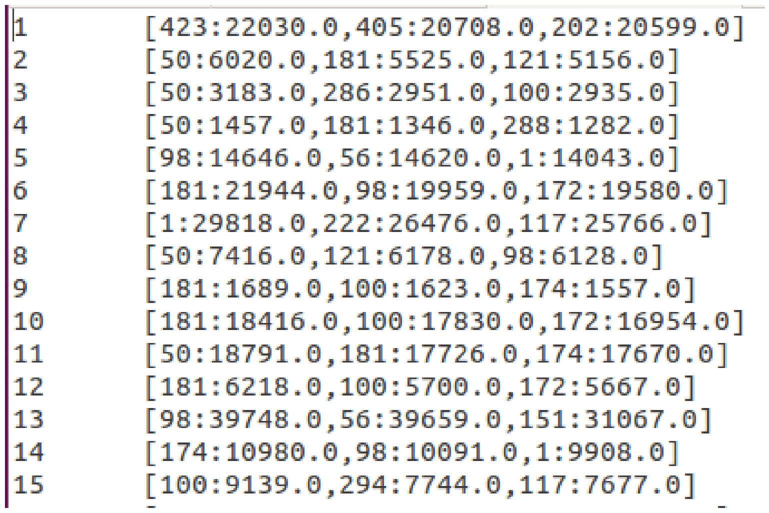


Show 20 entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1462072422038_0009	ahuja	Final Recommendation! yey	MAPREDUCE	default	Sun May 1 18:08:45 -0500 2016	Sun May 1 18:09:03 -0500 2016	FINISHED	SUCCEEDED		History
application_1462072422038_0008	ahuja	Partial Product - Final	MAPREDUCE	default	Sun May 1 18:07:43 -0500 2016	Sun May 1 18:08:43 -0500 2016	FINISHED	SUCCEEDED		History
application_1462072422038_0007	ahuja	Partial Multiply	MAPREDUCE	default	Sun May 1 18:07:24 -0500 2016	Sun May 1 18:07:40 -0500 2016	FINISHED	SUCCEEDED		History
application_1462072422038_0006	ahuja	Item Index	MAPREDUCE	default	Sun May 1 18:07:11 -0500 2016	Sun May 1 18:07:21 -0500 2016	FINISHED	SUCCEEDED		History
application_1462072422038_0005	ahuja	Caclulating Cooccurrence	MAPREDUCE	default	Sun May 1 18:06:32 -0500 2016	Sun May 1 18:07:08 -0500 2016	FINISHED	SUCCEEDED		History
application_1462072422038_0004	ahuja	Generating Users Vectors	MAPREDUCE	default	Sun May 1 18:06:12 -0500 2016	Sun May 1 18:06:23 -0500 2016	FINISHED	SUCCEEDED		History

  - Generating Users Vectors: generates a user vector for each user which shows what movies has a specific user watched.
  - Calculating co-occurrence: This job calculates the co-occurrences between every pair of movies. This job might take a while.
  - Item Index: This job has a quick map function to get the index of each item.
  - Partial Multiply: This job wraps the vectors and gets them ready for partial multiply.
  - Partial Product- Final: This is where we find the product of the matrices. This Job takes the longest time among all of them.
  - Final recommendation: this job sorts the recommendations and removes the movies that the user has already watched and gives the final recommendations.
- The output file will be on the HDFS in the directory you mentioned in the command to run the job.
- The output file will have the recommendations for each user and each user will have a value attached to it.

  Here is what it should look like:



```
1       [423:22030.0,405:20708.0,202:20599.0]
2       [50:6020.0,181:5525.0,121:5156.0]
3       [50:3183.0,286:2951.0,100:2935.0]
4       [50:1457.0,181:1346.0,288:1282.0]
5       [98:14646.0,56:14620.0,1:14043.0]
6       [181:21944.0,98:19959.0,172:19580.0]
7       [1:29818.0,222:26476.0,117:25766.0]
8       [50:7416.0,121:6178.0,98:6128.0]
9       [181:1689.0,100:1623.0,174:1557.0]
10      [181:18416.0,100:17830.0,172:16954.0]
11      [50:18791.0,181:17726.0,174:17670.0]
12      [181:6218.0,100:5700.0,172:5667.0]
13      [98:39748.0,56:39659.0,151:31067.0]
14      [174:10980.0,98:10091.0,1:9908.0]
15      [100:9139.0,294:7744.0,117:7677.0]
```

3.8 K-Means Clustering

3.8.1 What is clustering?

- As the name suggests, Clustering implies grouping items together. But on the basis of what?
- We use clustering in Big Data industry to group similar items/users together for better data management.
- It has numerous applications in the business and helps them to target their customers in a better way.
- K-Means algorithm is a simple, but widely used algorithm used for clustering. All objects need to be represented as a set of numerical features. Also, the user has to specify the number of groups (k) he wants.

3.8.2 Application of clustering

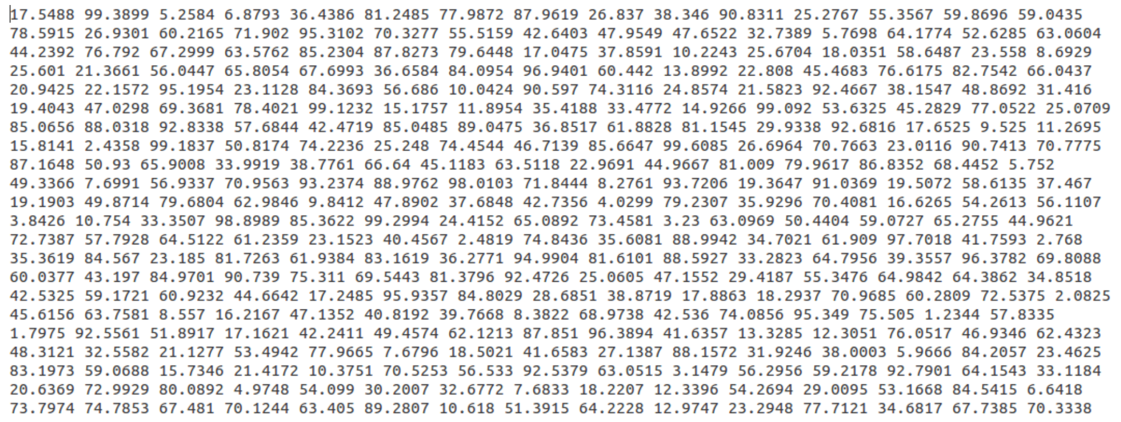
- Have you ever been creeped out of the advertisements on applications like Facebook, YouTube or any other website you visit? If yes, it is because of clustering. If not, then look carefully!
- The advertisements have become personalized to a great extent. If you searching pattern on google shows that you read a lot about Big Data, notice how you are flooded with Big Data company advertisements and jobs. It happened with us!
- There is no magic that goes behind this. The clustering algorithms help group similar kind of users together.
- This is how a bank decides to give you a loan or not. Based on your social security, past credit and other financial details they put you in an appropriate group.

3.8.3 Algorithm

- The first step is to choose the number of clusters (k). For example, the user chooses to have 5 clusters.
- In the next step, we choose 5 random records as the centroids of these 5 clusters.
- Now we have 5 cluster centroids but they are randomly chosen.
- Next, we assign each record/point to a specific cluster based on the distance of that point with the centroid of that cluster.
- Next, we compute the distances from each point and allot points to the cluster where the distance from the centroid is minimum.
- Once we are done with re-allotting the points to their new and final clusters, we create separate files for each cluster and save them into HDFS.

3.8.4 Implementing K-Means clustering

- The input files used for K-Means clustering were generated randomly from our code. The input file will be multi-dimensional and each line will have at least 15 records. Here is what our file looked like:



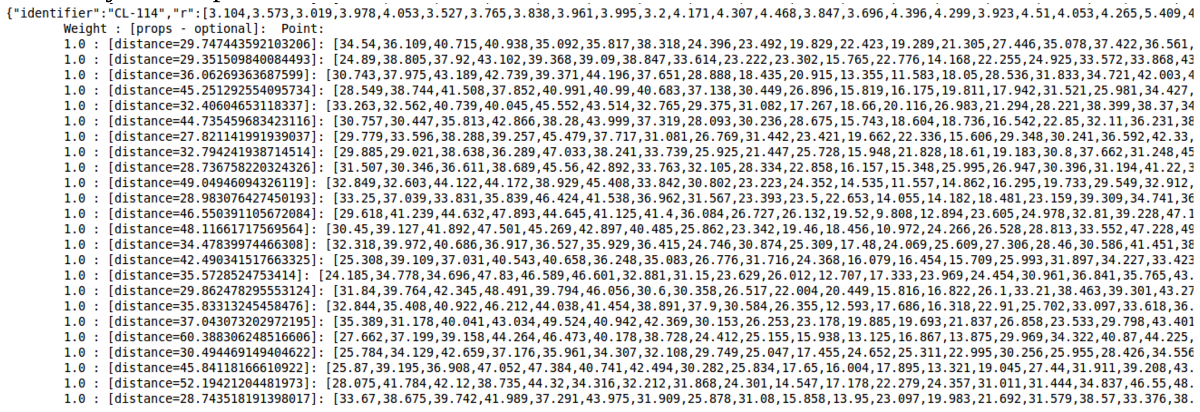
```
17.5488 99.3899 5.2584 6.8793 36.4386 81.2485 77.9872 87.9619 26.837 38.346 90.8311 25.2767 55.3567 59.8696 59.0435
78.5915 26.9301 60.2165 71.902 95.3102 70.3277 55.5159 42.6403 47.9549 47.6522 32.7389 5.7698 64.1774 52.6285 63.0604
44.2392 76.792 67.2999 63.5762 85.2304 87.8273 79.6448 17.0475 37.8591 10.2243 25.6704 18.0351 58.6487 23.558 8.6929
25.601 21.3661 56.0447 65.8054 67.6993 36.6584 84.0954 96.9401 60.442 13.8992 22.808 45.4683 76.6175 82.7542 66.0437
20.9425 22.1572 95.1954 23.1128 84.3693 56.686 10.0424 90.597 74.3116 24.8574 21.5823 92.4667 38.1547 48.8692 31.416
19.4043 47.0298 69.3681 78.4021 99.1232 15.1757 11.8954 35.4188 33.4772 14.9266 99.092 53.6325 45.2829 77.0522 25.0709
85.0656 88.0318 92.8338 57.6844 42.4719 85.0485 89.0475 36.8517 61.8828 81.1545 29.9338 92.6816 17.6525 9.525 11.2695
15.8141 2.4358 99.1837 50.8174 74.2236 25.248 74.4544 46.7139 85.6647 99.6085 26.6964 70.7663 23.0116 90.7413 70.7775
87.1648 50.93 65.9008 33.9919 38.7761 66.64 45.1183 63.5118 22.9691 44.9667 81.009 79.9617 86.8352 68.4452 5.752
49.3366 7.6991 56.9337 70.9563 93.2374 88.9762 98.0103 71.8444 8.2761 93.7206 19.3647 91.0369 19.5072 58.6135 37.467
19.1903 49.8714 79.6804 62.9846 9.8412 47.8902 37.6848 42.7356 4.0299 79.2307 35.9296 70.4081 16.6265 54.2613 56.1107
3.8426 10.754 33.3507 98.8989 85.3622 99.2994 24.4152 65.0892 73.4581 3.23 63.0969 50.4404 59.0727 65.2755 44.9621
72.7387 57.7928 64.5122 61.2359 23.1523 40.4567 2.4819 74.8436 35.6081 88.9942 34.7021 61.909 97.7018 41.7593 2.768
35.3619 84.567 23.185 81.7263 61.9384 83.1619 36.2771 94.9904 81.6101 88.5927 33.2823 64.7956 39.3557 96.3782 69.8088
60.0377 43.197 84.9701 90.739 75.311 69.5443 81.3796 92.4726 25.0605 47.1552 29.4187 55.3476 64.9842 64.3862 34.8518
42.5325 59.1721 60.9232 44.6642 17.2485 95.9357 84.8029 28.6851 38.8719 17.8863 18.2937 70.9685 60.2809 72.5375 2.0825
45.6156 63.7581 8.557 16.2167 47.1352 40.8192 39.7668 8.3822 68.9738 42.536 74.0856 95.349 75.505 1.2344 57.8335
1.7975 92.5561 51.8917 17.1621 42.2411 49.4574 62.1213 87.851 96.3894 41.6357 13.3285 12.3051 76.0517 46.9346 62.4323
48.3121 32.5582 21.1277 53.4942 77.9665 7.6796 18.5021 41.6583 27.1387 88.1572 31.9246 38.0003 5.9666 84.2057 23.4625
83.1973 59.0688 15.7346 21.4172 10.3751 70.5253 56.533 92.5379 63.0515 3.1479 56.2956 59.2178 92.7901 64.1543 33.1184
20.6369 72.9929 80.0892 4.9748 54.099 30.2007 32.6772 7.6833 18.2207 12.3396 54.2694 29.0095 53.1668 84.5415 6.6418
73.7974 74.7853 67.481 70.1244 63.405 89.2807 10.618 51.3915 64.2228 12.9747 23.2948 77.7121 34.6817 67.7385 70.3338
```

It is just a bunch of numbers! This is unstructured data and it might be a part of an excel file and now we will cluster each and every record.

- In the next step you would put this file on the HDFS using the hadoop put command.
- Again, you can find our source code for K-Means clustering on the following link:

  http://brazos.cs.tcu.edu/1516frogbdata/deliverables.html
- Next, you would run the job using the following command

  ```
  hadoop jar path_of_jar_file apache.KMJob path_of_input_file_HDFS
  path_of_output_file_HDFS
  ```
- Here is what your output should look like:



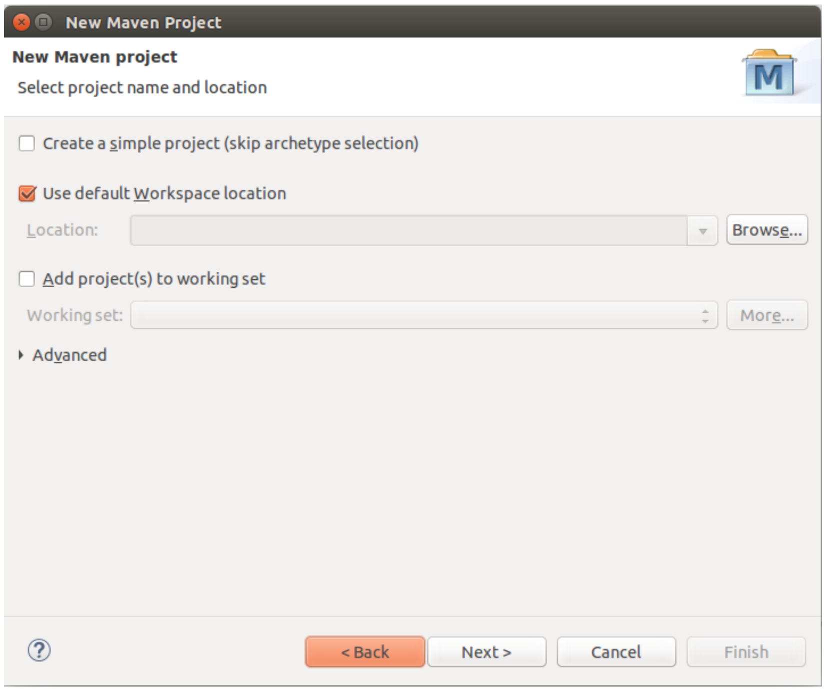
```
{"identifier":"CL-114","r":[3.104,3.573,3.019,3.978,4.053,3.527,3.765,3.838,3.961,3.995,3.2,4.171,4.307,4.468,3.847,3.696,4.396,4.299,3.923,4.51,4.053,4.265,5.409,4
        Weight : [props - optional]:  Point:
        1.0 : [distance=29.747443592103206]: [34.54,36.109,40.715,40.938,35.092,35.817,38.318,24.396,23.492,19.829,22.423,19.289,21.305,27.446,35.078,37.422,36.561,
        1.0 : [distance=29.351509840084493]: [24.89,38.805,37.92,43.102,39.368,39.09,38.847,33.614,23.222,23.302,15.765,22.776,14.168,22.255,24.925,33.572,33.868,43
        1.0 : [distance=36.06269363687599]: [30.743,37.975,43.189,42.739,39.371,44.196,37.651,28.888,18.435,20.915,13.355,11.583,18.05,28.536,31.833,34.721,42.003,4
        1.0 : [distance=45.251292554095734]: [28.549,38.744,41.508,37.852,40.991,40.99,40.683,37.138,30.449,26.896,15.819,16.175,19.811,17.942,31.521,25.981,34.427,
        1.0 : [distance=32.40604653118337]: [33.263,32.562,40.739,40.045,45.552,43.514,32.765,29.375,31.082,17.267,18.66,20.116,26.983,21.294,28.221,38.399,38.37,34
        1.0 : [distance=44.735459683423116]: [30.757,30.447,35.813,42.866,38.28,43.999,37.319,28.093,30.236,28.675,15.743,18.604,18.736,16.542,22.85,32.11,36.231,38
        1.0 : [distance=27.821141991939037]: [29.779,33.596,38.288,39.257,45.479,37.717,31.081,26.769,31.442,23.421,19.662,22.336,15.606,29.348,30.241,36.592,42.33,
        1.0 : [distance=32.794241938714514]: [29.885,29.021,38.638,36.289,47.033,38.241,33.739,25.925,21.447,25.728,15.948,21.828,18.61,19.183,30.8,37.662,31.248,45
        1.0 : [distance=28.736758220324326]: [31.507,30.346,36.611,38.689,45.56,42.892,33.763,32.105,28.334,22.858,16.157,15.348,25.995,26.947,30.396,31.194,41.22,3
        1.0 : [distance=49.04946094326119]: [32.849,32.603,44.122,44.172,38.929,45.408,33.842,30.802,23.223,24.352,14.535,11.557,14.862,16.295,19.733,29.549,32.912,
        1.0 : [distance=28.983076427450193]: [33.25,37.039,33.831,35.839,46.424,41.538,36.962,31.567,23.393,23.5,22.653,14.055,14.182,18.481,23.159,39.309,34.741,36
        1.0 : [distance=46.550391105672084]: [29.618,41.239,44.632,47.893,44.645,41.125,41.4,36.084,26.727,26.132,19.52,9.808,12.894,23.605,24.978,32.81,39.228,47.1
        1.0 : [distance=48.11661717569564]: [30.45,39.127,41.892,47.501,45.269,42.897,40.485,25.862,23.342,19.46,18.456,10.972,24.266,26.528,28.813,33.552,47.228,49
        1.0 : [distance=34.47839974466308]: [32.318,39.972,40.686,36.917,36.527,35.929,36.415,24.746,30.874,25.309,17.48,24.069,25.609,27.306,28.46,30.586,41.451,38
        1.0 : [distance=42.490341517663325]: [25.308,39.109,37.031,40.543,40.658,36.248,35.083,26.776,31.716,24.368,16.079,16.454,15.709,25.993,31.897,34.227,33.423
        1.0 : [distance=35.5728524753414]: [24.185,34.778,34.696,47.83,46.589,46.601,32.881,31.15,23.629,26.012,12.707,17.333,23.969,24.454,30.961,36.841,35.765,43.
        1.0 : [distance=29.862478295553124]: [31.84,39.764,42.345,48.491,39.794,46.056,30.6,30.358,26.517,22.004,20.449,15.816,16.822,26.1,33.21,38.463,39.301,43.27
        1.0 : [distance=35.83313245458476]: [32.844,35.408,40.922,46.212,44.038,41.454,38.891,37.9,30.584,26.355,12.593,17.686,16.318,22.91,25.702,33.097,33.618,36.
        1.0 : [distance=37.043073202972195]: [35.389,31.178,40.041,43.034,49.524,40.942,42.369,30.153,26.253,23.178,19.885,19.693,21.837,26.858,23.533,29.798,43.401
        1.0 : [distance=60.388306248516606]: [27.662,37.199,39.158,44.264,46.473,40.178,38.728,24.412,25.155,15.938,13.125,16.867,13.875,29.969,34.322,40.87,44.225,
        1.0 : [distance=30.494469149404622]: [25.784,34.129,42.659,37.176,35.961,34.307,32.108,29.749,25.047,17.455,24.652,25.311,22.995,30.256,25.955,28.426,34.556
        1.0 : [distance=45.84118166610922]: [25.87,39.195,36.908,47.052,47.384,40.741,42.494,30.282,25.834,17.65,16.004,17.895,13.321,19.045,27.44,31.911,39.208,43.
        1.0 : [distance=52.19421204481973]: [28.075,41.784,42.12,38.735,44.32,34.316,32.212,31.868,24.301,14.547,17.178,22.279,24.357,31.011,31.444,34.837,46.55,48.
        1.0 : [distance=28.743518191398017]: [33.67,38.675,39.742,41.989,37.291,43.975,31.909,25.878,31.08,15.858,13.95,23.097,19.983,21.692,31.579,38.57,33.376,38.
```

- As you can observe in this file, it gives the cluster number (“CL-114”), its radius (r) and distance of each record from the cluster’s centroid. Also it lists each record that is a part of that cluster.

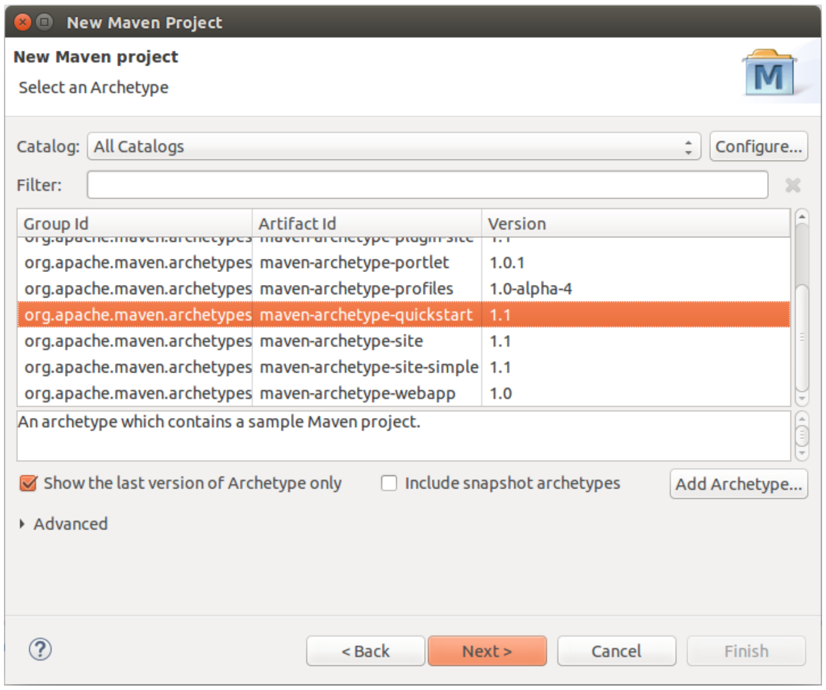
4 Spark

4.1 Creating and managing Maven Projects in Eclipse Mars

4.1.1 Create a Maven Project: File->New->Project->Maven->Maven project


New Maven Project
New Maven project
Select project name and location
Create a simple project (skip archetype selection)
Use default Workspace location
Location:
Browse...
Add project(s) to working set
Working set:
More...
Advanced
< Back
Next >
Cancel
Finish

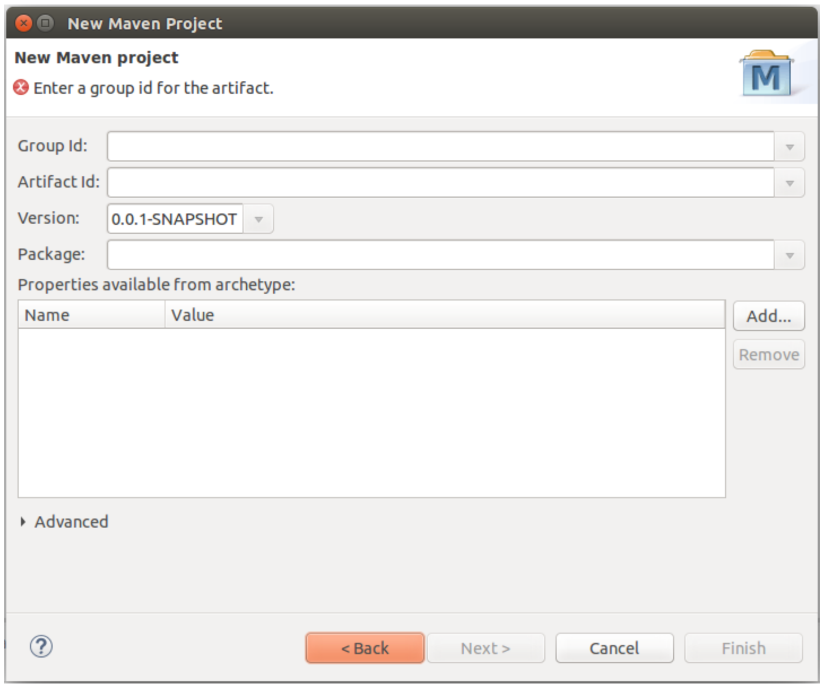
- Use the default maven version and click next.



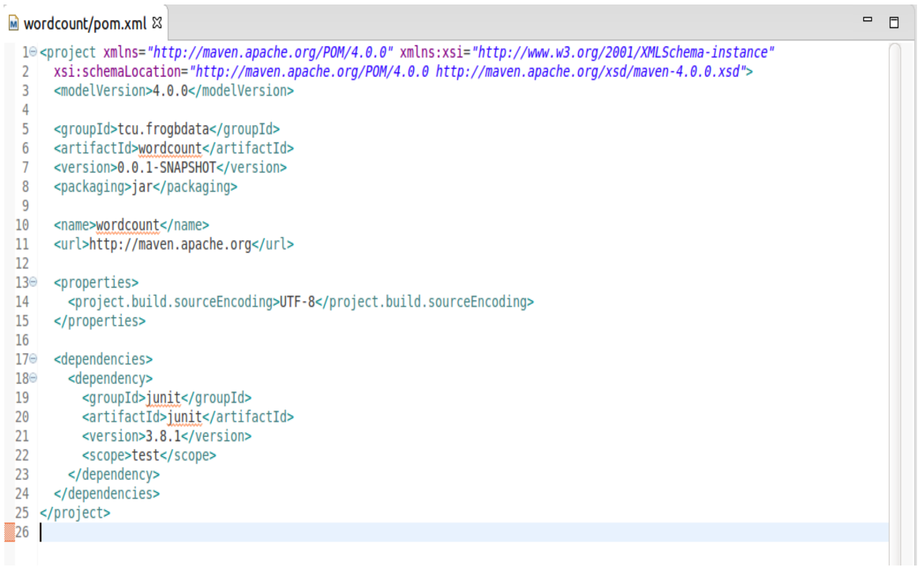
Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1
org.apache.maven.archetypes	maven-archetype-site-simple	1.1
org.apache.maven.archetypes	maven-archetype-webapp	1.0

An archetype which contains a sample Maven project.

Enter the group Id (say tcu.frogbdata) and artifact id (name of the project, say wordcount) and the appropriate package name will be generated.



4.1.2 POM file

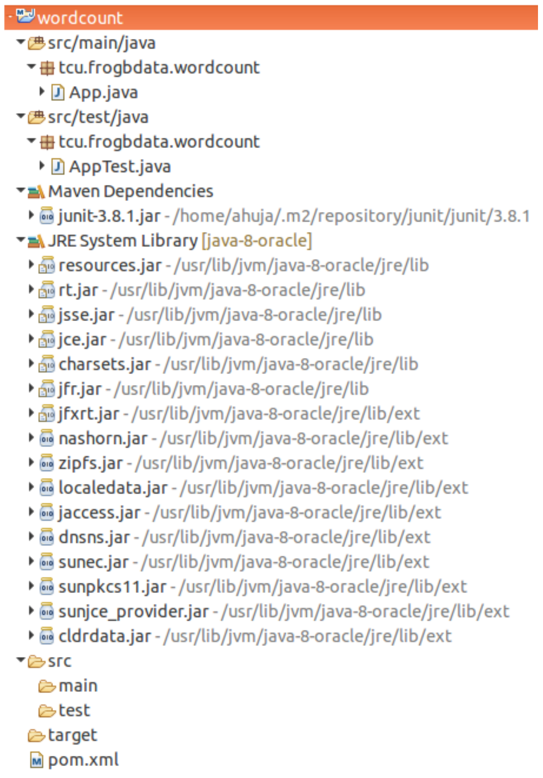


wordcount/pom.xml

```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>tcu.frogbdata</groupId>
  <artifactId>wordcount</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>wordcount</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>

```

4.1.3 Project Structure

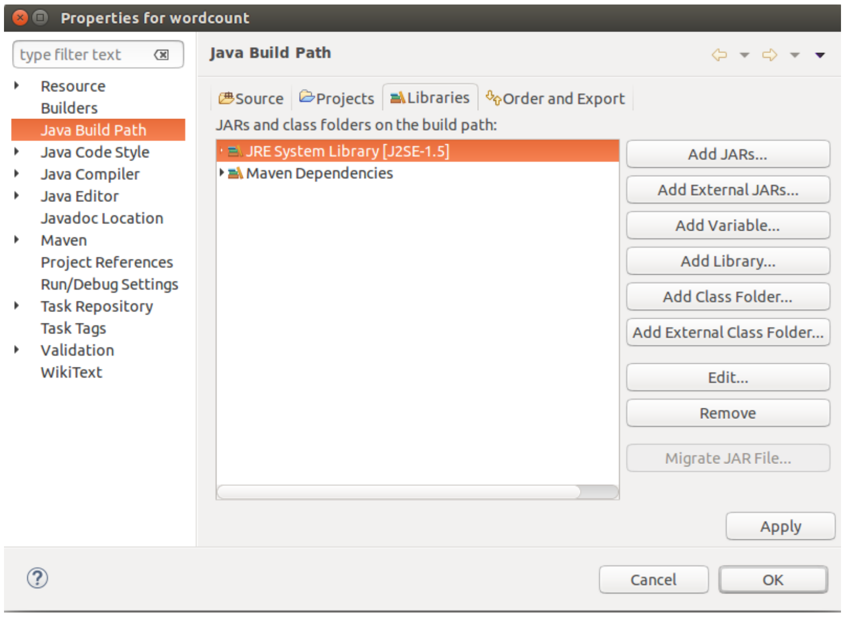
- Directories
    - This is a typical directory structure of a Maven project:



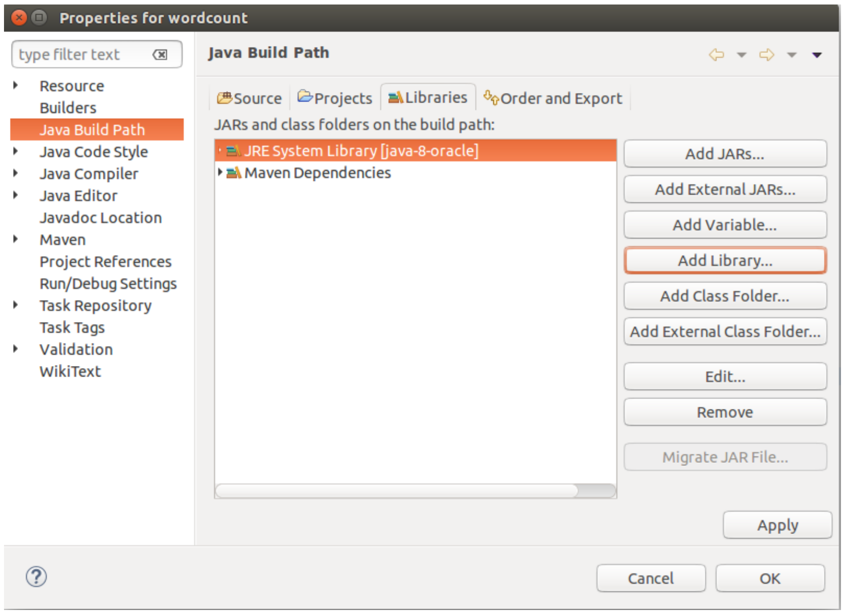
```
wordcount
  src/main/java
    tcu.frogbdata.wordcount
      App.java
  src/test/java
    tcu.frogbdata.wordcount
      AppTest.java
  Maven Dependencies
    junit-3.8.1.jar - /home/ahuja/.m2/repository/junit/junit/3.8.1
  JRE System Library [java-8-oracle]
    resources.jar - /usr/lib/jvm/java-8-oracle/jre/lib
    rt.jar - /usr/lib/jvm/java-8-oracle/jre/lib
    jsse.jar - /usr/lib/jvm/java-8-oracle/jre/lib
    jce.jar - /usr/lib/jvm/java-8-oracle/jre/lib
    charsets.jar - /usr/lib/jvm/java-8-oracle/jre/lib
    jfr.jar - /usr/lib/jvm/java-8-oracle/jre/lib
    jfxrt.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
    nashorn.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
    zipfs.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
    localedata.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
    jaccess.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
    dnsns.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
    sunec.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
    sunpkcs11.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
    sunjce_provider.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
    cldrdata.jar - /usr/lib/jvm/java-8-oracle/jre/lib/ext
  src
    main
    test
  target
  pom.xml
```

- Build Path

Remove the default JRE System Library by right clicking on project, select Build Path->Configure Build Path as shown in the following screenshot. Remove this library by pressing the remove button on the right.



Now, go to Add Library then select JRE System Library from the window prompt, press next and then press Finish.
You should see something similar as depicted in the following screenshot:



4.2 Writing Word Count and Matrix Multiplication programs

4.2.1 Word Count-my first Spark program

- Word Count-Hello World of Spark
- Setup the WC project
    - Create a new Maven project named WordCount
    - Edit the pom.xml file for Maven dependencies
        - Hadoop-client
          http://mvnrepository.com/artifact/org.apache.hadoop/hadoop-client/2.7.1
          http://mvnrepository.com/artifact/org.apache.spark/spark-core_2.10/1.5.2
    - Fix the build path
- Word Count Algorithm
    - The Transformation phase in our word count code filters the input data and gives the value of 1 to each word of the text file.
    - The Action phase then takes the input from the map phase (filtered data) and then counts the number of time each word occurs (sorting by key).
    - All these two functions are performed in Spark main abstraction RDD (Resilient Distributed Datasets)
- Format of input file
    - Refer to our **User's Manual and Research Result** document.
- Format of output file
    - The output file will have each word and its count in each line and will be sorted alphabetically.
- Congratulations on writing your first Spark program

4.2.2 Matrix Multiplication

- Matrix Multiplication
    - Next we build matrix multiplication Java Spark Program to multiply two matrices.
- Setup the MM project
    - Create a new Maven project named MatrixMultiply
    - Edit the pom.xml file for Maven dependencies
        - Hadoop-client
          http://mvnrepository.com/artifact/org.apache.hadoop/hadoop-client/2.7.1
          http://mvnrepository.com/artifact/org.apache.spark/spark-core_2.10/1.5.2
    - Fix the build path
        - Fix the build path as we mentioned earlier in the manual

- Matrix Multiply Algorithm
  - Get both matrices files and transform into a RDD
  - Convert both RDD into Matrix Entry type.
  - Convert both Matrix entries into coordinate matrix RDD and cache them since they are going to be used multiple time.
  - Now convert both coordinate matrices into Block matrix.
  - Then multiply both Block matrices and save the output file.
- Format of input file
  - Refer to User's and Research Results document.
- Format of output file
  - The output will have the final matrix in the form of: row, column, value.
- Congratulations on writing your second Spark program

4.3 Install Spark on Single Node without HDFS

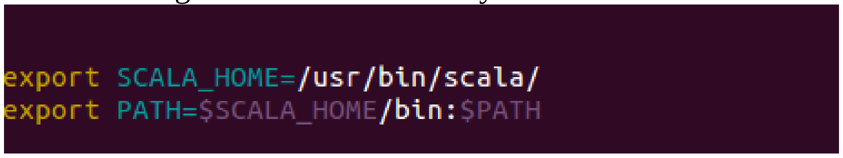
4.3.1 Installation instructions in detail.

- Download the latest version of scala or get the version used in the project from the following link:
  http://downloads.lightbend.com/scala/2.11.8/scala-2.11.8.tgz

- Go to terminal, inside the Downloads directory and enter the following commads:

```
$ tar xvf scala-2.11.8.tgz
$ sudo mv scala-2.11.8 /usr/bin/scala
$ cd
$ vim .bashrc
```

- Add the following lines at the bottom of your bashrc file:



```
export SCALA_HOME=/usr/bin/scala/
export PATH=$SCALA_HOME/bin:$PATH
```

- Restart bashrc

```
$ . .bashrc
```

- Check the scala version to verify successful installation

```
$ scala -version
```

- Install git

```
$ sudo apt-get install git
$ git --version
```

- Download **Spark** from the following link: http://d3kbcqa49mib13.cloudfront.net/spark-1.5.2-bin-hadoop2.6.tgz

- Extract it in the home directory and rename the folder as 'spark' and enter the following commands:

```
$ cd spark
$ sbt/sbt assembly
```

- Verify Spark installation by running a Pi example:

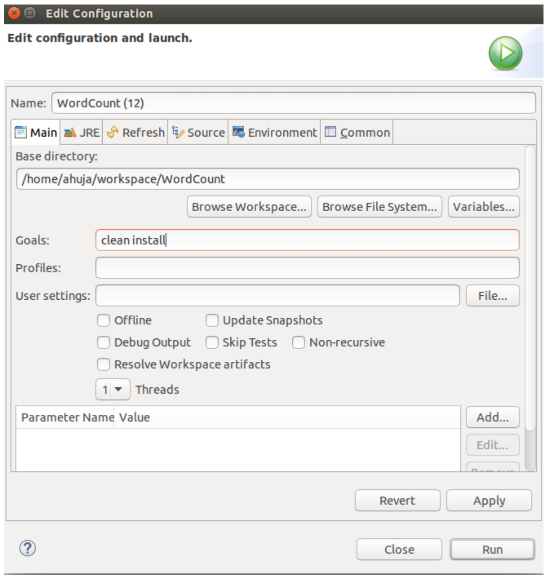
```
$ ./bin/run-example SparkPi 10
```

- You should get Pi's value as 3.140… it means that Spark is successfully installed on your computer.

4.4 Deploy WC and MM on single Node Spark without HDFS

4.4.1 Export JAR file with all dependencies

- Right-click on the project -> Run As -> Maven Build
  - On the pop-up window, write 'clean install' in the Goals text box and click run. Here is a screenshot of what it should it look like.



Edit configuration and launch.

Name: WordCount (12)

Base directory: /home/ahuja/workspace/WordCount

Goals: clean install

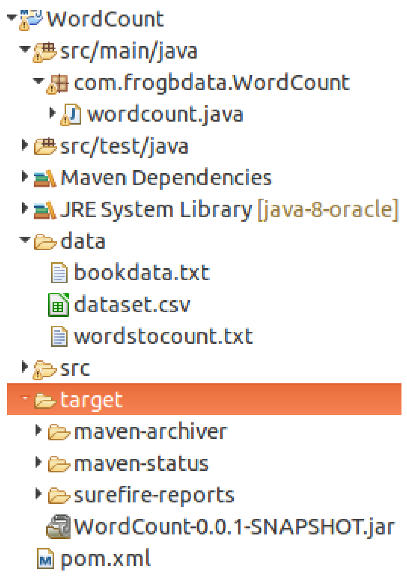
Profiles:

User settings:

☐ Offline ☐ Update Snapshots
☐ Debug Output ☐ Skip Tests ☐ Non-recursive
☐ Resolve Workspace artifacts
1 Threads

Parameter Name Value

- You should see the progress of Maven building the project on the console.
- You can find the newly created jar file of the project inside the 'target' folder of the project.



Note: All these 4 steps have to be followed in case of both: word count and matrix multiplication.

- Run the job through terminal
    - Run the following command:

```
./spark/bin/spark-submit --master local path_to_jar package.class path_to_input path_to_output
```

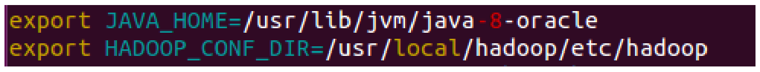
- Access the output from the desired location

4.5 Install Spark on Single Node with HDFS

4.5.1 Hadoop Installation

If Hadoop is not installed on your machine, click here and follow the instructions of Hadoop installation on single node using this link:

```
stop-dfs.sh

stop-yarn.sh

vim /home/username/spark/conf/spark-env.sh
```



```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
```

```
start-dfs.sh

start-yarn.sh
```

4.6 Deploy WC and MM on single Node Spark with HDFS

4.6.1 Export JAR file with all dependencies

- Refer to the above instructions (4.4.1)

4.6.2 Put the input file on HDFS

- Refer to 3.4.2 for instructions

4.6.3 Run the job through terminal

- Run the following command:

```
$ ./spark/bin/spark-submit --master yarn-cluster --num-executors 5 --executor-cores 1 --executor-memory 3G path_to_jar package.class path_to_input path_to_output
```

4.6.4 Access the output from HDFS and interpret results

- Refer to 3.4.4 for instructions

4.7 Setup a YARN Cluster

4.7.1 Why cluster?

- Hadoop was never built to run on a single node. There is no point of doing that except when you are testing your Spark code.
- Spark becomes powerful and runs at its best in a cluster of nodes.
- Parallel processing is only possible in a cluster where the data can be distributed among different nodes (datanodes) which are controlled by a manager node (namenode).

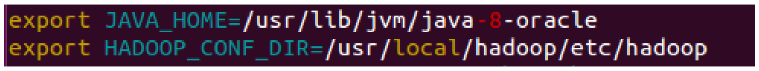
4.7.2 Structure of our cluster

- Our Cluster had 3 nodes, each with a configuration as follows:
  - 8 GB RAM
  - 500 GB HDD
  - Ubuntu 15.04
- Manager-Worker Architecture
  - 1 Manager
  - 2 Workers

4.7.3 Install Hadoop on 2 separate machines (workers) Using Hadoop instructions

- Click here for instructions if you have not set-up a cluster.
- Now change the appropriate configuration files

```
stop-dfs.sh
stop-yarn.sh
vim /home/username/spark/conf/spark-env.sh
```



```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
```

start-dfs.sh

start-yarn.sh

4.8 Deploy WC and MM on Cluster

4.8.1 Store the input files on HDFS

- Refer to 3.6.1

4.9 Development in Python

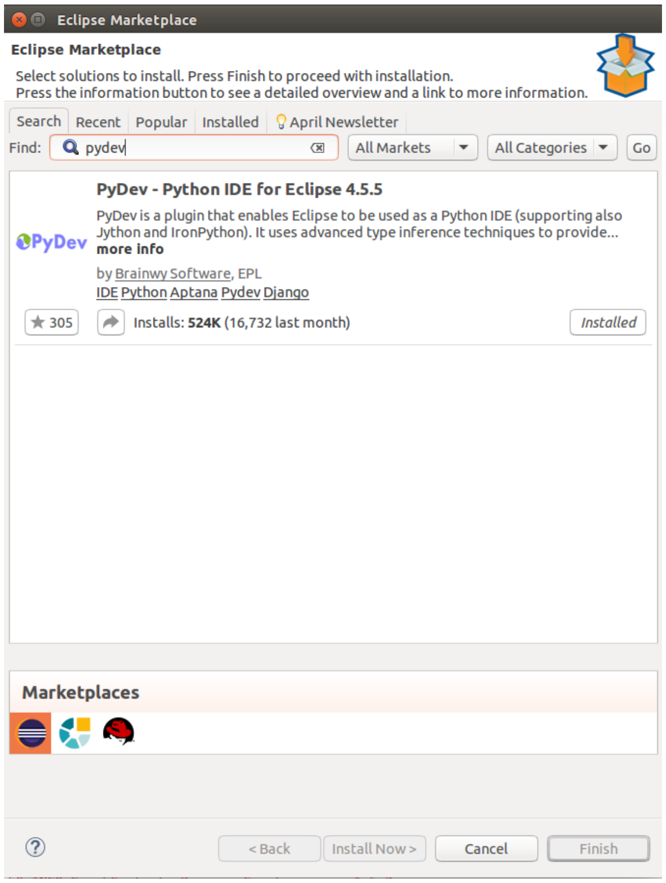
4.9.1 Install Python in all nodes

- Python should come pre-installed in Ubuntu 15.04 and above.

4.9.2 Install and configure PyDev plugin in Eclipse

- Go to Help->Eclipse Marketplace

  Search "PyDev" in the Find textbox
  Select PyDev and install it with all default settings and restart eclipse.


Eclipse Marketplace

Eclipse Marketplace

Select solutions to install. Press Finish to proceed with installation.
Press the information button to see a detailed overview and a link to more information.

Find: pydev

PyDev - Python IDE for Eclipse 4.5.5

PyDev is a plugin that enables Eclipse to be used as a Python IDE (supporting also Jython and IronPython). It uses advanced type inference techniques to provide... **more info**

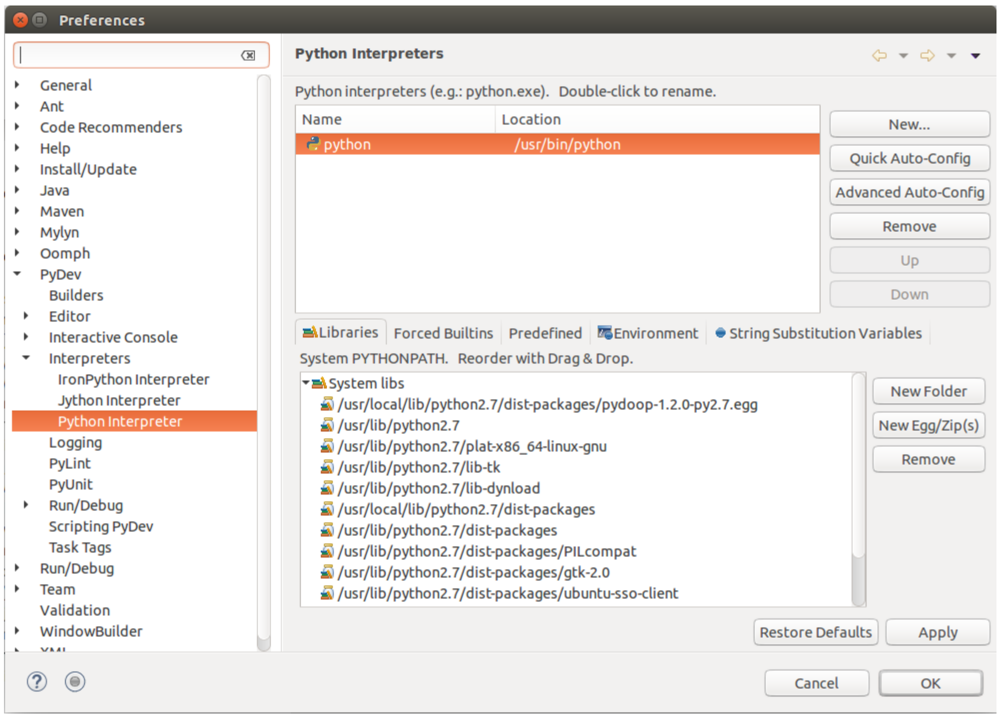
by Brainwy Software, EPL
IDE Python Aptana Pydev Django

305 Installs: **524K** (16,732 last month) *Installed*

- Python Interpreters

In Eclipse, go to Window->Preferences
In the left pane, select PyDev->Interpreters->Python Interpreter

Click Quick Auto-Config to include the python path to the interpreter.



Python Interpreters

Python interpreters (e.g.: python.exe). Double-click to rename.

Name	Location
python	/usr/bin/python

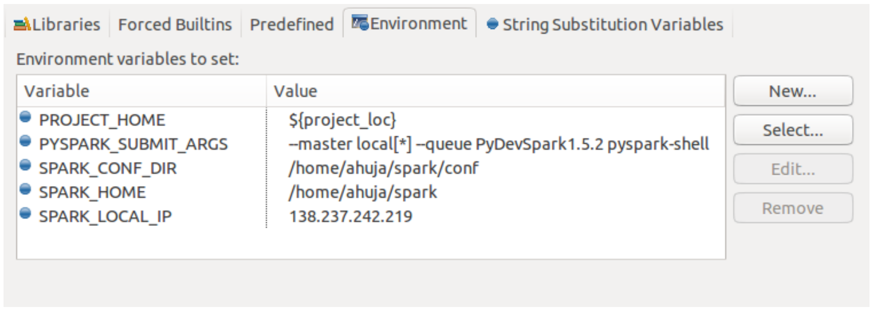
System PYTHONPATH. Reorder with Drag & Drop.

```
System libs
  /usr/local/lib/python2.7/dist-packages/pydoop-1.2.0-py2.7.egg
  /usr/lib/python2.7
  /usr/lib/python2.7/plat-x86_64-linux-gnu
  /usr/lib/python2.7/lib-tk
  /usr/lib/python2.7/lib-dynload
  /usr/local/lib/python2.7/dist-packages
  /usr/lib/python2.7/dist-packages
  /usr/lib/python2.7/dist-packages/PILcompat
  /usr/lib/python2.7/dist-packages/gtk-2.0
  /usr/lib/python2.7/dist-packages/ubuntu-sso-client
```

Run the following commands in order to install **NumPy** (Numerical Python)

```
sudo apt-get update
sudo apt-get install python-numpy
```

- In the **Libraries** tab, add the following folders:
  - /home/username/spark/python
  - /usr/lib/python2.7/dist-packages
  - /usr/local/bin
- Add the following Zips
  - /home/username/spark/python/lib/py4j-0.8.2.1-src.zip
- In the **Environment** tab, add the following Variables:
  - Variable Name: PROJECT_HOME, value: *${project_loc}*
  - Variable Name: PYSPARK_SUBMIT_ARGS, value: --*master local[*] –queue PyDevSpark1.5.1 pyspark-shell*
  - Variable Name: SPARK_CONF_DIR, value: */home/username/spark/conf*
  - Variable Name: SPARK_HOME, value: */home/username/spark*
  - Variable Name: SPARK_LOCAL_IP, value: *IP address of the machine*



Environment variables to set:

Variable	Value
PROJECT_HOME	${project_loc}
PYSPARK_SUBMIT_ARGS	--master local[*] --queue PyDevSpark1.5.2 pyspark-shell
SPARK_CONF_DIR	/home/ahuja/spark/conf
SPARK_HOME	/home/ahuja/spark
SPARK_LOCAL_IP	138.237.242.219

Restart Eclipse for all the changes to take an effect.

- Fix dependencies
  - Installation of pip
    - Download pip from the following link
      https://bootstrap.pypa.io/get-pip.py
    - Go into the downloaded directory and type the following command to install pip on your computer.
      ```
      sudo python get-pip.py
      ```
    - This will also install python setup tools and wheel if not already installed. Verify the installation of pip by entering the following command:
      ```
      whereis pip
      ```
    - If you get the following result, then the installation was successful.
      Pip: /usr/local/bin/pip /usr/local/bin/pip2.7
  - Installation of pydoop using pip
    - Enter the following command:
      ```
      sudo pip install pydoop
      ```
    - Verify the installation of pydoop by entering the following command:
      ```
      whereis pydoop
      ```
    - If the result is following, then the installation was not successful.
      Pydoop:
    - If this is the case, you will need to install pydoop manually.
  - Installation of pydoop manually
    - Download the tar file of pydoop from the following link:
      https://pypi.python.org/packages/75/75/085a6410b085f231328884ca3349287a8705822ad8afdca715401e5c4f33/pydoop-1.2.0.tar.gz#md5=e6b1dff3cf19cd7815b7134e67f683c4

- Extract the file and go inside the extracted directory. Now enter the following commands:
  ```
  vim pydoop/Hadoop_utils.py
  ```
- Go to line 410 and replace this line
  ```
  self.__hadoop_home = None by
  self.__hadoop_home = ‘/usr/local/hadoop’
  vim pydoop/utils/jvm.py
  ```
- Go to line 27 and replace the line
  ```
  return os.environment[“JAVA_HOME”] by
  return ‘/usr/lib/jvm/java-8-oracle’
  ```
- Now enter the following command:
  ```
  sudo python setup.py install
  ```
- Verify the installation of pydoop by entering the following command:
  ```
  whereis pydoop
  ```
- You should get the following result:pydoop:
  ```
   pydoop: /usr/local/bin/pydoop
  ```

4.10 Recommender

4.10.1 Collaborative Filtering

- In Spark, we used collaborative filtering algorithm using Apache MLlib library. According to Apache Spark documentation on MLlib collaborative filtering, “Collaborative filtering is commonly used for recommender systems. These techniques aim to fill in the missing entries of a user-item association matrix. ***spark.mllib*** currently supports model-based collaborative filtering, in which users and products are described by a small set of latent factors that can be used to predict missing entries. spark.mllib uses the **alternating least squares (ALS)** algorithm to learn these latent factors.”
- To learn more about ALS algorithm, refer to this link.

4.10.2 Source Code

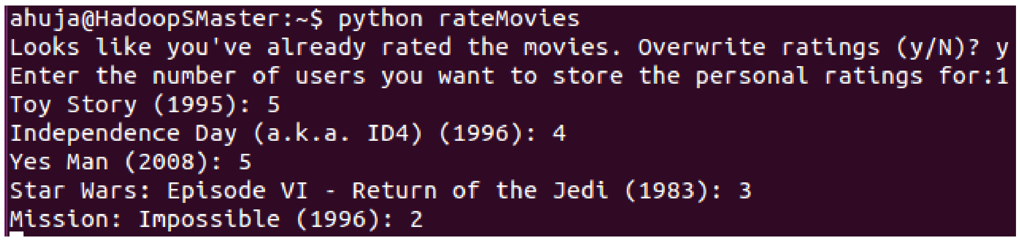
- To get the source code of our recommender, follow this link:
  http://brazos.cs.tcu.edu/1516frogbdata/deliverables.html
  Name of the file: MovieLensALSMain.py

4.10.3 Data Files for Spark Recommender

- To get the data files, follow this link:http://grouplens.org/datasets/movielens/
- Here, you will find different sizes of the real movie ratings (100K, 1M, 10M, 20M and 22M).
- For more details please refer to our User’s Manual and Research Results document.

4.10.4 Instructions

- To give the personalized ratings for any number of users, run the python script named “rateMovies” and follow the instructions: Keep in mind that the movies.dat file (movies.dat file should be the same file for which you want to run the recommender) should be in the same directory as the python script. This will generate the file called “personalRatings.txt” that contains the personal ratings of the user.
- Sample run of the python script is as follows:



```
ahuja@HadoopSMaster:~$ python rateMovies
Looks like you've already rated the movies. Overwrite ratings (y/N)? y
Enter the number of users you want to store the personal ratings for:1
Toy Story (1995): 5
Independence Day (a.k.a. ID4) (1996): 4
Yes Man (2008): 5
Star Wars: Episode VI - Return of the Jedi (1983): 3
Mission: Impossible (1996): 2
```

- **Note: To start the recommender on a cluster of 2 nodes, run the following commands:**

  start-dfs.sh

  start-yarn.sh

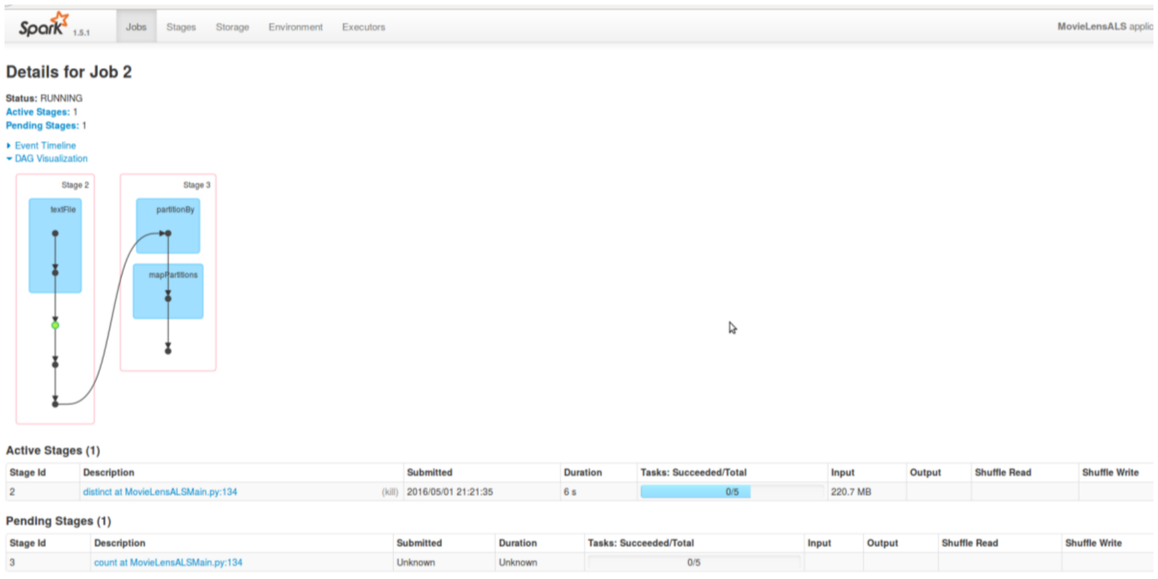
  hadoop fs -mkdir -p /user/username1/spark_recommender/

  hadoop fs -put /home/username/Downloads/ml-latest/ /user/username1/spark_recommender/ml-latest/

  hadoop fs -put /home/username/Desktop/personalRatings.txt/ /user/username1/spark_recommender/personalRatings.txt

  ./spark/bin/spark-submit --master yarn-cluster --num-executors 5 --executor-cores 1 --executor-memory 3G /home/username/workspace/MovieLensALS/src/MovieLensALSMain.py hdfs://HadoopSMaster:9000/user/username1/spark_recommender/ml-latest/ hdfs://HadoopSMaster:9000/user/username1/spark_recommender/personalRatings.txt hdfs://HadoopSMaster:9000/user/username1/spark_recommender/output22M

- Once the job is running, you can see the progress on the web with all the executors and the job process:



Details for Job 2

Status: RUNNING
Active Stages: 1
Pending Stages: 1

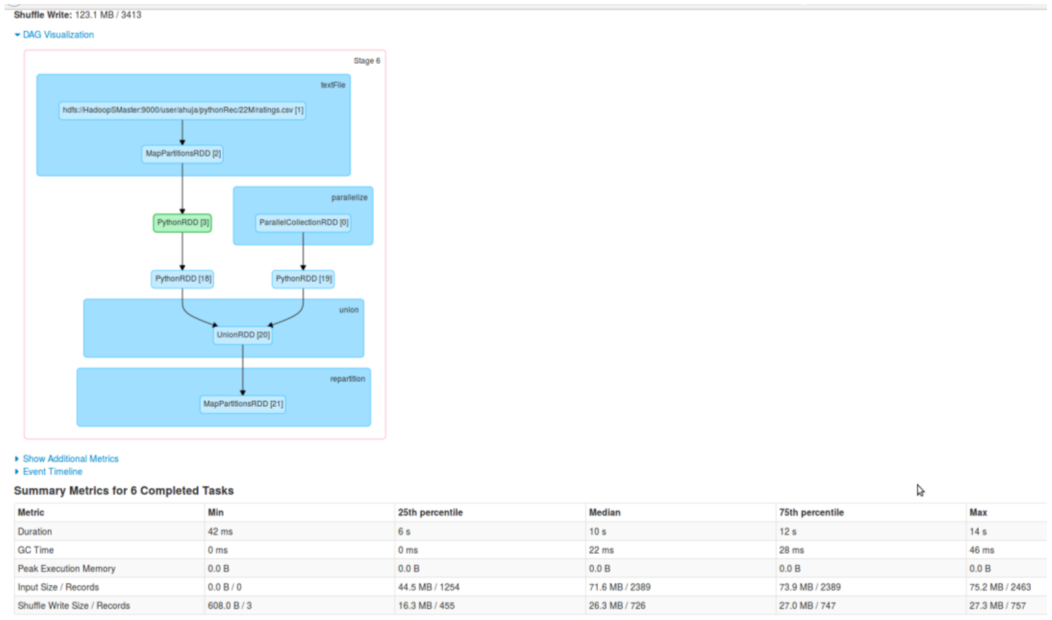
- Event Timeline
- DAG Visualization

Active Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
2	distinct at MovieLensALSMain.py:134 (kill)	2016/05/01 21:21:35	6 s	0/5	220.7 MB			

Pending Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
3	count at MovieLensALSMain.py:134	Unknown	Unknown	0/5				



Shuffle Write: 123.1 MB / 3413

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 6 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	42 ms	6 s	10 s	12 s	14 s
GC Time	0 ms	0 ms	22 ms	28 ms	46 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Input Size / Records	0.0 B / 0	44.5 MB / 1254	71.6 MB / 2389	73.9 MB / 2389	75.2 MB / 2463
Shuffle Write Size / Records	608.0 B / 3	16.3 MB / 455	26.3 MB / 726	27.0 MB / 747	27.3 MB / 757

4.11 K-Means Clustering

4.11.1 What is clustering?

- As the name suggests, Clustering implies grouping items together. But on the basis of what?
- We use clustering in Big Data industry to group similar items/users together for better data management.
- It has numerous applications in the business and helps them to target their customers in a better way.
- K-Means algorithm is a simple, but widely used algorithm used for clustering. All objects need to be represented as a set of numerical features. Also, the user has to specify the number of groups (k) he wants.

4.11.2 Application of clustering

- Have you ever been creeped out of the advertisements on applications like Facebook, YouTube or any other website you visit? If yes, it is because of clustering. If not, then look carefully!
- The advertisements have become personalized to a great extent. If you searching pattern on google shows that you read a lot about Big Data, notice how you are flooded with Big Data company advertisements and jobs. It happened with us!
- There is no magic that goes behind this. The clustering algorithms help group similar kind of users together.
- This is how a bank decides to give you a loan or not. Based on your social security, past credit and other financial details they put you in an appropriate group.

4.11.3 Algorithm

- The first step is to choose the number of clusters (k). For example, the user chooses to have 5 clusters.
- In the next step, we choose 5 random records as the centroids of these 5 clusters.
- Now we have 5 cluster centroids but they are randomly chosen.
- Next, we assign each record/point to a specific cluster based on the distance of that point with the centroid of that cluster.
- Next, we compute the distances from each point and allot points to the cluster where the distance from the centroid is minimum.
- Once we are done with re-allotting the points to their new and final clusters, we create separate files for each cluster and save them into HDFS.

5 Glossary of Terms

Apache Hadoop: Apache Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets.

Apache Hadoop Yarn: YARN (Yet Another Resource Negotiator) is a cluster management technology. It is characterized as a large-scale, distributed operating system for Big Data applications

Apache Mahout: An Apache software used to produce free implementations of distributed scalable machine learning algorithms that help in clustering and classification of data.

Apache Maven: A build automation tool for projects that uses XML to describe the project the project that is being built and its dependencies on other external modules.

Apache Spark: Apache Spark is an open source cluster computing framework which allows user programs to load data into a cluster's memory and query it repeatedly.

Big Data: Extremely large data sets that may be analyzed computationally to reveal patterns, trends, and associations, especially relating to human behavior and interactions.

Collaborative Filtering: Method to predict the interests of a user based on interests of other users.

Co-occurrence algorithm: Counting the number of times each pair of items occur together and then predicting the interests of a user based on the user’s previous interests and most co-occurred items.

HDFS: Hadoop Distributed File System is a Java based file system that provides scalable and reliable data storage.

IDE: Integrated Development Environment.

K-means clustering: A way of vector quantization used for cluster analysis in data mining.

Map Reduce: A programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster.

MLlib: Apache Spark’s scalable machine learning library that consists of common learning algorithms and utilities including classification, clustering, filtering etc.

PyDev: A Python IDE for Eclipse which is used in Python Development.

Root Access: Access to install various software and related items on Linux machines.

Scala: A programming language for general software applications.

XML: XML stands for Extensible Markup Language that defines the protocol for encoding documents in a format that is both, human and machine-readable.