Lab 2

Gangmin Li

**Big Data Analysis with Hadoop**

Understand big data analysis platform (Hadoop) and parallel programming model (MapReduce) with Lab tasks

**Lab Practice Two:**

**Big Data Analysis with Hadoop**

Gangmin Li

Lab Report Due: 20th November, 2020 17:00pm

Major reference website:

https://www.cloudera.com/products/open-source/apache-hadoop.html

I Objectives

**II Hadoop/MapReduce Lab tutorial**

[1. Introduction 4](#_Toc19981241)

[Hadoop Architecture 4](#_Toc19981242)

[How Does Hadoop Work? 5](#_Toc19981245)

[Advantages of Hadoop 5](#_Toc19981246)

[2. Set up Hadoop Running Environment 6](#_Toc19981247)

[Pre-installation 6](#_Toc19981248)

[Downloading Hadoop 8](#_Toc19981252)

[Hadoop Operation Modes 8](#_Toc19981253)

[Installing Hadoop in Standalone Mode 9](#_Toc19981254)

[Installing Hadoop in Pseudo Distributed Mode 11](#_Toc19981256)

[3. Hadoop - HDFS 16](#_Toc19981261)

[Features of HDFS 16](#_Toc19981262)

[HDFS Architecture 16](#_Toc19981263)

[Goals of HDFS 18](#_Toc19981267)

[HDFS Operations 18](#_Toc19981268)

[4. Hadoop - MapReduce 19](#_Toc19981274)

[What is MapReduce? 20](#_Toc19981275)

[The Algorithm 20](#_Toc19981276)

5. [Compilation and Execution of Process Units Program 24](#_Toc19981282)

6. [How to Interact with MapReduce Jobs 27](#_Toc19981294)

**III Example Project**

[1. Develop, debug and test wordCount.jar 32](#_Toc19981299)

[2. Run WordCount on Hadoop 43](#_Toc19981309)

[3. Explore Job running on Hadoop by Hadoop admin 45](#_Toc19981310)

**IV Your lab tasks**

[1. Project 49](#_Toc19981311)

[2. Tasks 49](#_Toc19981312)

[3. Submission 49](#_Toc19981313)

# I Objectives

This Lab practice aims to provide you a real Big Data Analysis platform experience.

We know that data analysis can be carried out in many platforms such as PC, work station and some clouds using different tools and software. Once facing real BIG data those platform may not be sufficient. Once restriction is the storage capacity of those conventional computer, the other is the computation power provided by the conventional computer which is not power enough to process large amount of the data in required time scale. If the data is stored else where the network band width will also be a problem if transfer large amount of data to a local machine. The organizations whose business is Data Analysis or processing mostly will have distributed systems to store data to take advantages of scalability and extendibility. They also either use HPC including middle-range or super computers or in most case use computer clusters. A computer cluster is network connected large number of conventional computers. In any case, most popular software deployed today is Hadoop. To achieve higher through-put on Hadoop the computation job has to be programmed in a specific fashion that can run in parallel. This program model is MapReduce.

So this lab practice is provide you chance to run a computation job which is programmed following MapReduce model on Hadoop environment, which is mostly on a computer cluster. However, Hadoop can be deployed on a single computer too. For learning purpose, we can use a single computer to simulate the situation where in reality they mostly will be on a computer cluster. The procedure and the needed knowledge are the same.

Overall, after you accomplished the lab practices you should have:

1. Familiar with Hadoop platform. Particularly understand distributed file or data storage in clouding data access and control
2. Be competent to write MapReduce program to do your data analysis to take advantage of computation efficiency
3. Knowing the job can be running on computer cluster and virtual machine

Please note that this is NOT a language practice. So you are assumed to possess the following knowledge：

1. Java programming language, running environment （JRE） and a Java development Kit (JDK) such as Eclipse.
2. Hadoop is mostly deployed on Cluster and its operating system (OS) is mostly Linux based like Ubuntu, Redhead, SuSE Linux, CentOS etc. Therefore it is assumed you should be competent to sue Linux commands.
3. If you need configure or install some extensions of Hadoop, you are also assumed to be competent with Linux Package management like Marven etc.
4. Lastly, since we are not suing Cluster, we will use Virtual Machine (VM), you are aslo assumed to have some knowledge of VMplayer

Don’t be scared away from practice. The massage is you should have all this knowledge but does not mean cannot learn them if you done know. The point is we don’t want you to be sidetracked. This lab practice and supplied tutorial will be sole focused on Hadoop and MapReduce.

This handbook covers three sections:

* Section one: Hadoop and MapReduce tutorials
* Section two: Sample project
* Section Three: Your lab tasks

**Section 1 Describes the virtual machine environment.** This tutorial will use a VM provided by **Cloudera,** (you are encouraged to explore the latest version on Cloudera website), which include everything you need to start Hadoop. Instead of using virtual machine, you are also encouraged to setup your own pseudo-distributed or fully distributed cluster and deploy any version of Hadoop that is at least 1.0 will suffice. (For an easy way to set up a cluster, you can try Cloudera Manager: <http://archive.cloudera.com/cm4/installer/latest/cloudera-manager-installer.bin>) or follows manual provided or follow instructions from **Apache Hadoop web site**. Notice that Apache Hadoop is different with Cloudera Hadoop. The core components are the same.

If you choose to setup your own cluster, you are responsible for making sure the cluster is working properly. The TAs may be unable to help you debug configuration issues in your own cluster.

**Section 2** **Produce your data analysis program**. One you have a Hadoop running environment. Our tutorial uses **Cloudera VM** to simulate a cluster. In real life, Hadoop is deployed on enterprises’ cluster. You not have access to for developing and debug purpose. Therefore you may need your won development environment such as **Eclipse**. This tutorial gives an example using Eclipse environment in the virtual machine (VM) to do the development. The tutorial provides an end-to-end example which makes you familiar with the Big Data analytics procedure and framework from creating a project, adding code, and building, running, and debugging it. And finally to produce a running program (jar) ready for deployment or run on a Hadoop platform (cluster).

**Section 3 Deploy your program or run your jobs on Hadoop platform.** The last phase of the data analysis on Hadoop platform is to actually deploy your job on the platform and run jobs. Once the job has been accomplished successfully you will have results.

Good luck.

# II Hadoop/MapReduce Lab Tutorial

## Hadoop Basics

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. The Hadoop framework application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

### Hadoop Architecture

At its core, Hadoop has two major layers:

* Processing/Computation layer (MapReduce), and
* Storage layer (Hadoop Distributed File System).



### MapReduce

MapReduce is a parallel programming model for writing distributed applications devised at Google for efficient processing of large amounts of data (multi-terabyte data-sets), on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. The MapReduce program runs on Hadoop which is an Apache open-source framework.

### Hadoop Distributed File System

The Hadoop Distributed File System (**HDFS**) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. It is highly fault-tolerant and is designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications having large datasets.

Apart from the above-mentioned two core components, Hadoop framework also includes the following two modules:

* **Hadoop Common** − Java libraries and utilities required by other Hadoop modules.
* **Hadoop YARN** − A framework for job scheduling and cluster resource management.

### How Does Hadoop Work?

It is quite expensive to build bigger servers with heavy configurations that handle large scale processing, but as an alternative, you can tie together many commodity computers, as a single functional distributed system and practically, the clustered machines can read the dataset in parallel and provide a much higher throughput. Moreover, it is cheaper than one high-end server. So this is the first motivational factor behind using Hadoop that it runs across clustered and low-cost machines.

Hadoop runs code across a cluster of computers. This process includes the following core tasks that Hadoop performs:

* Data is initially divided into directories and files. Files are divided into uniform sized blocks of 128M and 64M (preferably 128M).
* These files are then distributed across various cluster nodes for further processing.
* HDFS, being on top of the local file system, supervises the processing.
* Blocks are replicated for handling hardware failure.
* Checking that the code was executed successfully.
* Performing the sort that takes place between the map and reduce stages.
* Sending the sorted data to a certain computer.
* Writing the debugging logs for each job.

### Advantages of Hadoop

Hadoop framework allows the user to quickly write and test distributed systems. It is efficient, and it automatic distributes the data and work across the machines and in turn, utilizes the underlying parallelism of the CPU cores.

* Hadoop does not rely on hardware to provide fault-tolerance and high availability (FTHA), rather Hadoop library itself has been designed to detect and handle failures at the application layer.
* Servers can be added or removed from the cluster dynamically and Hadoop continues to operate without interruption.
* Another big advantage of Hadoop is that apart from being open source, it is compatible on all the platforms since it is Java based.

Hadoop is supported by GNU/Linux platform and its flavors. Therefore, we have to install a Linux operating system for setting up Hadoop environment. In other OS systems, you can install a Virtualbox (VM) software in it and have Linux inside the Virtualbox. For example, on Widows machine, you can install a VM and then install Linux (see appendix I).

## Set up Hadoop Running Environment

Once you have a Linux installed in a virtual machine you need set up a few environment variables. These are required by the Hadoop platform.

### Pre-installation

To setup Hadoop you need:

1. A Linux OS
2. A JRE (Java Running Environment) and a JDK (or text editor to write Java code)

**If you don’t have a Linux environment, Please follow instruction to download and build a Linux virtual box in appendix I.**

**We assume you have basic knowledge of Linux. Particularly ssh (Secure Shell). Some basic Linux command are provided in appendix II.**

**If you insisted on running Hadoop on windows, this blog may be helpful see appendix IV.**

**The following tutorial is help you to build a Hadoop deployment from scratch by using Apache Hadoop.**

Here are the steps and commands you need to use to set up Hadoop under a Linux environment.

### Creating a User

At the beginning, it is recommended to create a separate user for Hadoop to isolate Hadoop file system from Unix file system. Follow the steps given below to create a user.

* Open the root using the command “su”.
* Create a user from the root account using the command “useradd username”.
* Now you can open an existing user account using the command “su username”.

Open the Linux terminal and type the following commands to create a user.

$ su

password:

# useradd hadoop

# passwd hadoop

New passwd:

Retype new passwd

### SSH Setup and Key Generation

SSH setup is required to do different operations on a cluster such as starting, stopping, distributed daemon shell operations. To authenticate different users of Hadoop, it is required to provide public/private key pair for a Hadoop user and share it with different users.

The following commands are used for generating a key value pair using SSH. (If you have no knowledge of PKI and Java Keygen, please search and read related documents). Copy the public keys from id\_rsa.pub to authorized\_keys, and provide the owner with read and write permissions to authorized\_keys file respectively.

$ ssh-keygen -t rsa

$ cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys

$ chmod 0600 ~/.ssh/authorized\_keys

### Check and Install Java

Java is the main prerequisite for Hadoop. First of all, you should verify the existence of java in your system using the command “java -version”. The syntax of java version command is given below.

$ java -version

If everything is in order, it will give you the following output.

java version "1.7.0\_71"

Java(TM) SE Runtime Environment (build 1.7.0\_71-b13)

Java HotSpot(TM) Client VM (build 25.0-b02, mixed mode)

If java is not installed in your system, then follow the steps given below for installing java.

#### Step 1

Download java (JDK <latest version> - X64.tar.gz) by visiting the following link [www.oracle.com](https://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html)

Updated jdk download: (<https://www.oracle.com/java/technologies/javase-jdk11-downloads.html>)

Then **jdk-xxx-linux-x64.tar.gz** will be downloaded into your system.

#### Step 2

Generally you will find the downloaded java file in Downloads folder. Verify it and extract the **jdk-xxx-linux-x64.gz** file using the following commands.

$ cd Downloads/

$ ls

jdk-xxx-linux-x64.gz

$ tar zxf jdk-xxx-linux-x64.gz

$ ls

jdk1.7.0\_71 jdk-SuSI-linux-x64.gz

#### Step 3

To make java available to all the users, you have to move it to the location “/usr/local/”. Open root, and type the following commands.

$ su

password:

# mv jdk1.7.0\_71 /usr/local/

# exit

#### Step 4

For setting up **PATH** and **JAVA\_HOME** variables, add the following commands to **~/.bashrc** file.

export JAVA\_HOME=/usr/local/jdk1.7.0\_71

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

Now apply all the changes into the current running system.

$ source ~/.bashrc

#### Step 5

Use the following commands to configure java alternatives:

# alternatives --install /usr/bin/java java usr/local/java/bin/java 2

# alternatives --install /usr/bin/javac javac usr/local/java/bin/javac 2

# alternatives --install /usr/bin/jar jar usr/local/java/bin/jar 2

# alternatives --set java usr/local/java/bin/java

# alternatives --set javac usr/local/java/bin/javac

# alternatives --set jar usr/local/java/bin/jar

Now verify the java -version command from the terminal as explained above.

Once your Java is installed and running correctly. You can download and install Hadoop.

### Downloading Hadoop

Download and extract Hadoop latest version (3.2.0) from **Apache software foundation**

Source: <https://hadoop.apache.org/releases.html>, or <http://apache.claz.org/hadoop/common/>

using the following commands. (check the URL and use the latest verion number)

$ su

password:

# cd /usr/local

# wget http://apache.claz.org/hadoop/common/hadoop-3.2.1/

hadoop-3.2.0.tar.gz

# tar xzf hadoop-3.2.1.tar.gz

# mv hadoop-3.2.0/\* to hadoop/

# exit

### Hadoop Operation Modes

Once you have downloaded Hadoop, you can operate your Hadoop cluster in one of the three supported modes −

**Local/Standalone Mode** − After downloading Hadoop in your system, by default, it is configured in a standalone mode and can be run as a single java process.

**Pseudo Distributed Mode** − It is a distributed simulation on single machine. Each Hadoop daemon such as hdfs, yarn, MapReduce etc., will run as a separate java process. This mode is useful for development.

**Fully Distributed Mode** − This mode is fully distributed with minimum two or more machines as a cluster. We will come across this mode in detail in the coming chapters.

### Installing Hadoop in Standalone Mode

Here we will discuss the installation of **Hadoop 3.2.1** in standalone mode.

There are no daemons running and everything runs in a single JVM. Standalone mode is suitable for running MapReduce programs during development, since it is easy to test and debug them.

#### Setting Up Hadoop

You can set Hadoop environment variables by appending the following commands to **~/.bashrc** file.

export HADOOP\_HOME=/usr/local/hadoop

Before proceeding further, you need to make sure that Hadoop is working fine. Just issue the following command −

$ hadoop version

If everything is fine with your setup, then you should see the following result:

Hadoop 3.2.0

Subversion https://svn.apache.org/repos/asf/hadoop/common -r 1529768

Compiled by hortonmu on 2013-10-07T06:28Z

Compiled with protoc 2.5.0

From source with checksum 79e53ce7994d1628b240f09af91e1af4

It means your Hadoop's standalone mode setup is working fine. By default, Hadoop is configured to run in a non-distributed mode on a single machine.

Example

Let's check a **simple example of Hadoop**. Hadoop installation delivers the following example MapReduce jar file, which provides basic functionality of MapReduce and can be used for calculating, like Pi value, word counts in a given list of files, etc.

$HADOOP\_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar

Let's have an input directory where we will push a few files and our requirement is to count the total number of words in those files. To calculate the total number of words, we do not need to write our MapReduce, provided the **.jar** file contains the implementation for word count. You can try other examples using the same **.jar** file; just issue the following commands to check supported MapReduce functional programs by hadoop-mapreduce-examples-2.2.0.jar file.

$ hadoop jar $HADOOP\_HOME/share/hadoop/mapreduce/hadoop-mapreduceexamples-2.2.0.jar

#### Step 1

Create temporary content files in the input directory. You can create this input directory anywhere you would like to work.

$ mkdir input

$ cp $HADOOP\_HOME/\*.txt input

$ ls -l input

It will give the following files in your input directory −

total 24

-rw-r--r-- 1 root root 15164 Feb 21 10:14 LICENSE.txt

-rw-r--r-- 1 root root 101 Feb 21 10:14 NOTICE.txt

-rw-r--r-- 1 root root 1366 Feb 21 10:14 README.txt

These files have been copied from the Hadoop installation home directory. For your experiment, you can have different and large sets of files.

#### Step 2

Let's start the Hadoop process to count the total number of words in all the files available in the input directory, as follows −

$ hadoop jar $HADOOP\_HOME/share/hadoop/mapreduce/hadoop-mapreduceexamples-2.2.0.jar wordcount input output

#### Step 3

Step-2 will do the required processing and save the output in output/part-r00000 file, which you can check by using −

$cat output/\*

It will list down all the words along with their total counts available in all the files available in the input directory.

"AS 4

"Contribution" 1

"Contributor" 1

"Derivative 1

"Legal 1

"License" 1

"License"); 1

"Licensor" 1

"NOTICE” 1

"Not 1

"Object" 1

"Source” 1

"Work” 1

"You" 1

"Your") 1

"[]" 1

"control" 1

"printed 1

"submitted" 1

(50%) 1

(BIS), 1

(C) 1

(Don't) 1

(ECCN) 1

(INCLUDING 2

(INCLUDING, 2

.............

### Installing Hadoop in Pseudo Distributed Mode

Follow the steps given below to install Hadoop 3.2.0 in pseudo distributed mode.

#### Step 1 − Setting Up Hadoop

You can set Hadoop environment variables by appending the following commands to **~/.bashrc** file.

export HADOOP\_HOME=/usr/local/hadoop

export HADOOP\_MAPRED\_HOME=$HADOOP\_HOME

export HADOOP\_COMMON\_HOME=$HADOOP\_HOME

export HADOOP\_HDFS\_HOME=$HADOOP\_HOME

export YARN\_HOME=$HADOOP\_HOME

export HADOOP\_COMMON\_LIB\_NATIVE\_DIR=$HADOOP\_HOME/lib/native

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

export HADOOP\_INSTALL=$HADOOP\_HOME

Now apply all the changes into the current running system.

$ source ~/.bashrc

#### Step 2 − Hadoop Configuration

You can find all the Hadoop configuration files in the location “$HADOOP\_HOME/etc/hadoop”. It is required to make changes in those configuration files according to your Hadoop infrastructure.

$ cd $HADOOP\_HOME/etc/hadoop

In order to develop Hadoop programs in java, you have to reset the java environment variables in **hadoop-env.sh** file by replacing **JAVA\_HOME** value with the location of java in your system.

export JAVA\_HOME=/usr/local/jdk1.7.0\_71

The following are the list of files that you have to edit to configure Hadoop.

**core-site.xml**

The **core-site.xml** file contains information such as the port number used for Hadoop instance, memory allocated for the file system, memory limit for storing the data, and size of Read/Write buffers.

Open the core-site.xml and add the following properties in between <configuration>, </configuration> tags.

<configuration>

<property>

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

<value>hdfs://localhost:9000</value>

</property>

</configuration>

hdfs-site.xml

The **hdfs-site.xml** file contains information such as the value of replication data, namenode path, and datanode paths of your local file systems. It means the place where you want to store the Hadoop infrastructure.

Let us assume the following data.

dfs.replication (data replication value) = 1

(In the below given path /hadoop/ is the user name.

hadoopinfra/hdfs/namenode is the directory created by hdfs file system.)

namenode path = //home/hadoop/hadoopinfra/hdfs/namenode

(hadoopinfra/hdfs/datanode is the directory created by hdfs file system.)

datanode path = //home/hadoop/hadoopinfra/hdfs/datanode

Open this file and add the following properties in between the <configuration> </configuration> tags in this file.

<configuration>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.name.dir</name>

<value>file:///home/hadoop/hadoopinfra/hdfs/namenode </value>

</property>

<property>

<name>dfs.data.dir</name>

<value>file:///home/hadoop/hadoopinfra/hdfs/datanode </value>

</property>

</configuration>

**Note** − In the above file, all the property values are user-defined and you can make changes according to your Hadoop infrastructure.

yarn-site.xml

This file is used to configure yarn into Hadoop. Open the yarn-site.xml file and add the following properties in between the <configuration>, </configuration> tags in this file.

<configuration>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

</configuration>

mapred-site.xml

This file is used to specify which MapReduce framework we are using. By default, Hadoop contains a template of yarn-site.xml. First of all, it is required to copy the file from **mapred-site.xml.template** to **mapred-site.xml** file using the following command.

$ cp mapred-site.xml.template mapred-site.xml

Open **mapred-site.xml** file and add the following properties in between the <configuration>, </configuration>tags in this file.

<configuration>

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

</configuration>

### Verifying Hadoop Installation

The following steps are used to verify the Hadoop installation.

#### Step 1 − Name Node Setup

Set up the namenode using the command “hdfs namenode -format” as follows.

$ cd ~

$ hdfs namenode -format

The expected result is as follows.

10/24/14 21:30:55 INFO namenode.NameNode: STARTUP\_MSG:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

STARTUP\_MSG: Starting NameNode

STARTUP\_MSG: host = localhost/192.168.1.11

STARTUP\_MSG: args = [-format]

STARTUP\_MSG: version = 3.2.0

...

...

10/24/14 21:30:56 INFO common.Storage: Storage directory

/home/hadoop/hadoopinfra/hdfs/namenode has been successfully formatted.

10/24/14 21:30:56 INFO namenode.NNStorageRetentionManager: Going to

retain 1 images with txid >= 0

10/24/14 21:30:56 INFO util.ExitUtil: Exiting with status 0

10/24/14 21:30:56 INFO namenode.NameNode: SHUTDOWN\_MSG:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SHUTDOWN\_MSG: Shutting down NameNode at localhost/192.168.1.11

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### Step 2 − Verifying Hadoop dfs

The following command is used to start dfs. Executing this command will start your Hadoop file system.

$ start-dfs.sh

The expected output is as follows −

10/24/14 21:37:56

Starting namenodes on [localhost]

localhost: starting namenode, logging to /home/hadoop/hadoop

3.2.0/logs/hadoop-hadoop-namenode-localhost.out

localhost: starting datanode, logging to /home/hadoop/hadoop

3.2.0/logs/hadoop-hadoop-datanode-localhost.out

Starting secondary namenodes [0.0.0.0]

#### Step 3 − Verifying Yarn Script

The following command is used to start the yarn script. Executing this command will start your yarn daemons.

$ start-yarn.sh

The expected output as follows −

10/24/14 21:37:56

Starting namenodes on [localhost]

localhost: starting namenode, logging to /home/hadoop/hadoop

2.4.1/logs/hadoop-hadoop-namenode-localhost.out

localhost: starting datanode, logging to /home/hadoop/hadoop

2.4.1/logs/hadoop-hadoop-datanode-localhost.out

Starting secondary namenodes [0.0.0.0]

#### Step 3 − Verifying Yarn Script

The following command is used to start the yarn script. Executing this command will start your yarn daemons.

$ start-yarn.sh

The expected output as follows −

starting yarn daemons

starting resourcemanager, logging to /home/hadoop/hadoop

2.4.1/logs/yarn-hadoop-resourcemanager-localhost.out

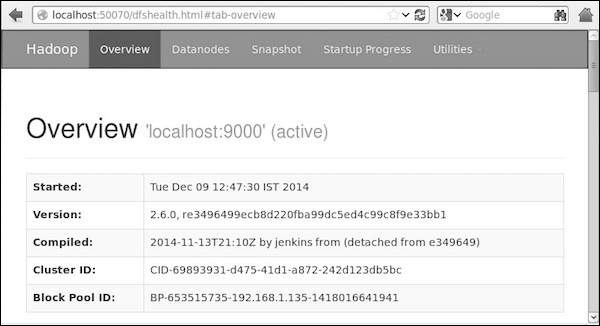
localhost: starting nodemanager, logging to /home/hadoop/hadoop

2.4.1/logs/yarn-hadoop-nodemanager-localhost.out

#### Step 4 − Accessing Hadoop on Browser

The default port number to access Hadoop is 50070. Use the following url to get Hadoop services on browser.

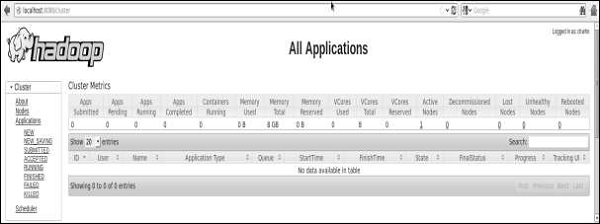
http://localhost:50070/



#### Step 5 − Verify All Applications for Cluster

The default port number to access all applications of cluster is 8088. Use the following url to visit this service.

http://localhost:8088/



So far we have setup a working Hadoop environment. You can use Hadoop as a standalone mode but the purpose of this lab practice is let you familiar with REAL BDA platforms. It refers to a distributed system and many machines organized and ruing in Cluster manner. Therefore we will use Hadoop in its Pseudo Distributed Mode.

We have the knowledge that Hadoop manages data in a distributed manner with an important component called HDFS. In other words, Hadoop provide a distributed storage capacity by HDFS.

Let us briefly review HDFS.

## Hadoop - HDFS

Hadoop File System was developed using distributed file system design. It is run on commodity hardware. Unlike other distributed systems, HDFS is highly fault tolerant and designed using low-cost hardware.

HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

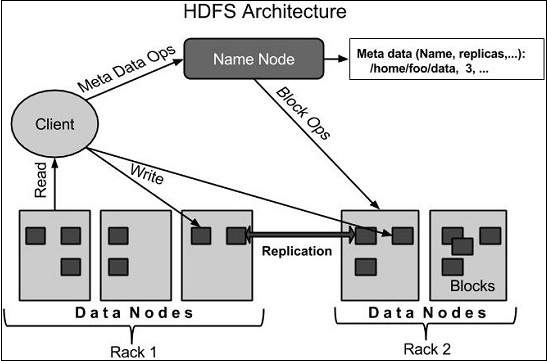
### Features of HDFS

It is suitable for the distributed storage and processing.

* Hadoop provides a command interface to interact with HDFS.
* The built-in servers of **namenode** and **datanode** help users to easily check the status of cluster.
* Streaming access to file system data.
* HDFS provides file permissions and authentication.

### HDFS Architecture

Given below is the architecture of a Hadoop File System.



HDFS follows the master-slave architecture and it has the following elements.

### Namenode

The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks −

* Manages the file system namespace.
* Regulates client’s access to files.
* It also executes file system operations such as renaming, closing, and opening files and directories.

### Datanode

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

* Datanodes perform read-write operations on the file systems, as per client request.
* They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

### Block

Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

### Goals of HDFS

**Fault detection and recovery** − Since HDFS includes a large number of commodity hardware, failure of components is frequent. Therefore HDFS should have mechanisms for quick and automatic fault detection and recovery.

**Huge datasets** − HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.

**Hardware at data** − A requested task can be done efficiently, when the computation takes place near the data. Especially where huge datasets are involved, it reduces the network traffic and increases the throughput.

### HDFS Operations

### Starting HDFS

Initially you have to format the configured HDFS file system, open namenode (HDFS server), and execute the following command.

$ hadoop namenode -format

After formatting the HDFS, start the distributed file system. The following command will start the namenode as well as the data nodes as cluster.

$ start-dfs.sh

### Listing Files in HDFS

After loading the information in the server, we can find the list of files in a directory, status of a file, using **‘ls’**. Given below is the syntax of **ls** that you can pass to a directory or a filename as an argument.

$ $HADOOP\_HOME/bin/hadoop fs -ls <args>

### Inserting Data into HDFS

Assume we have data in the file called file.txt in the local system which is ought to be saved in the HDFS file system. Follow the steps given below to insert the required file in the Hadoop file system.

#### Step 1

You have to create an input directory.

$ $HADOOP\_HOME/bin/hadoop fs -mkdir /user/input

#### Step 2

Transfer and store a data file from local systems to the Hadoop file system using the put command.

$ $HADOOP\_HOME/bin/hadoop fs -put /home/file.txt /user/input

#### Step 3

You can verify the file using ls command.

$ $HADOOP\_HOME/bin/hadoop fs -ls /user/input

### Retrieving Data from HDFS

Assume we have a file in HDFS called **outfile**. Given below is a simple demonstration for retrieving the required file from the Hadoop file system.

#### Step 1

Initially, view the data from HDFS using **cat** command.

$ $HADOOP\_HOME/bin/hadoop fs -cat /user/output/outfile

#### Step 2

Get the file from HDFS to the local file system using **get** command.

$ $HADOOP\_HOME/bin/hadoop fs -get /user/output/ /home/hadoop\_tp/

### Shutting Down the HDFS

You can shut down the HDFS by using the following command.

$ stop-dfs.sh

There are many more commands in **"$HADOOP\_HOME/bin/hadoop fs"** than are demonstrated here, although these basic operations will get you started. For more HDFS commands please see appendix III.

## Hadoop - MapReduce

MapReduce is a software developing framework. Using this framework we can write applications to process huge amounts of data, in **parallel**, on large clusters of commodity hardware in a reliable manner.

### What is MapReduce?

MapReduce is a processing technique and a program model for distributed computing based on **java**. The MapReduce algorithm contains two important tasks, namely **Map** and **Reduce**:

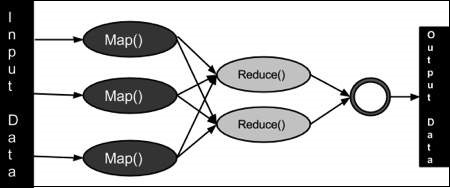
* Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).
* Reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called **mappers** and **reducers**. Decomposing a data processing application into *mappers* and *reducers* is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

### The Algorithm

Generally MapReduce paradigm is based on sending the computer and application program to where the data resides! This is called edge-computing. In a conventional term, we say moving program to data, rather than move data to program.

* MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
  + **Map stage** − The map or mapper’s job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
  + **Reduce stage** − This stage is the combination of the **Shuffle** stage and the **Reduce** stage. The Reducer’s job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.
* During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
* The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
* Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
* After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.



### Inputs and Outputs (Java Perspective)

The MapReduce framework operates on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.

The key and the value classes should be in serialized manner by the framework and hence, need to implement the Writable interface. Additionally, the key classes have to implement the Writable-Comparable interface to facilitate sorting by the framework. Input and Output types of a **MapReduce job** − (Input) <k1, v1> → map → <k2, v2> → reduce → <k3, v3>(Output).

|  |  |  |
| --- | --- | --- |
|  | **Input** | **Output** |
| **Map** | <k1, v1> | list (<k2, v2>) |
| **Reduce** | <k2, list(v2)> | list (<k3, v3>) |

### Terminology

* **PayLoad** − Applications implement the Map and the Reduce functions, and form the core of the job.
* **Mapper** − Mapper maps the input key/value pairs to a set of intermediate key/value pair.
* **NamedNode** − Node that manages the Hadoop Distributed File System (HDFS).
* **DataNode** − Node where data is presented in advance before any processing takes place.
* **MasterNode** − Node where JobTracker runs and which accepts job requests from clients.
* **SlaveNode** − Node where Map and Reduce program runs.
* **JobTracker** − Schedules jobs and tracks the assign jobs to Task tracker.
* **Task Tracker** − Tracks the task and reports status to JobTracker.
* **Job** − A program is an execution of a Mapper and Reducer across a dataset.
* **Task** − An execution of a Mapper or a Reducer on a slice of data.
* **Task Attempt** − A particular instance of an attempt to execute a task on a SlaveNode.

### Example Scenario

Given below is the data regarding the electrical consumption of an organization. It contains the monthly electrical consumption and the annual average for various years.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Jan** | **Feb** | **Mar** | **Apr** | **May** | **Jun** | **Jul** | **Aug** | **Sep** | **Oct** | **Nov** | **Dec** | **Avg** |
| 1979 | 23 | 23 | 2 | 43 | 24 | 25 | 26 | 26 | 26 | 26 | 25 | 26 | 25 |
| 1980 | 26 | 27 | 28 | 28 | 28 | 30 | 31 | 31 | 31 | 30 | 30 | 30 | 29 |
| 1981 | 31 | 32 | 32 | 32 | 33 | 34 | 35 | 36 | 36 | 34 | 34 | 34 | 34 |
| 1984 | 39 | 38 | 39 | 39 | 39 | 41 | 42 | 43 | 40 | 39 | 38 | 38 | 40 |
| 1985 | 38 | 39 | 39 | 39 | 39 | 41 | 41 | 41 | 00 | 40 | 39 | 39 | 45 |

If the above data is given as input, we have to write applications to process it and produce results such as finding the year of maximum usage, year of minimum usage, and so on. It is simple and easy task for a finite number of records. However think of the data representing the electrical consumption of all the largescale industries of a particular state. Then it will be a problem:

* The whole task will take a lot of time to execute.
* There will be a heavy network traffic when we move data from source to network server and so on.

To solve these problems, we have to use the MapReduce framework.

The following is a walkover for the programmers. They should simply write the logic to produce the required output, and pass the data to the application they have written and leave the rest to Hadoop.

### Input Data

The above data is saved as **sample.txt** and given as input. The input file looks as shown below.

1979 23 23 2 43 24 25 26 26 26 26 25 26 25

1980 26 27 28 28 28 30 31 31 31 30 30 30 29

1981 31 32 32 32 33 34 35 36 36 34 34 34 34

1984 39 38 39 39 39 41 42 43 40 39 38 38 40

1985 38 39 39 39 39 41 41 41 00 40 39 39 45

### Example Program

Given below is the sample program to the sample data using MapReduce framework. The attention should be drawn comments in the code.

package hadoop;

import java.util.\*;

import java.io.IOException;

import java.io.IOException;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.conf.\*;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapred.\*;

import org.apache.hadoop.util.\*;

public class ProcessUnits {

**//Mapper class**

public static class E\_EMapper extends MapReduceBase implements

Mapper<LongWritable , /\*Input key Type \*/

Text, /\*Input value Type\*/

Text, /\*Output key Type\*/

IntWritable> /\*Output value Type\*/

{

**//Map function**

public void map(LongWritable key, Text value,

OutputCollector<Text, IntWritable> output,

Reporter reporter) throws IOException {

String line = value.toString();

String lasttoken = null;

StringTokenizer s = new StringTokenizer(line,"\t");

String year = s.nextToken();

while(s.hasMoreTokens()) {

lasttoken = s.nextToken();

}

int avgprice = Integer.parseInt(lasttoken);

output.collect(new Text(year), new IntWritable(avgprice));

}

}

//Reducer class

public static class E\_EReduce extends MapReduceBase implements Reducer< Text, IntWritable, Text, IntWritable > {

//Reduce function

public void reduce( Text key, Iterator <IntWritable> values,

OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {

int maxavg = 30;

int val = Integer.MIN\_VALUE;

while (values.hasNext()) {

if((val = values.next().get())>maxavg) {

output.collect(key, new IntWritable(val));

}

}

}

}

//Main function

public static void main(String args[])throws Exception {

JobConf conf = new JobConf(ProcessUnits.class);

conf.setJobName("max\_eletricityunits");

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(E\_EMapper.class);

conf.setCombinerClass(E\_EReduce.class);

conf.setReducerClass(E\_EReduce.class);

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, new Path(args[0]));

FileOutputFormat.setOutputPath(conf, new Path(args[1]));

JobClient.runJob(conf);

}

}

Save the above program as **ProcessUnits.java.** The compilation and execution of the program is explained below.

## Compilation and Execution of Process Units Program

Let us assume we are in the home directory of a Hadoop user (e.g. /home/hadoop).

Follow the steps given below to compile and execute the above program.

### Step 1

The following command is to create a directory to store the compiled java classes.

$ mkdir units

### Step 2

Download **Hadoop-core-1.2.1.jar,** which is used to compile and execute the MapReduce program. Visit the following link [mvnrepository.com](http://mvnrepository.com/artifact/org.apache.hadoop/hadoop-core/1.2.1) to download the jar. Let us assume the downloaded folder is **/home/hadoop/.**

### Step 3

The following commands are used for compiling the **ProcessUnits.java** program and creating a jar for the program.

$ javac -classpath hadoop-core-1.2.1.jar -d units ProcessUnits.java

$ jar -cvf units.jar -C units/.

### Step 4

The following command is used to create an input directory in HDFS.

$HADOOP\_HOME/bin/hadoop fs -mkdir input\_dir

### Step 5

The following command is used to copy the input file named **sample.txt** in the input directory of HDFS.

$HADOOP\_HOME/bin/hadoop fs -put /home/hadoop/sample.txt input\_dir

### Step 6

The following command is used to verify the files in the input directory.

$HADOOP\_HOME/bin/hadoop fs -ls input\_dir/

### Step 7

The following command is used to run the Eleunit\_max application by taking the input files from the input directory.

$HADOOP\_HOME/bin/hadoop jar units.jar hadoop.ProcessUnits input\_dir output\_dir

Wait for a while until the file is executed. After execution, as shown below, the output will contain the number of input splits, the number of Map tasks, the number of reducer tasks, etc.

INFO mapreduce.Job: Job job\_1414748220717\_0002

completed successfully

14/10/31 06:02:52

INFO mapreduce.Job: Counters: 49

File System Counters

FILE: Number of bytes read = 61

FILE: Number of bytes written = 279400

FILE: Number of read operations = 0

FILE: Number of large read operations = 0

FILE: Number of write operations = 0

HDFS: Number of bytes read = 546

HDFS: Number of bytes written = 40

HDFS: Number of read operations = 9

HDFS: Number of large read operations = 0

HDFS: Number of write operations = 2 Job Counters

Launched map tasks = 2

Launched reduce tasks = 1

Data-local map tasks = 2

Total time spent by all maps in occupied slots (ms) = 146137

Total time spent by all reduces in occupied slots (ms) = 441

Total time spent by all map tasks (ms) = 14613

Total time spent by all reduce tasks (ms) = 44120

Total vcore-seconds taken by all map tasks = 146137

Total vcore-seconds taken by all reduce tasks = 44120

Total megabyte-seconds taken by all map tasks = 149644288

Total megabyte-seconds taken by all reduce tasks = 45178880

Map-Reduce Framework

Map input records = 5

Map output records = 5

Map output bytes = 45

Map output materialized bytes = 67

Input split bytes = 208

Combine input records = 5

Combine output records = 5

Reduce input groups = 5

Reduce shuffle bytes = 6

Reduce input records = 5

Reduce output records = 5

Spilled Records = 10

Shuffled Maps = 2

Failed Shuffles = 0

Merged Map outputs = 2

GC time elapsed (ms) = 948

CPU time spent (ms) = 5160

Physical memory (bytes) snapshot = 47749120

Virtual memory (bytes) snapshot = 2899349504

Total committed heap usage (bytes) = 277684224

File Output Format Counters

Bytes Written = 40

### Step 8

The following command is used to verify the resultant files in the output folder.

$HADOOP\_HOME/bin/hadoop fs -ls output\_dir/

### Step 9

The following command is used to see the output in **Part-00000** file. This file is generated by HDFS.

$HADOOP\_HOME/bin/hadoop fs -cat output\_dir/part-00000

Below is the output generated by the MapReduce program.

1981 34

1984 40

1985 45

### Step 10

The following command is used to copy the output folder from HDFS to the local file system for analyzing.

$HADOOP\_HOME/bin/hadoop fs -cat output\_dir/part-00000/bin/hadoop dfs get output\_dir /home/hadoop

### Question

1. The above work through, Step 5, why we used copy? What it mean and what we should use if the Hadoop is deployed on other machine or a remote server?
2. Step 10 why need to copy the output from HDFS to a local file system?

## How to Interact with MapReduce Jobs

Usage − hadoop job [GENERIC\_OPTIONS]

The following are the Generic Options available in a Hadoop job.

|  |  |
| --- | --- |
| **Sr.No.** | **GENERIC\_OPTION & Description** |
| 1 | **-submit <job-file>**  Submits the job. |
| 2 | **-status <job-id>**  Prints the map and reduce completion percentage and all job counters. |
| 3 | **-counter <job-id> <group-name> <countername>**  Prints the counter value. |
| 4 | **-kill <job-id>**  Kills the job. |
| 5 | **-events <job-id> <fromevent-#> <#-of-events>**  Prints the events' details received by jobtracker for the given range. |
| 6 | **-history [all] <jobOutputDir> - history < jobOutputDir>**  Prints job details, failed and killed tip details. More details about the job such as successful tasks and task attempts made for each task can be viewed by specifying the [all] option. |
| 7 | **-list[all]**  Displays all jobs. -list displays only jobs which are yet to complete. |
| 8 | **-kill-task <task-id>**  Kills the task. Killed tasks are NOT counted against failed attempts. |
| 9 | **-fail-task <task-id>**  Fails the task. Failed tasks are counted against failed attempts. |
| 10 | **-set-priority <job-id> <priority>**  Changes the priority of the job. Allowed priority values are VERY\_HIGH, HIGH, NORMAL, LOW, VERY\_LOW |

### To see the status of job

$ $HADOOP\_HOME/bin/hadoop job -status <JOB-ID>

e.g.

$ $HADOOP\_HOME/bin/hadoop job -status job\_201310191043\_0004

### To see the history of job output-dir

$ $HADOOP\_HOME/bin/hadoop job -history <DIR-NAME>

e.g.

$ $HADOOP\_HOME/bin/hadoop job -history /user/expert/output

### To kill the job

$ $HADOOP\_HOME/bin/hadoop job -kill <JOB-ID>

e.g.

$ $HADOOP\_HOME/bin/hadoop job -kill job\_201310191043\_0004

Above all, shows you the fundamental Hadoop and MapReduce structure and framework. We provide a basic work though for use Hadoop. However this is minimum since Hadoop in real life is mostly deployed on a cluster and works on multiple node (“Master- Slave”) fashion. The Hadoop can also run in Streaming mode. This utility allows users to create and run Map/Reduce jobs with any executable or script (such as python, Perl and Ruby) in two phases: mapper and reducer.

Further Hadoop tutorials

• Yahoo! Hadoop Tutorial:

<http://developer.yahoo.com/hadoop/tutorial/>

• Cloudera Hadoop Tutorial:

<http://www.cloudera.com/content/cloudera-content/cloudera-docs/HadoopTutorial/CDH4/Hadoop-Tutorial.html>

• How to Debug MapReduce Programs:

[http://wiki.apache.org/hadoop/HowToDebugMapReducePro](http://wiki.apache.org/hadoop/HowToDebugMapReducePrograms)

# III Sample project

This sample project assume you are familiar with the Hadoop set up. The main purpose of this sample project is demonstrate how use Hadoop to run a BIG data analysis project.

Task – word count

You are given a book (hug text file) the task is count the words appears in that book.

Let us review the work through of the steps need for accomplish the task with Hadoop.

1. We need a Linux OS
2. We need a JRE and a JDK (or text editor to write Java code)
3. We need a Hadoop deployment (stand alone or distributed)
4. We need to write java code (in MapReduce mode) and to compile java code and to generate a job Jar code
5. We need to upload (print to data source) data
6. We need to run MapReduce
7. We need to get results
8. We need to download results from server to local.

For our example project we take **an easy route**. We assume you have a windows machine. We will use a virtual box which has:

* CentOS 6.2
* JDK 6 (1.6.0 32)
* Hadoop 2.3.0
* Eclipse 4.2.6 (Juno)

This virtual machine is provided by one of the most adopted Hadoop provider Cloudera. You can download the latest version from,

<https://www.cloudera.com/downloads/quickstart_vms/5-13.html> (select Virtual Box)

Cloudera QuickStart VMs (single-node cluster) make it easy to quickly get hands-on with Cloudera Hadoop (CDH) for testing and self-learning purposes, and include Cloudera Manager for managing your cluster. Cloudera QuickStart VM also includes a tutorial, sample data, and scripts for getting started.  We have downloaded an older version of the same virtual machine but for learning purpose, it is enough.

You can download the virtual machine from BOX, it is sit in the folder: cloudera-quickstart-vm-5.0.0-0-virtualbox

<https://box.xjtlu.edu.cn/d/b5302d2a397146f0ba62/>

(Please create a new folder INT303 on your desktop, then download both ‘.vmdk’ and ‘.vbox’ file to this folder),

Let us getting start.

You can open the virtual machine by double clicking file: “cloudera-quickstart-vm-5.0.0-0-virtualbox.vbox” (the size of the file is about 10k),

Now your virtual machine should now appear in the left column of Virtual Box. Select it and click on Start to launch it. You just need to click “ok” if any window pops out. It will spend few minutes to start the virtual machine, please be patient.

Note: There is a possibility that a previous user may leave a virtual machine interface unclosed. In this case you need to close the virtual machine interface and launch your own virtual machine. The method is selecting the current interface then right click select the remove. Then re-launch it by double clicking the ‘.vbox’ file.

## Create Accounts

Once you launch the VM, you are automatically logged in as the cloudera user:

* username: cloudera
* password: cloudera

The cloudera account has sudo privileges in the VM. The root account password is cloudera.

 You will need copy text from local computer to virtual machine: click Devices (on the left corner of the virtual machine, choose Shared Clipboard->Host to Guest)

You can also Download other source files (pg100.txt, WordCount.java). They are needed for the example) from Learning Mall or from <https://box.xjtlu.edu.cn/d/1655a90f3c0c422796d2/>.

Please download these 2 source files (pg100.txt, WordCount.java) into the virtual machine which will be used for your later work.

## Develop, debug and test wordCount.jar

The first thing to use Hadoop is to develop a job.jar file. You can follow “Compilation and Execution of Process Units Program” in previous section to compile your java code and produce a jar file. Then you can submit your code and data file to HDFS to run Hadoop jobs.

Since our Virtual machine has a JDK (eclipse), we will use this integrated developing environment to develop our Java source and Jar executable.

If you are not familiar with Eclipse, you need to read related document and familiar with Eclipse.

Eclipse is GUI based program development environment (kit). It is widely used JDK (java development Kit). Once you configured correctly you test, debug and run simulation of Hadoop in Eclipse completely.

Further Eclipse tutorials:

• Genera Eclipse tutorial:

<http://www.vogella.com/articles/Eclipse/article.html>.

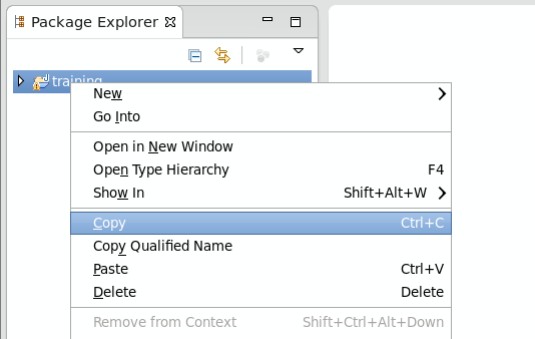
• Tutorial on how to use the Eclipse debugger:

[http://www.vogella.com/articles/ EclipseDebugging/article.html](http://www.vogella.com/articles/EclipseDebugging/article.html).

## Create a new Eclipse Hadoop project

In this section you will create a new Eclipse Hadoop project, compile, and execute it. Remember our project is to count works appear in a txt file.

### Create a new project.

Open Eclipse. Goto Package Explorer, right click on the project “training”, click “Copy”, then click “paste”, give new project a name called “WordCount”, by enter WordCount into project name and click OK, see Figure 2.

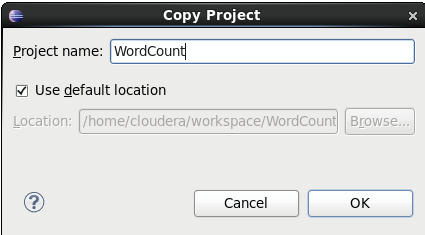
Figure 1. Create a Hadoop Project.

Figure 2. Create a Hadoop Project.

### Create a new package (in the newly created project)

Create a new package called “edu.xjtlu.cse313.wordcount” by right-clicking on the src node and selecting New → Package. See Figure 3.

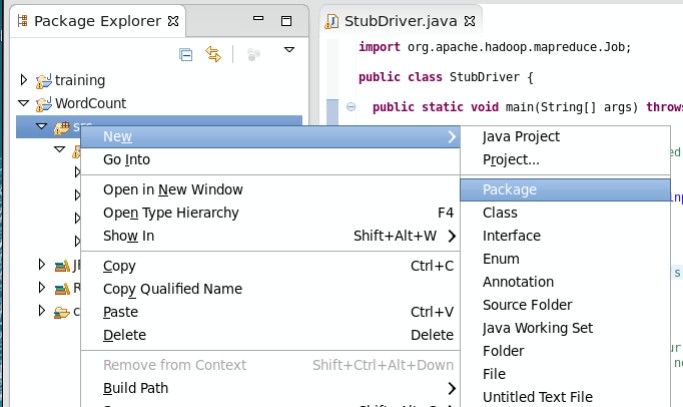


Figure 3: Create a Hadoop Project.

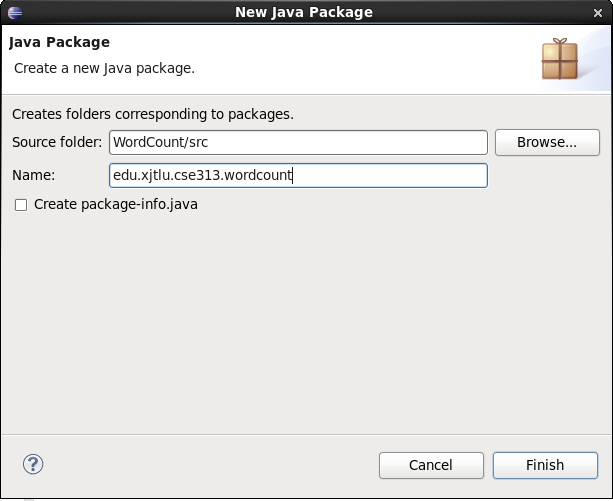
Enter edu.xjtlu.cse313.wordcount in the Name field and click Finish. See Figure 4.

Figure 4: Create a Hadoop Project.

### Create a new class

Create a new class in that package called “WordCount” by right-clicking on the edu.xjtlu.cse313.wordcount node and selecting New → Class. See Figure 5.

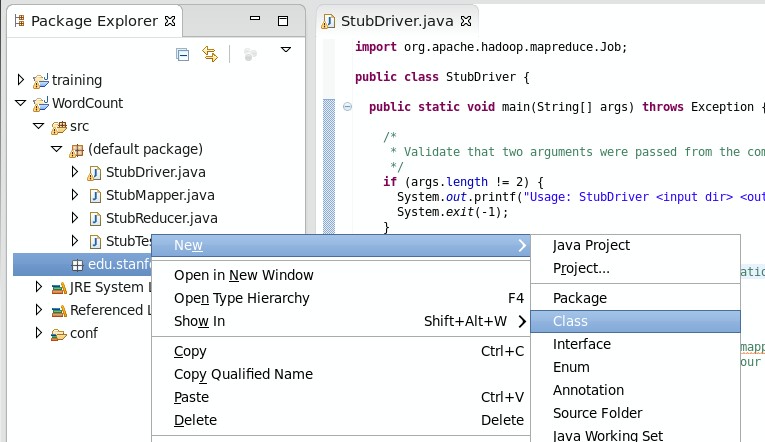


Figure 5. Create a Hadoop Project.

In the pop-up dialog, enter WordCount as the Name. See Figure 6.

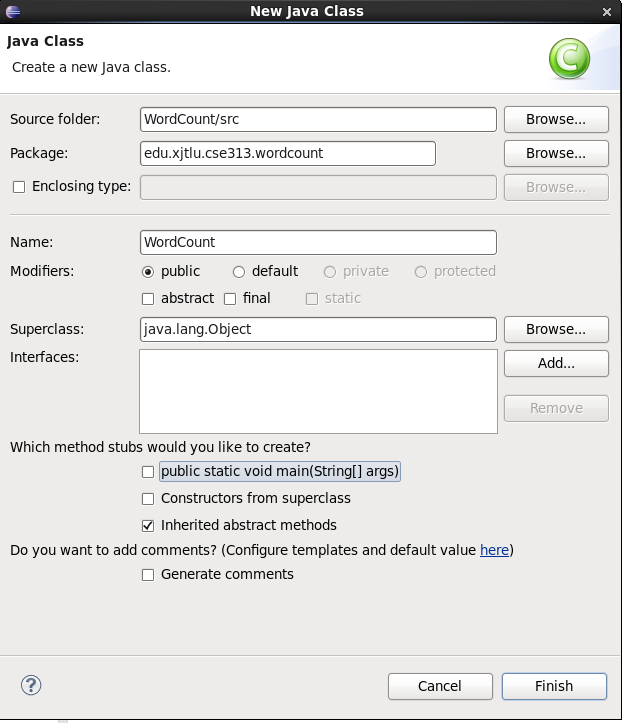


Figure 6. Create a java class.

### Linked with Hadoop configuration

In the Superclass field, enter Configured and click the Browse button. From the pop- up window select Configured – org.apache.hadoop.conf and click the OK button. See Figure 7.

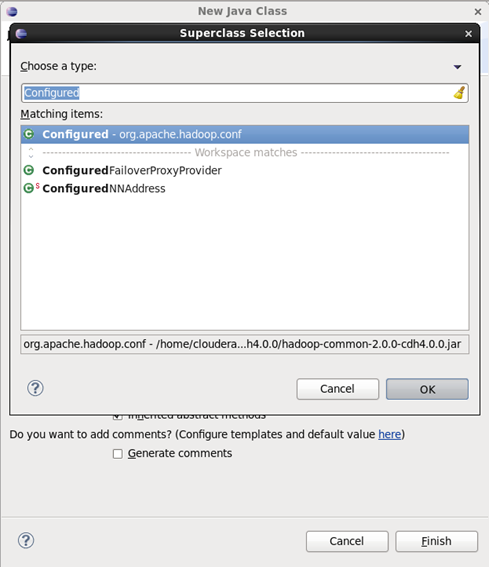


Figure 7. Create java file.

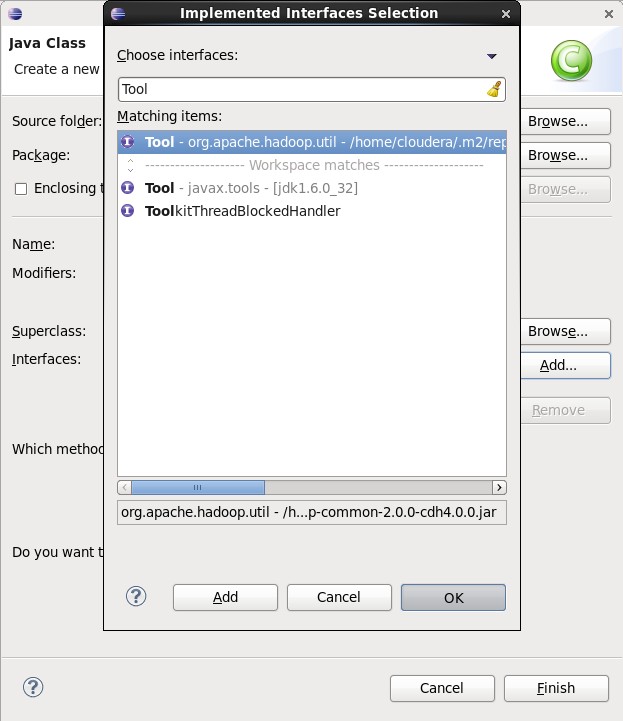
In the Interfaces section, click the Add button. From the pop-up window select Tool - org.apache.hadoop.util and click the OK button. See Figure 8.

Figure 8. Link Hadoop Util as tool

Check the boxes for public static void main (String args[]) and inherited abstract methods and click the Finish button. See Figure 9.

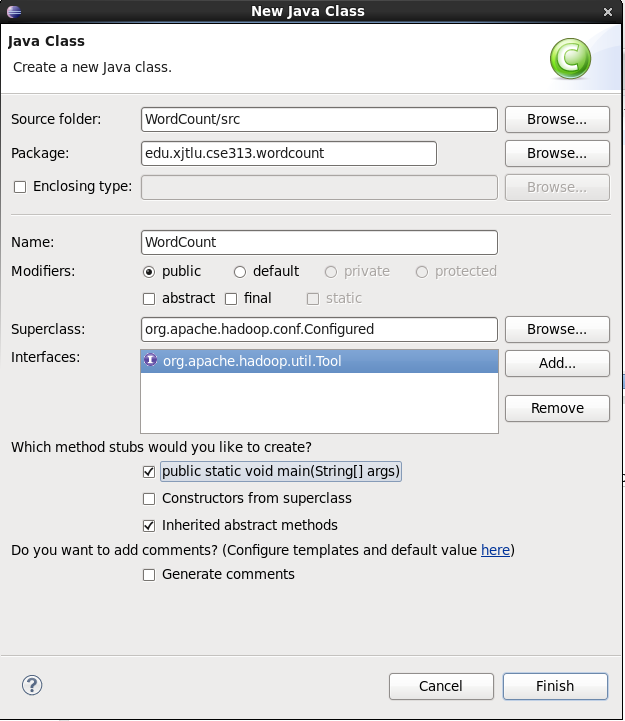


Figure 9. Create WordCount.java class.

### Develop your Wordcount Java Job with MapReduce program model

You will now have a rough skeleton of a Java file as in Figure 10. You can now add code to this class to implement your Hadoop job.

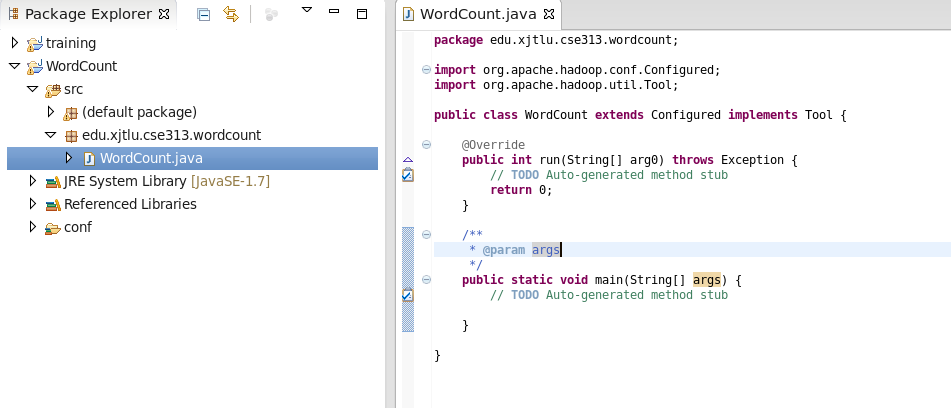


Figure 10. Create WordCount.java.

Rather than implement a job from scratch, copy the contents from WordCount.java (you can find it in the ICE) and paste it into the WordCount.java files. See Figure 11. The code in WordCount.java calculates the frequency of each word in a given dataset.

If you don’t understand the java code. It is good idea to look into Java code and discuss with your classmate.

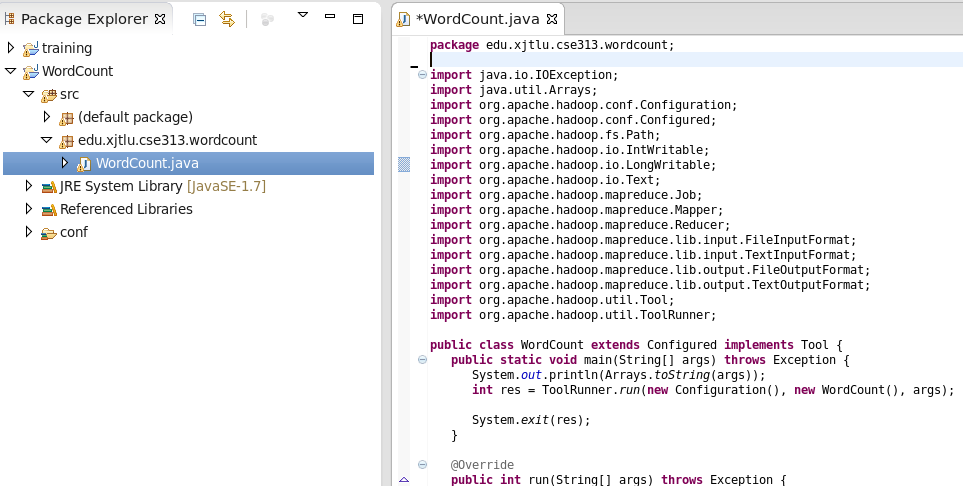


Figure 11. Create WordCount.java.

### Test/debug your WordCount.Java with input dataset

Find source data file pg100.txt from the folder you just downloaded. Put pg100.txt in ~/workspace/WordCount directory.

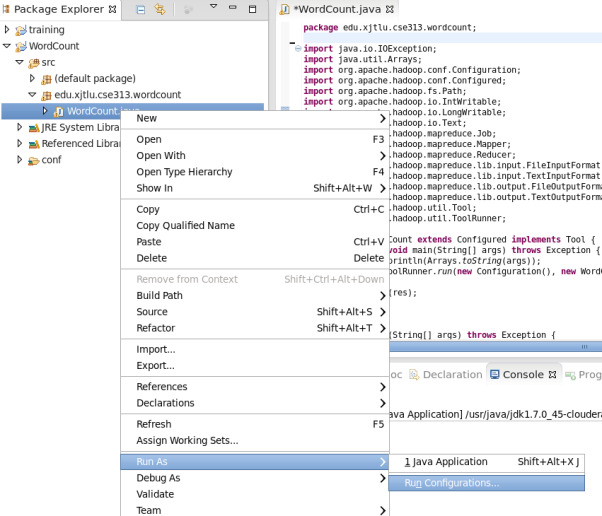
Right-click on the project and select Run As → Run Configurations. See Figure 12.

Figure 12. Run WordCount.java.

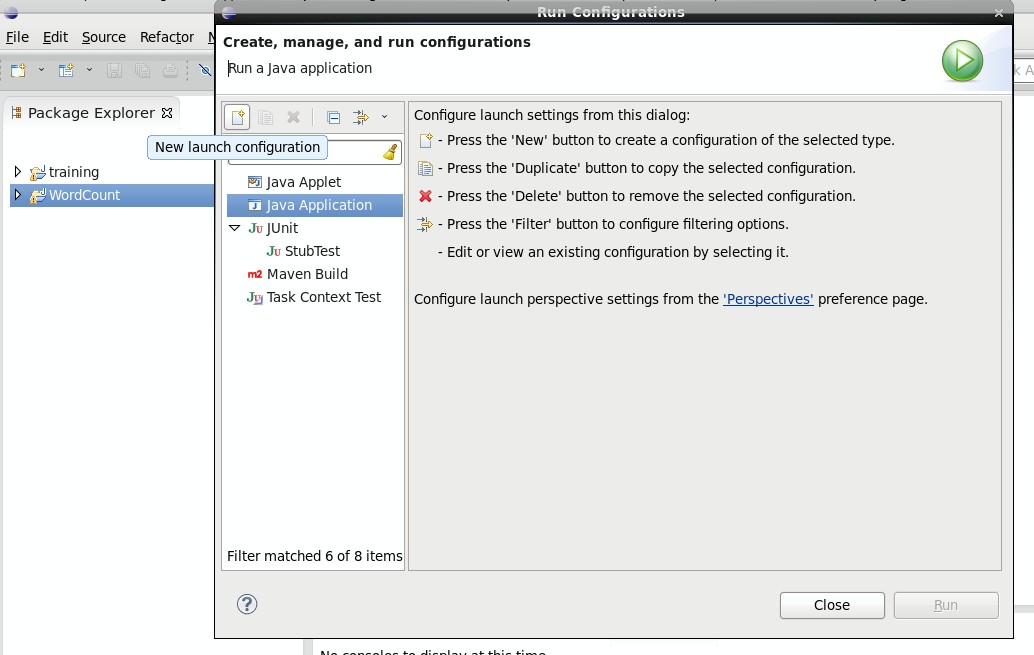
In the pop-up dialog, select the Java Application node and click the New launch configuration button in the upper left corner. See Figure 13,

Figure 13 Run WordCount.java.

Enter a name in the Name field and WordCount in the Main class field. See Figure 14.

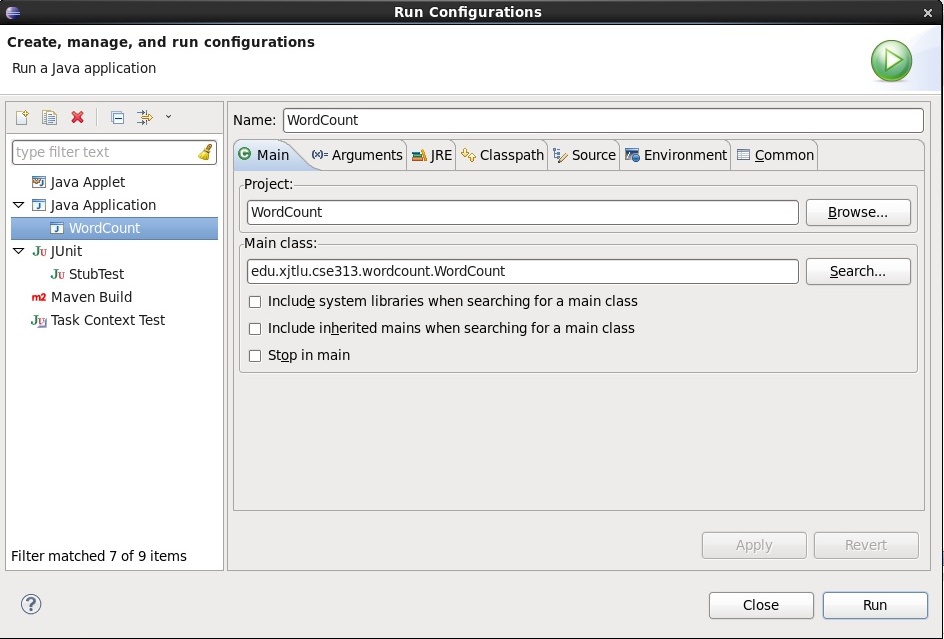


Figure 14. Run WordCount.java.

Switch to the Arguments tab and put “pg100.txt output” in the Program arguments field. See Figure 15. Click Apply and Close.

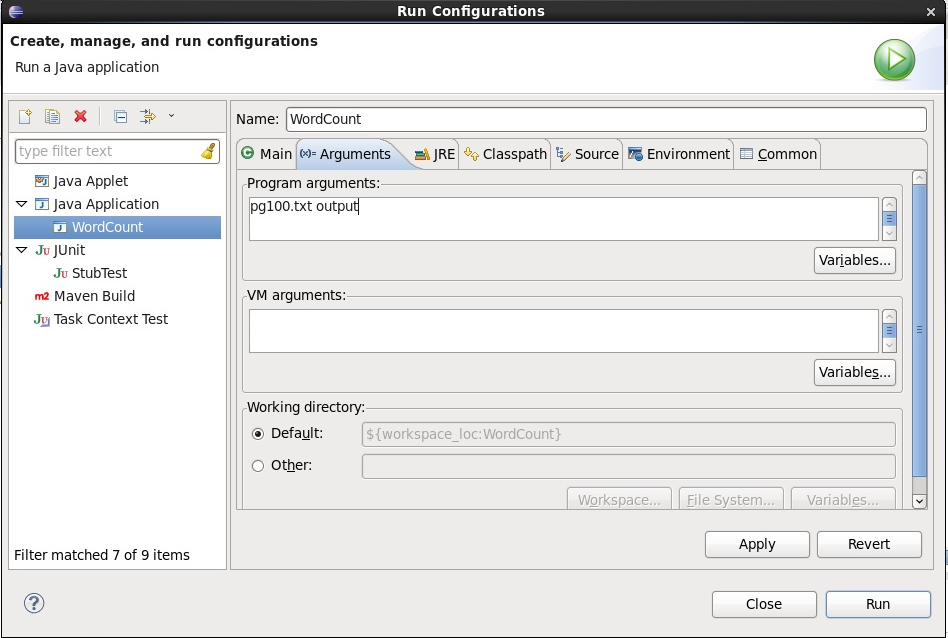


Figure 15. Run WordCount.java.

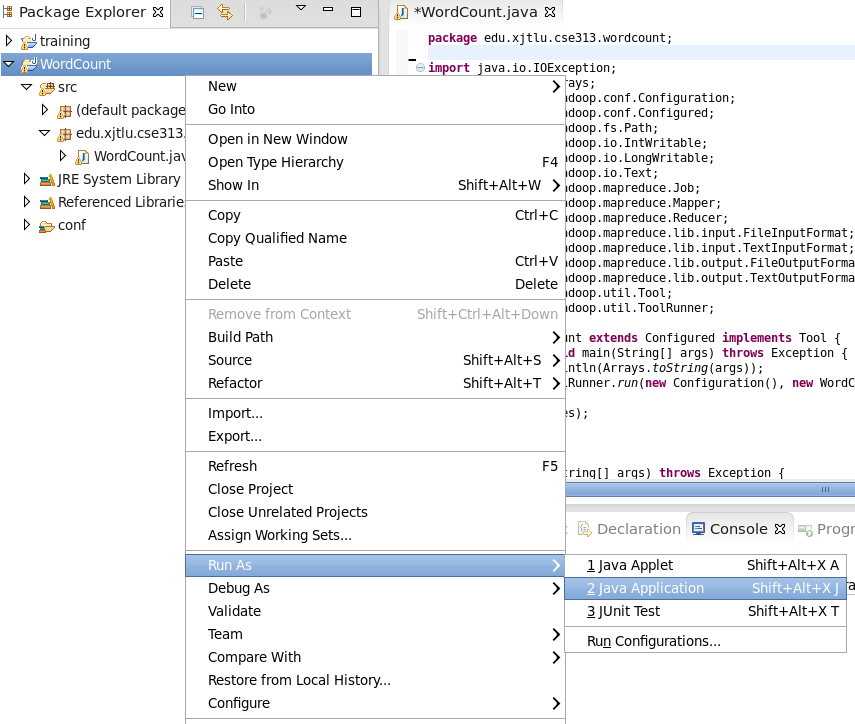
Right-click on the project and select Run As → Java Application. See Figure 16.

Figure 16. Run WordCount.java.

In the pop-up dialog select WordCount – edu.xjtlu.cse313.wordcount from the selection list and click OK. See Figure 17.

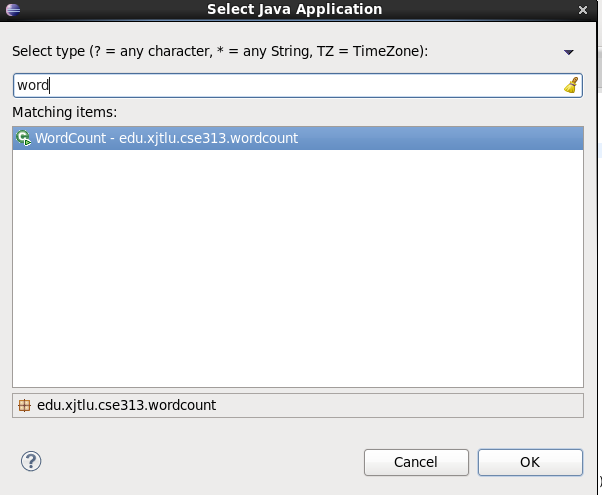


Figure 17. Run WordCount.java.

### Check up your results

You will see the command output in the console window, and if the job succeeds, you’ll find the results in the ~/workspace/WordCount/output directory. If the job fails complaining that it cannot find the input file, make sure that the pg100.txt file is located in the ~/workspace/WordCount directory.

### Export your java project (jar) from eclipse ready for submission to Hadoop

Once you have a successful java project, you need to export it and build an executable (jar). Right-click on the project and select Export from Eclipes. See Figure 18.

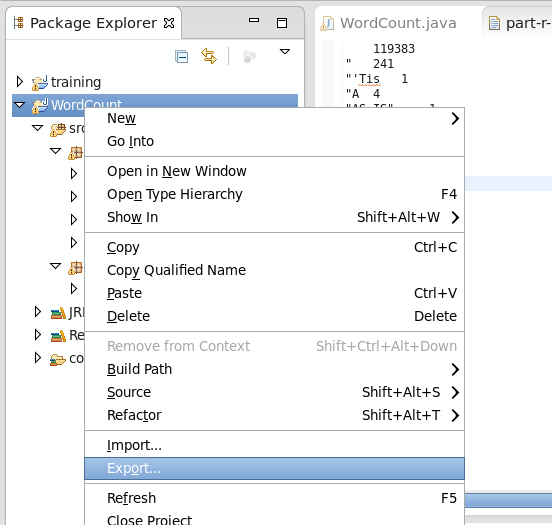


Figure 18. Export a Hadoop Project.

In the pop-up dialog, expand the Java node and select JAR file. See Figure 19. Click Next >.

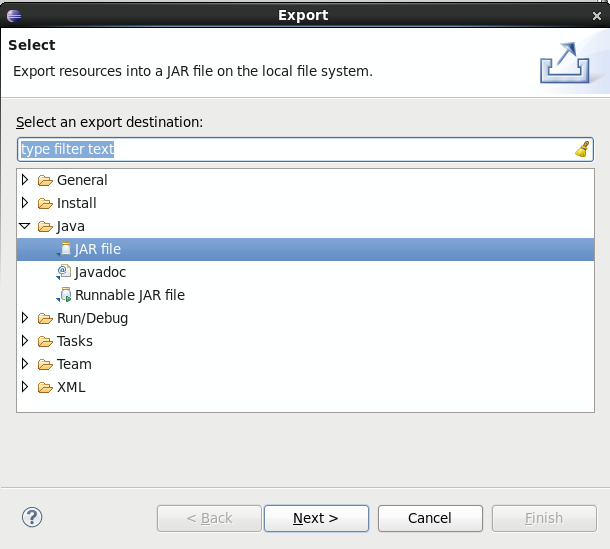


Figure 19. Export a Hadoop Project.

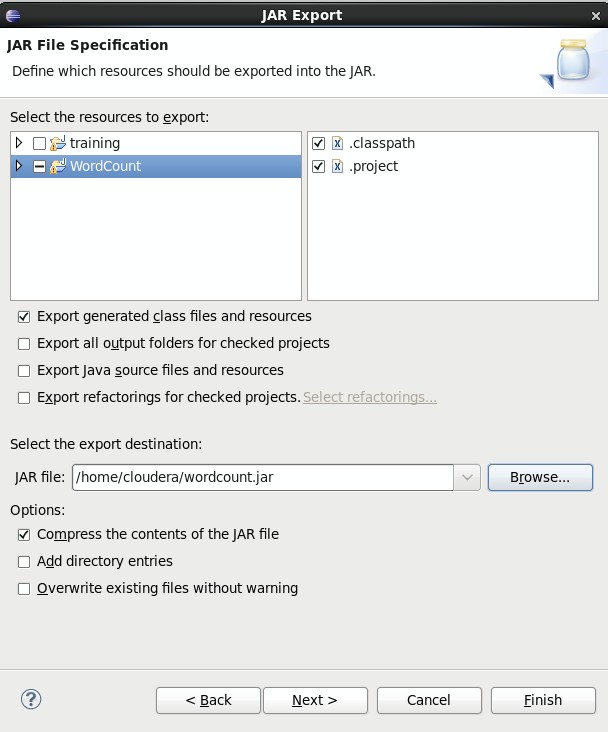
Enter /home/cloudera/wordcount.jar in the JAR file field and click Finish. See Figure 20.

Figure 20. Export a Hadoop Project.

If you see an error dialog warning that the project compiled with warnings, you can simply click OK.

With the above 8 steps we have our WordCount.jar now. We can then submit data source and data analysis program to Hadoop to take advantage of its distributed storage and parallel computing.

## Run WordCount on Hadoop

We knew that Generally Hadoop can be run in three modes.

* **Standalone (or local) mode:** There are no daemons used in this mode. Hadoop uses the local file system as a substitute for HDFS file system. The jobs will run as if there is 1 mapper and 1 reducer.
* **Pseudo-distributed mode**: All the daemons run on a single machine and this setting mimics the behavior of a cluster. All the daemons run on your machine locally using the HDFS protocol. There can be multiple mappers and reducers.
* **Fully-distributed mode**: This is how Hadoop runs on a real cluster.

In our installation (VM) the Hadoop installed as standalone. Therefore, this tutorial will show you how to run Hadoop jobs in Standalone mode (very useful for developing and debugging) and also in Pseudo-distributed mode (to mimic the behavior of a cluster environment).

Open a terminal and run the following two commands: (for more HDFS commands check HDFS tutorials)

Question

Why do we need to open a terminal? What it is simulate now?

To submit the original dataset to Hadoop HDFS you will use,

hadoop fs -put workspace/WordCount/pg100.txt

To run a Hadoop job (jar) with Hadoop command,

hadoop jar wordcount.jar edu.xjtlu.cse313.wordcount.WordCount *pg100.txt* <*your own output file directory*>

Here < *your own output file directory* > is an output directory to be created. Such as: “fangli12”

Please wait until it finishes running, it needs about 1 minute.

You can check Hadoop’s output by HDFS command,

hadoop fs -ls <*your own output file directory*>

You should see an output file for each reducer. Since there was only one reducer for this job, you should only see one part-\* file. Note that sometimes the files will be called part-NNNNN, and sometimes they’ll be called part-r-NNNNN. See Figure 21.

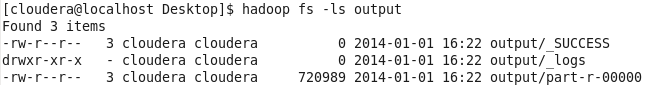


Figure 21. Run WordCount job.

You can check your results (contents of the output file) by Run the command:

hadoop fs -cat output/part*\*\* | head

You should see the same output as when you ran the job locally, as shown in Figure 22.

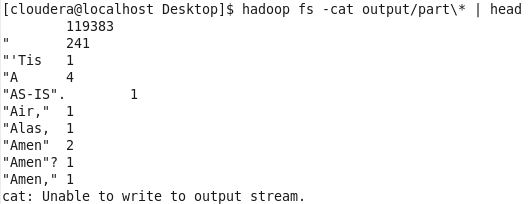


Figure 22. Run WordCount job.

Question

Where is the output file produced by Hadoop is stored now? Do you know its physical location?

## Explore Job running on Hadoop by Hadoop admin

To view the job’s logs, through Hadoop JobTracker (Hadoop job admin master which is replaced by YARN by Haddop 2)

Open the browser in the VM and point it to <http://localhost.localdomain:19888/jobhistory> as shown in Figure 23,

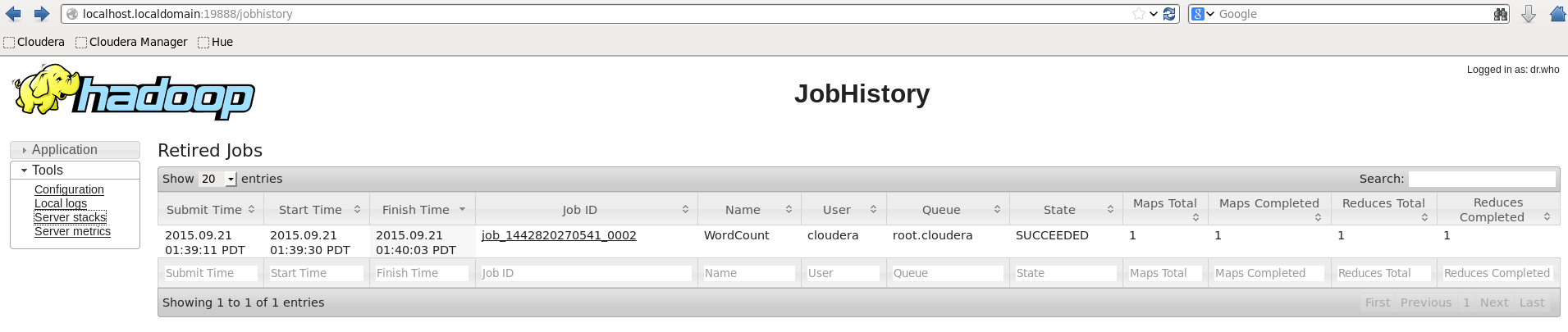


Figure 23. View WordCount job logs.

Click on the link for the “job ID”. See Figure 24,

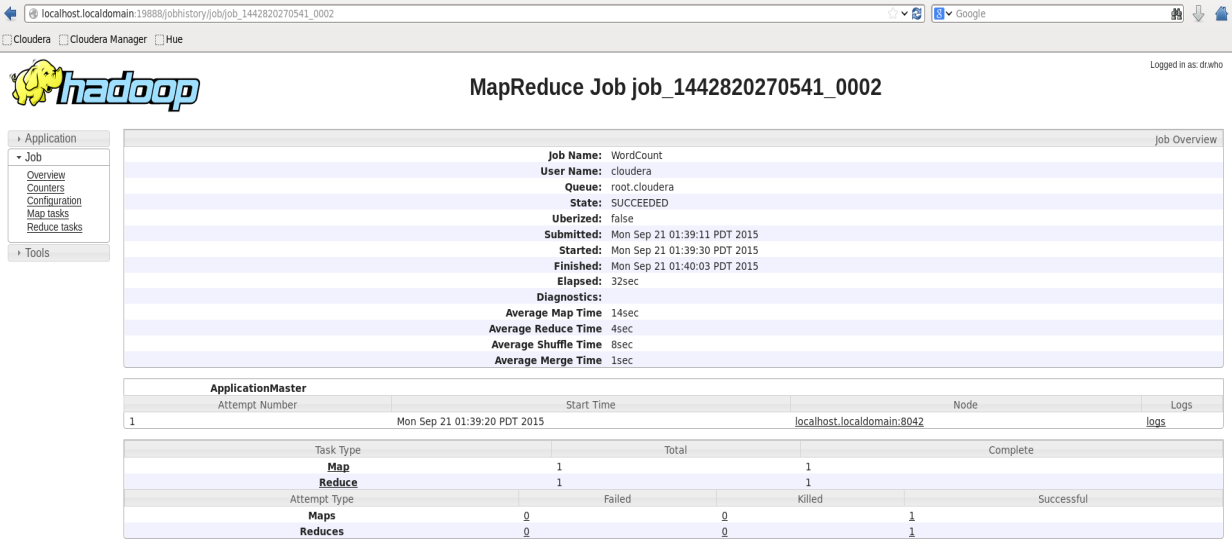


Figure 24. View WordCount job logs

• Click on the link for the Map under the “Task Type”. See Figure 25,

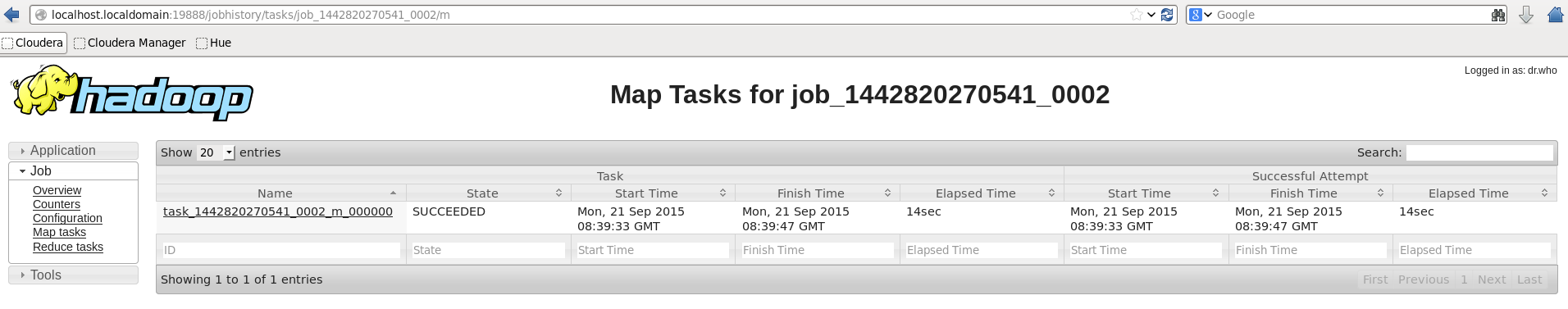
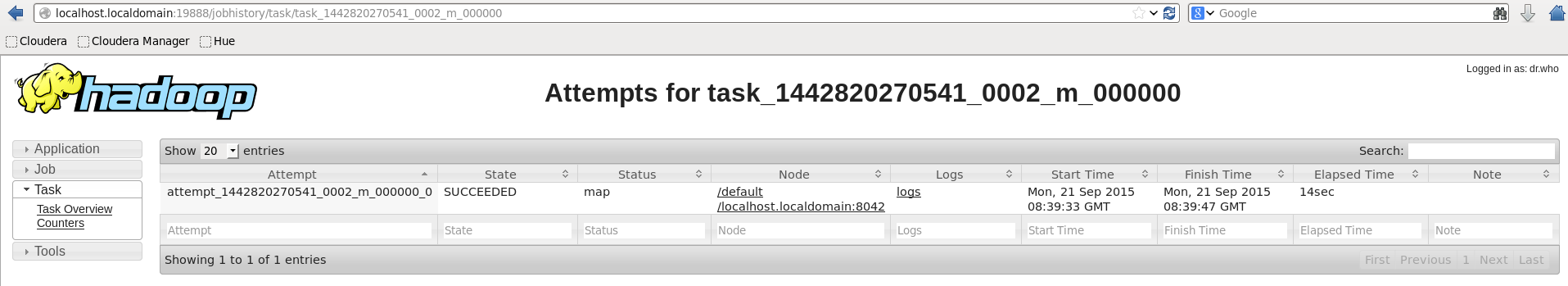


Figure 25. View Map

Click on the link under the “Name”. See Figure 26,

Figure 26. View Map Node

Click on the link for the Counters under the “Task”. See Figure 27,

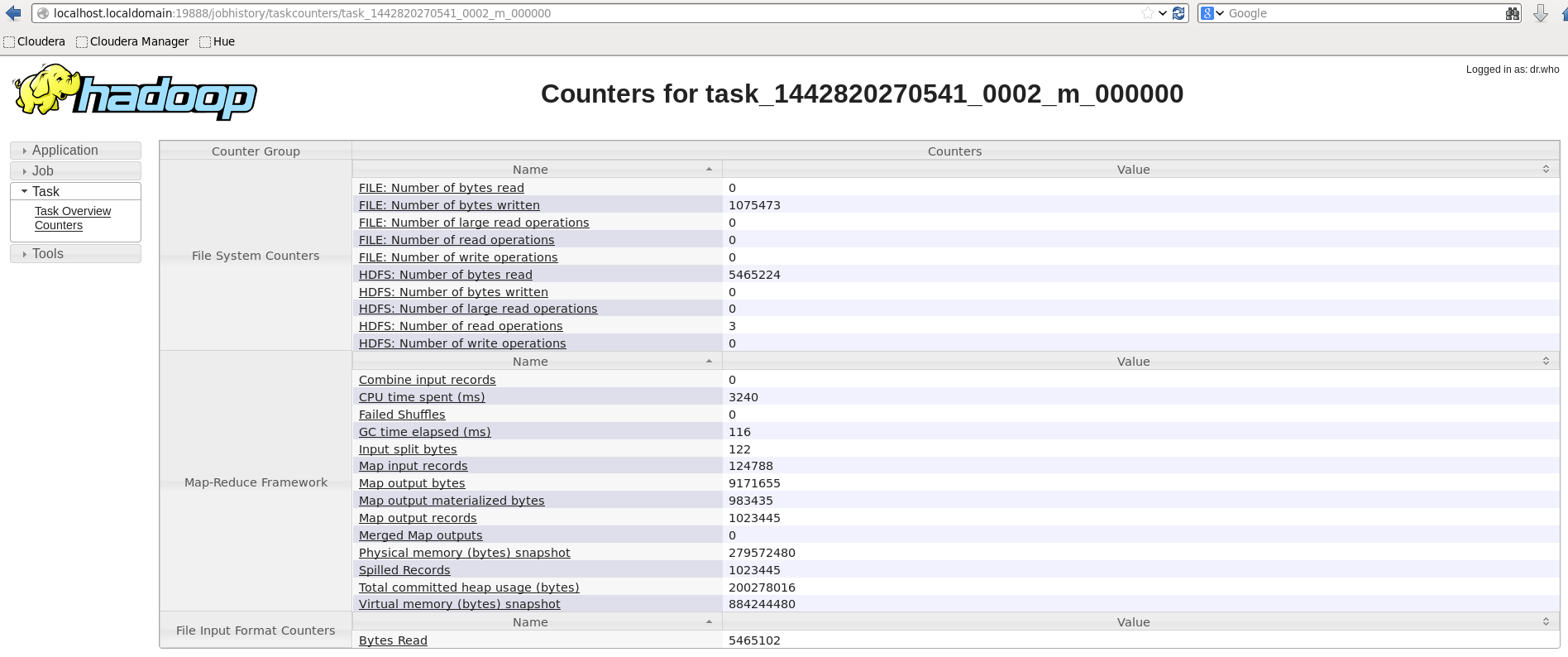


Figure 27. View Map Counters

Click on the link for the “Reduce Tasks” under the “Job” to observe reducer tasks execution details. See Figure 28,

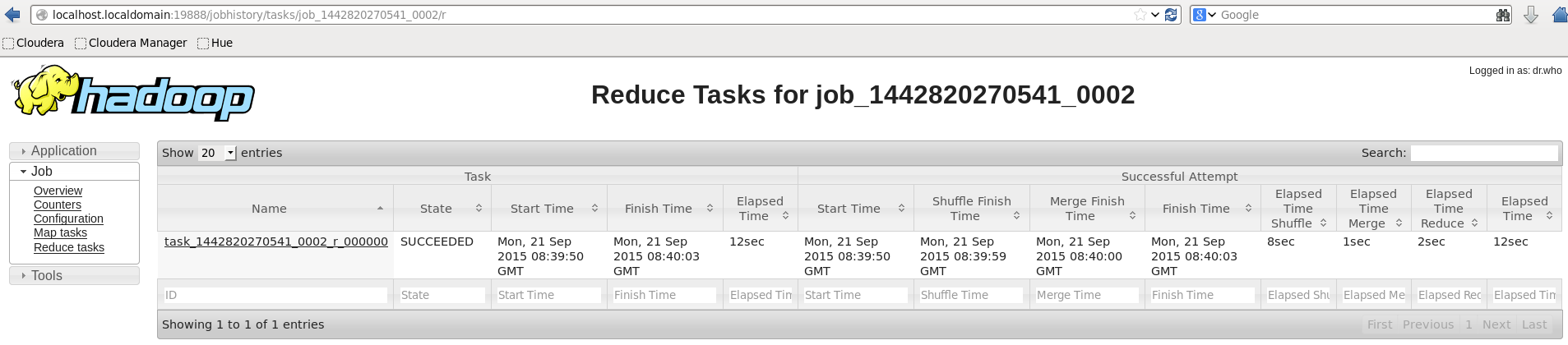
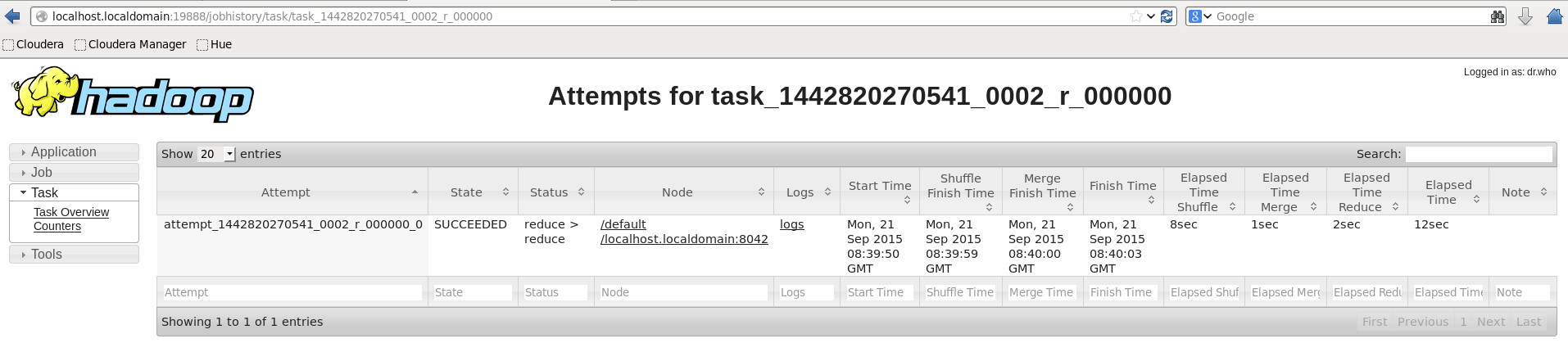
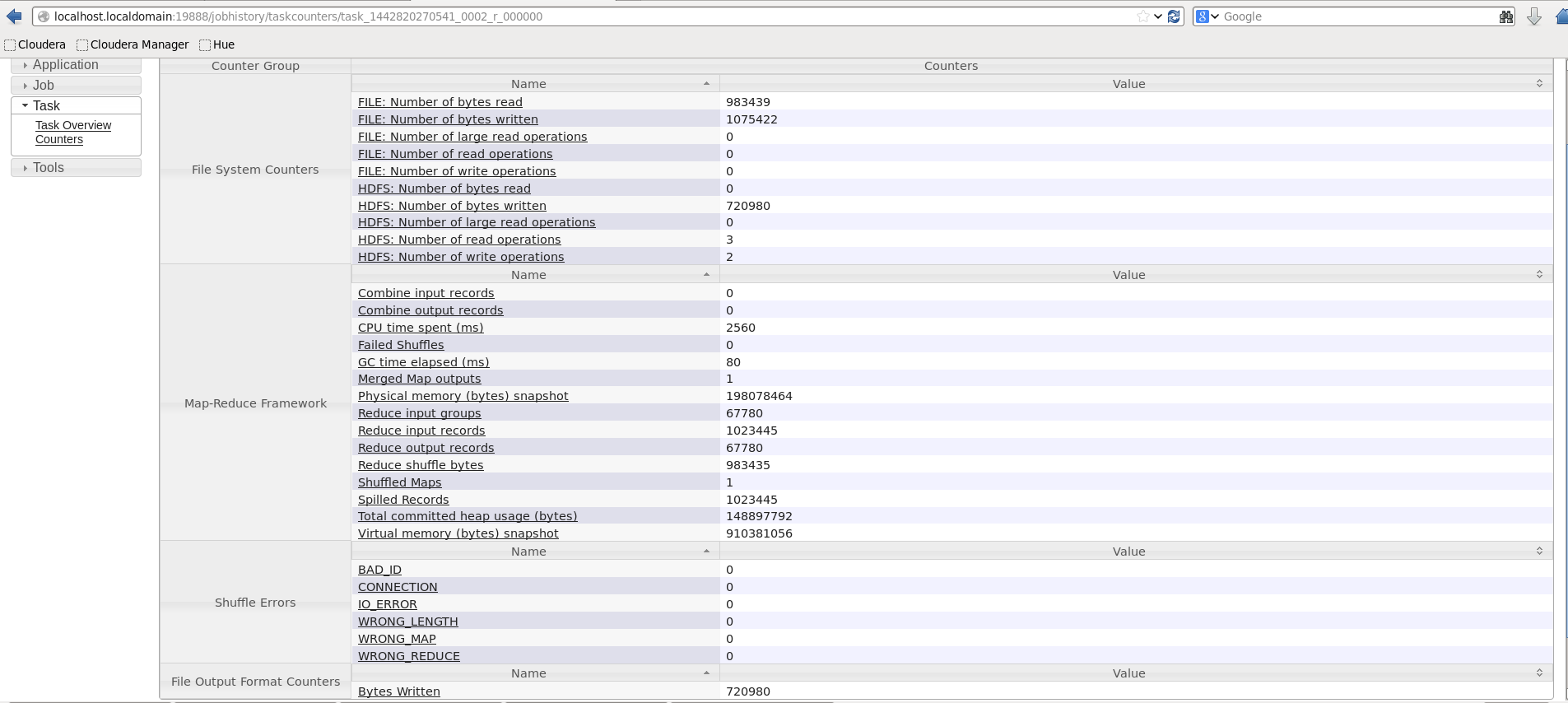


Figure 28. View Reduce

Click on the link under the “Name” to exam reducer details. See Figure 29.

Figure 29. View Reduce Node

Click on the link for the Counters under the “Task” to check up for counter details. See Figure 30.

Figure 30. View Reduce Counters

Question:

1. What is the purposes of exam Hadoop JobTracker? What do you find?

2. in the above example project. If I want you give me your analysis results, what you should do? If your result is in a file how to get it?

Now it is your turn. Please do the following project independently by refers to the tutorial and the sample project.

# IV Your lab two task:

Word Count with MapReduce in Hadoop

Due: 20th November, 2020 17:00pm

## Project

The purpose of this lab practice is to get you start using Hadoop. It provides a chance for you to experience how an analysis can be done on a computer cluster using advanced platform and programming model. You need to look at what we have done and simulate it to write, compile, debug MapReduce program (development phase) and execute a simple Hadoop Job on Cluster which deploys a Hadoop platform. You need writing MapReduce (Java) program in Eclipse and test/debug it, finally compile it to produce a Jar file. One you have a jar file for your job, you can submit your job to a Hadoop cluster (simulate) to get it run on the cluster and have your analysis results.

## Tasks:

You are given a datasets (**pg100.txt**, which is a book “THE SONNETS” by William Shakespeare), you are asked towrite your own MapReduce program to accomplish the following tasks:

* Output only the top 10 most common bigram (pair of adjacent words) in the given dataset, what are they and how many appearances?
* Output only the lines that contain the word ‘torture’.

## Submission

You should submit **a report** with **cover page** (download from ICE) in both hard copies (can be submitted either in class or in the submission box located on the 4th Flore of SD building by the lift) and e-copy (on ICE) .

Your submission template: (a single word document)

|  |
| --- |
| * + - 1. **My results files:** |
| File 1: top 10 bigram and occurrences （remove punctuations and simple count anything like Let’s, it’s, hasn’t…. as two words）  File 2: the single line contains “torture” |
| **2. My code with comments:** |
| package hadoop;  import java.util.\*;  import java.io.IOException;  import java.io.IOException;  import org.apache.hadoop.fs.Path;  import org.apache.hadoop.conf.\*;  import org.apache.hadoop.io.\*;  import org.apache.hadoop.mapred.\*;  import org.apache.hadoop.util.\*;  public class ProcessUnits {    **//Mapper class**  public static class E\_EMapper extends MapReduceBase implements  Mapper<LongWritable , /\*Input key Type \*/  Text, /\*Input value Type\*/  Text, /\*Output key Type\*/  IntWritable> /\*Output value Type\*/  {  **//Map function**  public void map(LongWritable key, Text value,  OutputCollector<Text, IntWritable> output,    Reporter reporter) throws IOException {  String line = value.toString();  String lasttoken = null;  StringTokenizer s = new StringTokenizer(line,"\t");  String year = s.nextToken();    while(s.hasMoreTokens()) {  lasttoken = s.nextToken();  }  int avgprice = Integer.parseInt(lasttoken);  output.collect(new Text(year), new IntWritable(avgprice));  }  }    //Reducer class  public static class E\_EReduce extends MapReduceBase implements Reducer< Text, IntWritable, Text, IntWritable > {    //Reduce function  public void reduce( Text key, Iterator <IntWritable> values,  OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {  int maxavg = 30;  int val = Integer.MIN\_VALUE;    while (values.hasNext()) {  if((val = values.next().get())>maxavg) {  output.collect(key, new IntWritable(val));  }  }  }  }  //Main function  public static void main(String args[])throws Exception {  JobConf conf = new JobConf(ProcessUnits.class);    conf.setJobName("max\_eletricityunits");  conf.setOutputKeyClass(Text.class);  conf.setOutputValueClass(IntWritable.class);  conf.setMapperClass(E\_EMapper.class);  conf.setCombinerClass(E\_EReduce.class);  conf.setReducerClass(E\_EReduce.class);  conf.setInputFormat(TextInputFormat.class);  conf.setOutputFormat(TextOutputFormat.class);    FileInputFormat.setInputPaths(conf, new Path(args[0]));  FileOutputFormat.setOutputPath(conf, new Path(args[1]));    JobClient.runJob(conf);  }  } |
| 1. Hadoop command list and order you have used to run your job and get your results |
| ….  $HADOOP\_HOME/bin/hadoop fs -cat output\_dir/part-00000/bin/hadoop dfs get output\_dir /home/hadoop |

# Appendix I. Virtual Machine

This virtual machine is provided by one of the most adopted Hadoop provider Cloudera. You can download the latest version from,

<https://www.cloudera.com/downloads/quickstart_vms/5-13.html> (select Virtual Box),

The web site has been through a major change recently. It is possible the virtual machine has been stopped but I am sure there would be an equivalent package. Explore by yourself. Otherwise you can download my pre-downloaded version. From:

BOX: <https://box.xjtlu.edu.cn/d/b5302d2a397146f0ba62/>

The virtual machine is in folder: cloudera-quickstart-vm-5.0.0-0-virtualbox

The virtual box has:

* CentOS 6.2 (Linux)
* JDK 6 (1.6.0 32) (JDK and JRE)
* Hadoop 2.3.0
* Eclipse 4.2.6 (Juno)

Create a new folder on your desktop, name it “INT303” and save your download into that folder, it should has two files:

* cloudera-quickstart-vm-5.0.0-0-virtualbox. vmdk
* cloudera-quickstart-vm-5.0.0-0-virtualbox.vbox

To use the virtual machine, you simply

**Double clicking file: “cloudera-quickstart-vm-5.0.0-0-virtualbox.vbox”**

Your virtual machine should now appear in the left column of Virtual Box. Select it and click on Start to launch it. You just need to click “ok” if any window pops out. It will spend few minutes to start the virtual machine, please be patient.

Once the VM is booted and up running, you are automatically logged in as the cloudera super user with a password as follows:

* username: cloudera
* password: cloudera

Now, you have Linux box, eclipse and JRE everything you need for Hadoop.

# Appendix II. Unix (Lunix) Commands

|  |  |  |
| --- | --- | --- |
| **Command** | **Example** | **Description** |
| pwd | pwd | Print current directory |
| ls | ls | Lists files in current directory List in long format |
| ls hadoop\* | ls hadoop\* | Shows all the file names starts with hadoop |
| cd | cd temp cd ..  cd ~dhyatt/web-docs | Change directory to tempdir  Move back one directory  Move into dhyatt's web-docs directory |
| mkdir | mkdir -p /myfolder/tmp | Make a directory called graphics |
| rm –rf | rm –rf mydir | Remove directory or file |
| cp | cp file1 web-docs  cp file1 file1.bak | Copy file into directory Make backup of file1 |
| cat | Cat file-name.txt | Display content of a file |
| kill -9 | Kill -9 pid | Kill the process |
| ps –ef |grep process | ps –ef |grep spark | Finds out a running process |
| chmod | chmod –R 777 /myfolder | Provides the permission for file |
| vi | vi file.txt - Open a file  i - Go into insert mode to write  ESC:wq – Save File  ESC:q! – Don’t save file | File editor command |

# Appendix III. Hadoop HDFS commands

Running ./bin/hadoop dfs with no additional arguments will list all the commands that can be run with the FsShell system. Furthermore, **$HADOOP\_HOME/bin/hadoop fs -help** commandName will display a short usage summary for the operation in question, if you are stuck.

A table of all the operations is shown below. The following conventions are used for parameters:

"<path>" means any file or directory name.

"<path>..." means one or more file or directory names.

"<file>" means any filename.

"<src>" and "<dest>" are path names in a directed operation.

"<localSrc>" and "<localDest>" are paths as above, but on the local file system.

All other files and path names refer to the objects inside HDFS.

|  |  |
| --- | --- |
| **Sr.No** | **Command & Description** |
| 1 | **-ls <path>**  Lists the contents of the directory specified by path, showing the names, permissions, owner, size and modification date for each entry. |
| 2 | **-lsr <path>**  Behaves like -ls, but recursively displays entries in all subdirectories of path. |
| 3 | **-du <path>**  Shows disk usage, in bytes, for all the files which match path; filenames are reported with the full HDFS protocol prefix. |
| 4 | **-dus <path>**  Like -du, but prints a summary of disk usage of all files/directories in the path. |
| 5 | **-mv <src><dest>**  Moves the file or directory indicated by src to dest, within HDFS. |
| 6 | **-cp <src> <dest>**  Copies the file or directory identified by src to dest, within HDFS. |
| 7 | **-rm <path>**  Removes the file or empty directory identified by path. |
| 8 | **-rmr <path>**  Removes the file or directory identified by path. Recursively deletes any child entries (i.e., files or subdirectories of path). |
| 9 | **-put <localSrc> <dest>**  Copies the file or directory from the local file system identified by localSrc to dest within the DFS. |
| 10 | **-copyFromLocal <localSrc> <dest>**  Identical to -put |
| 11 | **-moveFromLocal <localSrc> <dest>**  Copies the file or directory from the local file system identified by localSrc to dest within HDFS, and then deletes the local copy on success. |
| 12 | **-get [-crc] <src> <localDest>**  Copies the file or directory in HDFS identified by src to the local file system path identified by localDest. |
| 13 | **-getmerge <src> <localDest>**  Retrieves all files that match the path src in HDFS, and copies them to a single, merged file in the local file system identified by localDest. |
| 14 | **-cat <filen-ame>**  Displays the contents of filename on stdout. |
| 15 | **-copyToLocal <src> <localDest>**  Identical to -get |
| 16 | **-moveToLocal <src> <localDest>**  Works like -get, but deletes the HDFS copy on success. |
| 17 | **-mkdir <path>**  Creates a directory named path in HDFS.  Creates any parent directories in path that are missing (e.g., mkdir -p in Linux). |
| 18 | **-setrep [-R] [-w] rep <path>**  Sets the target replication factor for files identified by path to rep. (The actual replication factor will move toward the target over time) |
| 19 | **-touchz <path>**  Creates a file at path containing the current time as a timestamp. Fails if a file already exists at path, unless the file is already size 0. |
| 20 | **-test -[ezd] <path>**  Returns 1 if path exists; has zero length; or is a directory or 0 otherwise. |
| 21 | **-stat [format] <path>**  Prints information about path. Format is a string which accepts file size in blocks (%b), filename (%n), block size (%o), replication (%r), and modification date (%y, %Y). |
| 22 | **-tail [-f] <file2name>**  Shows the last 1KB of file on stdout. |
| 23 | **-chmod [-R] mode,mode,... <path>...**  Changes the file permissions associated with one or more objects identified by path.... Performs changes recursively with R. mode is a 3-digit octal mode, or {augo}+/-{rwxX}. Assumes if no scope is specified and does not apply an umask. |
| 24 | **-chown [-R] [owner][:[group]] <path>...**  Sets the owning user and/or group for files or directories identified by path.... Sets owner recursively if -R is specified. |
| 25 | **-chgrp [-R] group <path>...**  Sets the owning group for files or directories identified by path.... Sets group recursively if -R is specified. |
| 26 | **-help <cmd-name>**  Returns usage information for one of the commands listed above. You must omit the leading '-' character in cmd. |