Developing Hadoop Applications
Lab Guide

Overview:

– Developing Hadoop Applications
– Build Hadoop MapReduce Applications
– Manage and Test Hadoop MapReduce Applications
– Launch Jobs and Advanced Hadoop MapReduce Applications

Using This Guide

Overview of Labs

The table below lists the lab exercises included in this guide. Lab exercises are numbered to correspond to the learning goals in the course.

Lessons and Labs
Lesson 1: Introduction to MapReduce • Lab 1.2: Run `wordcount` • Lab 1.3: Examine job metrics in JobHistoryServer
Lesson 2: Job Execution Framework • Lab 2.3 – Run `DistributedShell` • Lab 2.4 – Examine job results
Lesson 3: Write MapReduce Programs • Lab 3.3 – Modify a MapReduce program
Lesson 4: Use the MapReduce API • Lab 4.3 – Write a MapReduce program
Lesson 5: Manage, Monitor, and Test MapReduce Jobs • Lab 5.1a – Examine default job output • Lab 5.1b – Use custom counters • Lab 5.3a – Use standard output, error, and logging • Lab 5.3b – Use the Hadoop CLI to manage jobs • Lab 5.4 – Test a MapReduce application
Lesson 6: Manage Performance • Lab 6.3 – De-tune a job and measure performance impact
Lesson 7: Working with Data • Lab 7.3 – Run a MapReduce program using HBase as source
Lesson 8: Launching Jobs • Lab 8.3 – Write a MapReduce driver to launch two jobs
Lesson 9: Streaming MapReduce • Lab 9.3 – Implement a MapReduce streaming application

Icons Used in This Guide

This lab guide uses the following icons to draw attention to different types of information:

Note: Additional information that will clarify something, provides details, or helps you avoid mistakes.

CAUTION: Details you **must** read to avoid potentially serious problems.

Q&A: A question posed to the learner during a lab exercise.

Try This! Exercises you can complete after class (or during class if you finish a lab early) to strengthen learning.

Command Syntax

When command syntax is presented, any arguments that are enclosed in chevrons, `<like this>`, should be substituted with an appropriate value. For example this:

```
# cp <source file> <destination file>
```

might be entered by the user as this:

```
# cp /etc/passwd /etc/passwd.bak
```

Note: Sample commands provide guidance, but do not always reflect exactly what you will see on the screen. For example, if there is output associated with a command, it may not be shown.

Caution: Code samples in this lab guide may not work correctly when cut and pasted. For best results, type commands in rather than cutting and pasting.

Start Hadoop Syntax

Important: First switch to user *hdoop* from the terminal before starting hadoop:

```
$ su - hdoop
```

Hadoop has been already installed. Run following scripts as *hdoop* user to start all the daemons:

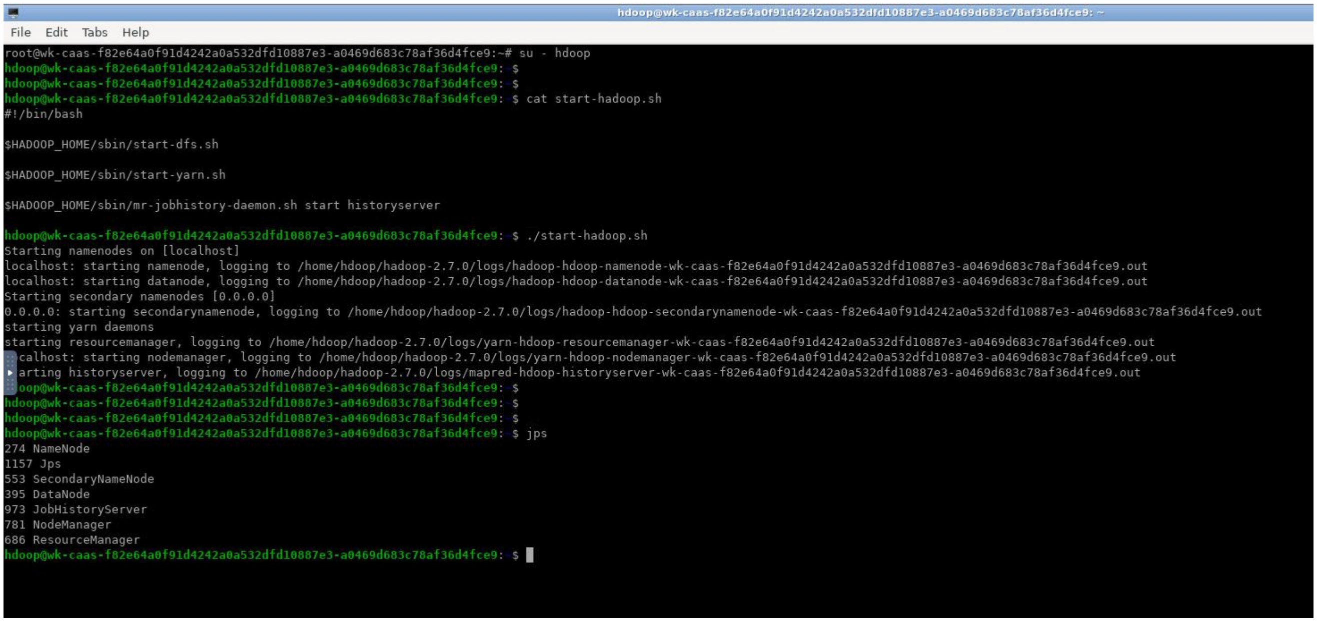
```
$ HADOOP_HOME/sbin/start-dfs.sh

$ HADOOP_HOME/sbin/start-yarn.sh

$ HADOOP_HOME/sbin/mr-jobhistory-daemon.sh start historyserver
```

Go to hadoop-2.7.0/sbin directory and start all the daemons.

To check that all the Hadoop services are up and running, run the below command.

Command: jps

Note: ./start-hadoop.sh has been created at /home/hdoop which will run all of the above scripts.


hdoop@wk-caas-f82e64a0f91d4242a0a532dfd10887e3-a0469d683c78af36d4fce9: ~
File
Edit
Tabs
Help

```
root@wk-caas-f82e64a0f91d4242a0a532dfd10887e3-a0469d683c78af36d4fce9:~# su - hdoop
hdoop@wk-caas-f82e64a0f91d4242a0a532dfd10887e3-a0469d683c78af36d4fce9:~$
hdoop@wk-caas-f82e64a0f91d4242a0a532dfd10887e3-a0469d683c78af36d4fce9:~$
hdoop@wk-caas-f82e64a0f91d4242a0a532dfd10887e3-a0469d683c78af36d4fce9:~$ cat start-hadoop.sh
#!/bin/bash

$HADOOP_HOME/sbin/start-dfs.sh

$HADOOP_HOME/sbin/start-yarn.sh

$HADOOP_HOME/sbin/mr-jobhistory-daemon.sh start historyserver

hdoop@wk-caas-f82e64a0f91d4242a0a532dfd10887e3-a0469d683c78af36d4fce9:~$ ./start-hadoop.sh
Starting namenodes on [localhost]
localhost: starting namenode, logging to /home/hdoop/hadoop-2.7.0/logs/hadoop-hdoop-namenode-wk-caas-f82e64a0f91d4242a0a532dfd10887e3-a0469d683c78af36d4fce9.out
localhost: starting datanode, logging to /home/hdoop/hadoop-2.7.0/logs/hadoop-hdoop-datanode-wk-caas-f82e64a0f91d4242a0a532dfd10887e3-a0469d683c78af36d4fce9.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /home/hdoop/hadoop-2.7.0/logs/hadoop-hdoop-secondarynamenode-wk-caas-f82e64a0f91d4242a0a532dfd10887e3-a0469d683c78af36d4fce9.out
starting yarn daemons
starting resourcemanager, logging to /home/hdoop/hadoop-2.7.0/logs/yarn-hdoop-resourcemanager-wk-caas-f82e64a0f91d4242a0a532dfd10887e3-a0469d683c78af36d4fce9.out
  calhost: starting nodemanager, logging to /home/hdoop/hadoop-2.7.0/logs/yarn-hdoop-nodemanager-wk-caas-f82e64a0f91d4242a0a532dfd10887e3-a0469d683c78af36d4fce9.out
  arting historyserver, logging to /home/hdoop/hadoop-2.7.0/logs/mapred-hdoop-historyserver-wk-caas-f82e64a0f91d4242a0a532dfd10887e3-a0469d683c78af36d4fce9.out
  oop@wk-caas-f82e64a0f91d4242a0a532dfd10887e3-a0469d683c78af36d4fce9:~$
hdoop@wk-caas-f82e64a0f91d4242a0a532dfd10887e3-a0469d683c78af36d4fce9:~$
hdoop@wk-caas-f82e64a0f91d4242a0a532dfd10887e3-a0469d683c78af36d4fce9:~$
hdoop@wk-caas-f82e64a0f91d4242a0a532dfd10887e3-a0469d683c78af36d4fce9:~$ jps
274 NameNode
1157 Jps
553 SecondaryNameNode
395 DataNode
973 JobHistoryServer
781 NodeManager
686 ResourceManager
hdoop@wk-caas-f82e64a0f91d4242a0a532dfd10887e3-a0469d683c78af36d4fce9:~$
```

MapReduce Essentials

Lesson 1: Introduction to MapReduce

Lab Overview

In the lesson’s lab exercises, you will run a few MapReduce jobs from the command line and examine job information in the MapR Control System (MCS).

> **Note:** Some commands shown throughout this lab guide are too long to fit on a single line. The backslash character (`\`) indicates that the command continues on the next line. Do not include the backslash character, or a carriage return, when typing the commands.

Lab 1.2: Run `wordcount`

Estimated time to complete: 30 minutes

Run `wordcount` against a text file

1. Log into the cluster as the user `hdoop`.
2. Create a directory in your home directory as follows:

   ```
   $ cd ~ && mkdir input
   ```

3. Create a file in the directory as follows:

   ```
   $ echo "Hello world! Hello" > input/in.txt
   $ ls -l input
   ```

   > **Note**: Run following command first to put text file in hadoop filesystem:
   >
   > ```
   > $ hadoop fs -mkdir -p input
   > $ hadoop fs -put ./input/in.txt input
   > ```

   `Run the wordcount application against the input file.`

   ```
   $ hadoop jar /home/hdoop/hadoop-2.7.0/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.0.jar wordcount input/in.txt output
   ```

4. Check the output of the `wordcount` application:

```
$ hadoop fs -copyToLocal output .
$ ls -ltr output
$ cat output/*
```

Run wordcount against a set of files

1. Create a directory in your home directory, and also create a set of files as follows:

```
$ mkdir -p /home/hdoop/Lab1.3/IN2
$ cd /home/hdoop/Lab1.3
$ cp /etc/*.conf /home/hdoop/Lab1.3/IN2 2>/dev/null
```

Note: Run following command first to put all files in hadoop filesystem:

```
$ hadoop fs -mkdir -p IN2
$ hadoop fs -put /home/hdoop/Lab1.3/IN2/*.conf IN2
```

2. Determine how many files are in that directory:

```
$ ls /home/hdoop/Lab1.3/IN2 | wc -l
```

3. Run the `wordcount` application against the directory:

```
$ hadoop jar /home/hdoop/hadoop-2.7.0/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.0.jar wordcount IN2 OUT2
```

Check the output of the wordcount application:

```
$ hadoop fs -copyToLocal OUT2 /home/hdoop/Lab1.3/OUT2
$ wc -l /home/hdoop/Lab1.3/OUT2/part-r-00000
$ more  /home/hdoop/Lab1.3/OUT2/part-r-00000
```

Run wordcount Against a Binary File

1. Create a directory in your home directory as follows:

```
$ mkdir -p /home/hdoop/Lab1.3/IN3
```

2. Create a binary file in that directory as follows:

```
$ cp /bin/cp /home/hdoop/Lab1.3/IN3/mybinary
```

3. Verify the file is a binary:

```
$ file /home/hdoop/Lab1.3/IN3/mybinary
```

4. See if there is any readable text in the binary:

```
$ strings /home/hdoop/Lab1.3/IN3/mybinary | more
```

Note: Run following command first to put binary file in hadoop filesystem:

```
$ hadoop fs -mkdir -p IN3
$ hadoop fs -put /home/hdoop/Lab1.3/IN3/mybinary IN3
```

1. Run the `wordcount` application against the input file.

   ```
   $ hadoop jar /home/hdoop/hadoop-2.7.0/share/hadoop/mapreduce/hadoop-
   mapreduce-examples-2.7.0.jar wordcount IN3/mybinary \
   /home/hdoop/Lab1.3/OUT3
   ```

   Note: Run following command first to downloads results to local filesytem from hadoop filesystem:

   ```
   $ hadoop fs -copyToLocal /home/hdoop/Lab1.3/OUT3
   /home/hdoop/Lab1.3/OUT3
   ```

2. Check the output of the `wordcount` application:

   ```
   $ more /home/hdoop/Lab1.3/OUT3/part-r-00000
   ```

3. Cross-reference the frequency of the “word” `ATUH` in the binary and in the `wordcount` output:

   ```
   $ strings /home/hdoop/Lab1.3/IN3/mybinary |  grep -c ATUH
   $ egrep -ac ATUH /home/hdoop/Lab1.3/OUT3/part-r-00000
   ```

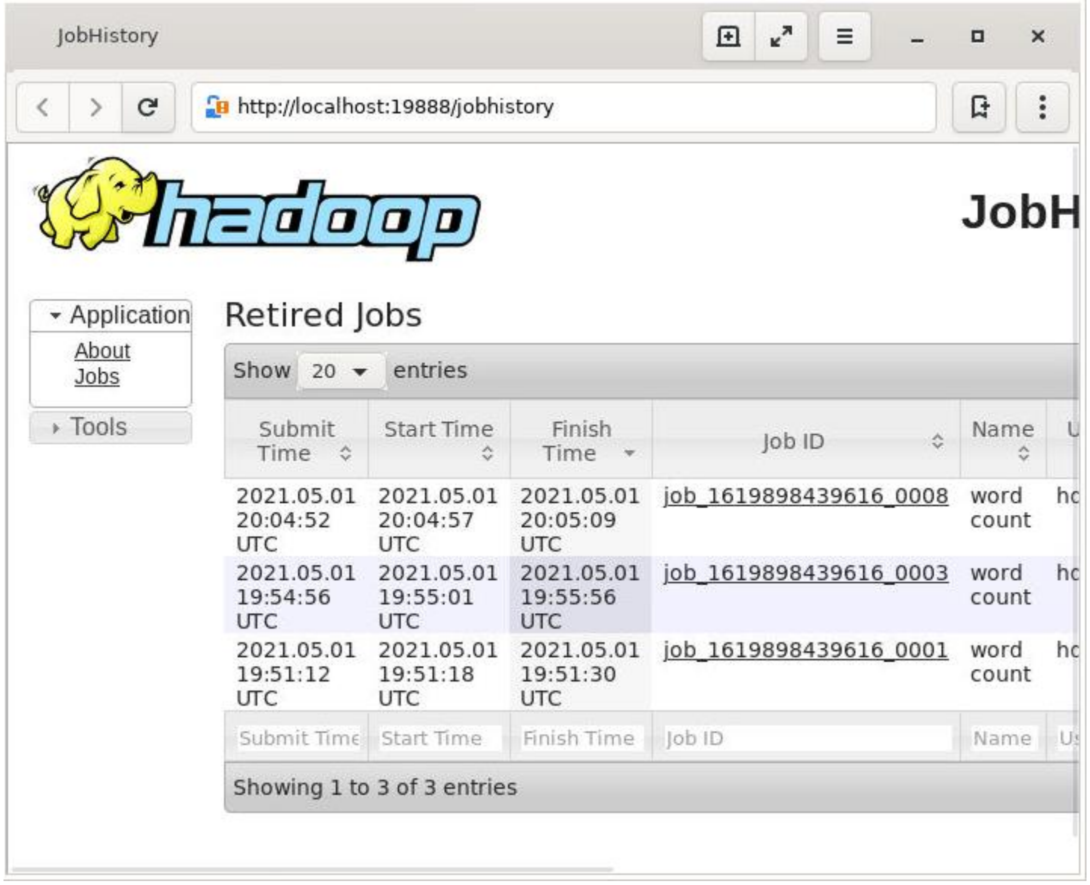
Lab 1.3: Examine Job Metrics in JobHistoryServer

Estimated time to complete: 30 minutes

1. Connect to the JobHistoryServer in your Web browser. Launch Midori Browser by clicking icon on the desktop :

   ```
   https://<IP address>:19888
   ```

2. The JobHistory page displays:



Retired Jobs

Submit Time	Start Time	Finish Time	Job ID	Name	U
2021.05.01 20:04:52 UTC	2021.05.01 20:04:57 UTC	2021.05.01 20:05:09 UTC	job_1619898439616_0008	word count	hc
2021.05.01 19:54:56 UTC	2021.05.01 19:55:01 UTC	2021.05.01 19:55:56 UTC	job_1619898439616_0003	word count	hc
2021.05.01 19:51:12 UTC	2021.05.01 19:51:18 UTC	2021.05.01 19:51:30 UTC	job_1619898439616_0001	word count	hc
Submit Time	Start Time	Finish Time	Job ID	Name	Us

Showing 1 to 3 of 3 entries

3. In the list of displayed jobs, scroll to the bottom to find your jobs (as a combination of the **Job Name** and **User** fields).

4. Click the link **word count** for one of the jobs you launched.

   a. How many tasks comprised that job?

   b. How long did they each last?

   c. On which node did they run? Note that in a single-node cluster, there's only one machine the job can run on.

Build Hadoop MapReduce Applications

Lesson 2: Job Execution Framework

Lab 2.3: Run `DistributedShell`

Estimated time to complete: 20 minutes

Run `DistributedShell` with a shell command

1. Launch the YARN job using the `yarn` command:

```
$ yarn jar /home/hdoop/hadoop-2.7.0/share/hadoop/yarn/hadoop-
yarn-applications-distributedshell-2.7.0.jar \
-shell_command /bin/ls \
-shell_args /home/hdoop \
-jar /home/hdoop/hadoop-2.7.0/share/hadoop/yarn/hadoop-yarn-
applications-distributedshell-2.7.0.jar
```

```
When the job completes, scroll back through the output to
determine your container ID for the  shell, as shown in the
sample output below:
```

```
15/01/21 18:34:07 INFO distributedshell.Client: Got application
report from ASM for, appId=1, clientToAMToken=null, appDiagnostics=,
appMasterHost=yarn-training/192.168.56.102,   appQueue=root.hdoop,
appMasterRpcPort=-1,   appStartTime=1421894036331,
yarnAppState=FINISHED,   distributedFinalState=SUCCEEDED,
appTrackingUrl=http://yarn-
training:8088/proxy/application_1421893926516_0001/A,    appUser=hdoop
```

Check Standard Output and Standard Error for Job

1. Change directory to the output directory for YARN jobs:

```
$ cd $HADOOP_HOME/logs/userlogs
```

2. List the contents of the directory:

```
$ ls
```

3. Change directory to your application output directory:

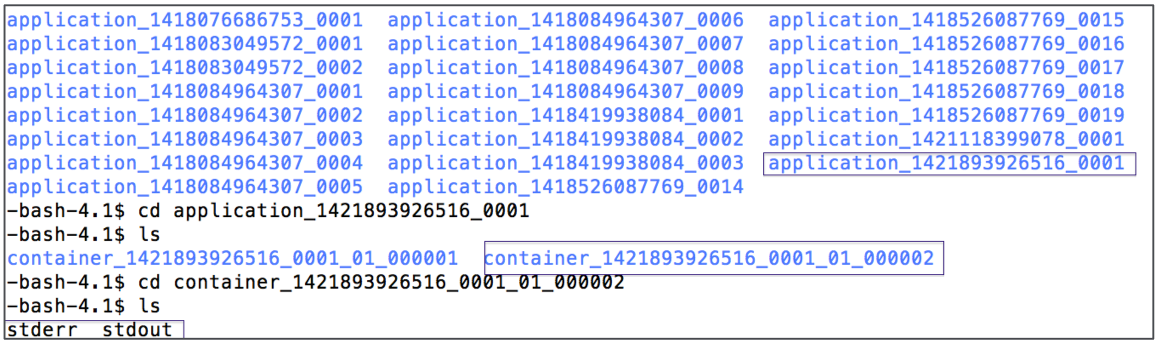
```
$ cd application_<timestamp>_<appid>
```

4. List the contents of the directory:

```
$ ls
```

5. Change directory to the second container output directory:

```
$ cd container_<timestamp>_<appid>_01_000002
```



```
application_1418076686753_0001  application_1418084964307_0006  application_1418526087769_0015
application_1418083049572_0001  application_1418084964307_0007  application_1418526087769_0016
application_1418083049572_0002  application_1418084964307_0008  application_1418526087769_0017
application_1418084964307_0001  application_1418084964307_0009  application_1418526087769_0018
application_1418084964307_0002  application_1418419938084_0001  application_1418526087769_0019
application_1418084964307_0003  application_1418419938084_0002  application_1421118399078_0001
application_1418084964307_0004  application_1418419938084_0003  application_1421893926516_0001
application_1418084964307_0005  application_1418526087769_0014
-bash-4.1$ cd application_1421893926516_0001
-bash-4.1$ ls
container_1421893926516_0001_01_000001  container_1421893926516_0001_01_000002
-bash-4.1$ cd container_1421893926516_0001_01_000002
-bash-4.1$ ls
stderr  stdout
```

6. Display the contents of the `stdout` file. You should see a listing of the /home/hdoop directory.

```
$ cat stdout
```

7. Display the contents of the `stderr` file. It should be empty.

```
$ cat stderr
```

Lab 2.4: Examine Job Results

Estimated time to complete: 15 minutes

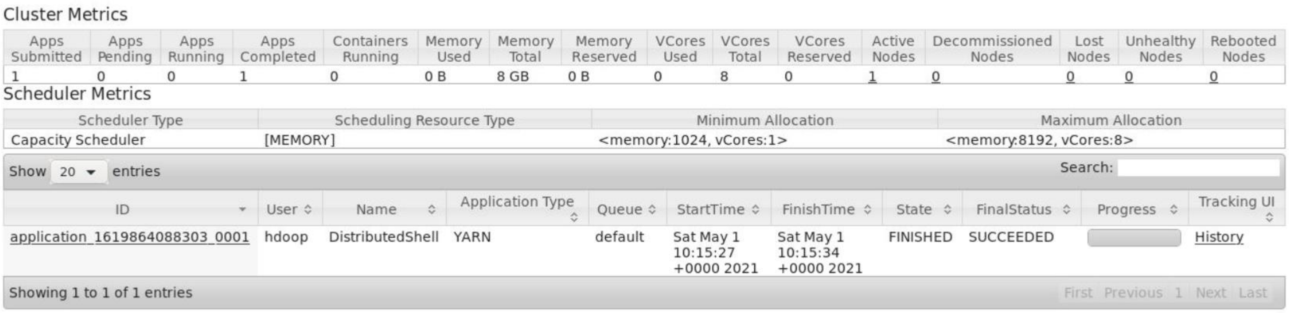
In this exercise, you will use the Web UI provided by the History Server to examine information for the job you previously launched.

1. Connect to the History Server in your web browser:

```
http://localhost:8088
http://<IP address>:8088
```

2. Scroll through the applications to find your application ID.

3. Click the link associated with your YARN job. How long did the job take?



Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
1	0	0	1	0	0 B	8 GB	0 B	0	8	0	1	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:8>

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1619864088303_0001	hdoop	DistributedShell	YARN	default	Sat May 1 10:15:27 +0000 2021	Sat May 1 10:15:34 +0000 2021	FINISHED	SUCCEEDED		History

Showing 1 to 1 of 1 entries

Note: Job output will also print appTrackingUrl which can be used to directly access job history

21/05/01 10:15:28 INFO distributedshell.Client: Got application report from ASM for, appId=1, clientToAMToken=null, appDiagnostics=, appMasterHost=N/A, appQueue=default, appMasterRpcPort=-1, appStartTime=1619864127194, yarnAppState=ACCEPTED, distributedFinalState=UNDEFINED, **appTrackingUrl**=http://wk-caas-f82e64a0f91d4242a0a532dfd10887e3-32e66f4f9a589a508f61bc:8088/proxy/application_1619864088303_0001/, appUser=hdoop

Lesson 3: Write MapReduce Programs

Lab Overview

The lab for this lesson covers how to make some modifications to an existing MapReduce program, compile it, run it, and examine the output. The existing code calculates minimum and maximum values in the data set. You will modify the code to calculate the mean surplus or deficit.

The data set we're using is the history of the United States federal budget from the year 1901 to 2012. The data was downloaded from the white house website and has been massaged for this exercise. The existing code calculates minimum and maximum values in the data set. You will modify the code to calculate the mean surplus or deficit.

Here is a sample record from the data set:

```
1968 152973 178134 -25161 128056 155798 -27742 24917 22336 2581
```

The fields of interest in this exercise are the first and fourth fields (year and surplus or deficit). The second field is the total income derived from federal income taxes, and the third field is the expenditures for that year. The fourth field is the difference between the second and third fields. A negative value in the fourth field indicates a budget deficit and a positive value indicates a budget surplus.

Lab 3.3: Modify a MapReduce Program

Estimated time to complete: 60 minutes

Copy the Lab Files

1. Log into the terminal as `hdoop`.
2. Create a directory for the lab work, and position yourself in that directory:

   ```
   $ su - hdoop
   $ mkdir /home/hdoop/Lab3
   $ cd /home/hdoop/Lab3
   ```

3. Copy the source code for the lab from Desktop (run as root user):

   ```
   $ cp -r /root/Desktop/apache-hadoop/Lab3/* /home/hdoop/Lab3
   $ chown -R hdoop /home/hdoop/Lab3
   ```

   This will create two directories: `RECEIPTS_LAB`, which contains the source files for the lab, and `RECEIPTS_SOLUTION` which contains files with the solution correctly implemented. You can review solutions files as needed for help completing the lab.

Modify Code in the Driver

1. Change directory into the `RECEIPTS_LAB` directory.

   ```
   $ cd RECEIPTS_LAB
   ```

2. Open the `ReceiptsDriver.java` source file with your favorite text editor.

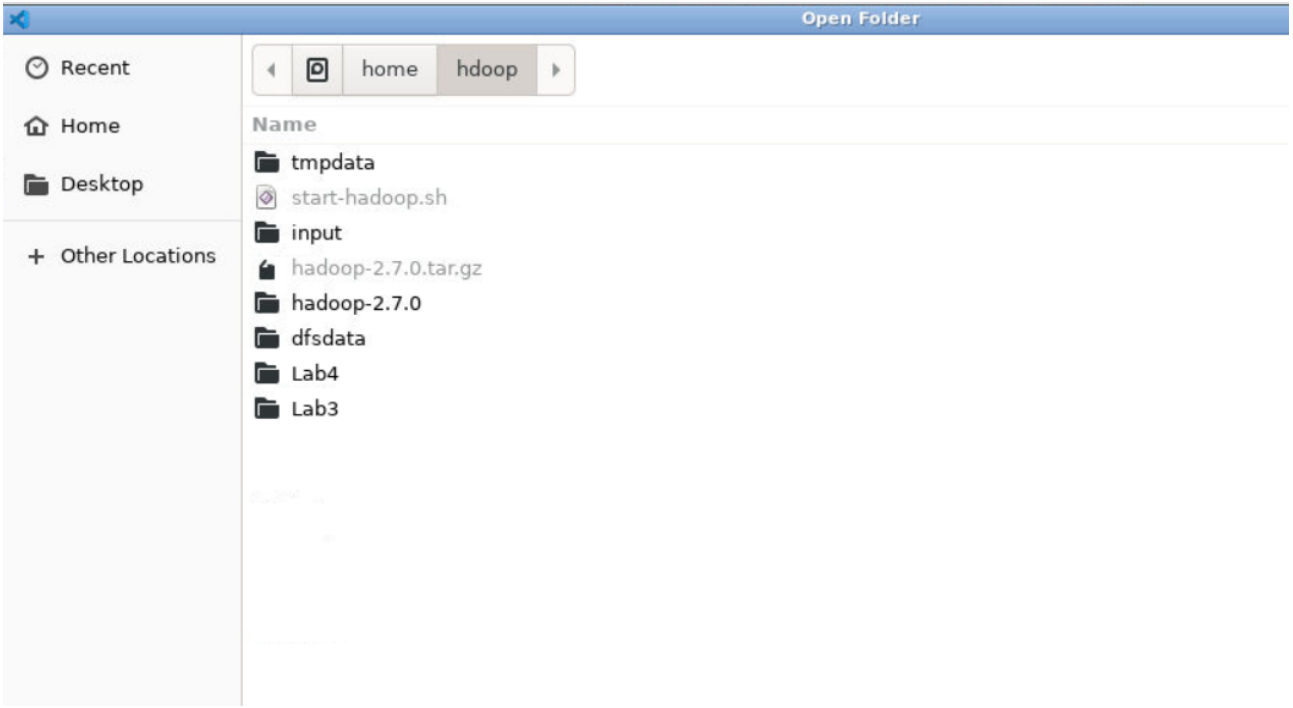
   ```
   $ vi ReceiptsDriver.java
   ```

3. Look for the string `// TODO` in the file, and follow the instructions to make the necessary changes.

4. Save the `ReceiptsDriver.java` file.

Note: You can also open files in vscode. Open vscode and click “File” > “Open Folder”.

Other Locations > Computer > home > hdoop > folderName



Compile and Run the MapReduce Program

1. Execute the `rebuild.sh` script to compile your code.

   ```
   $ ./rebuild.sh
   ```

2. Execute the `rerun.sh` script to run your code.

   ```
   $ ./rerun.sh
   ```

3. Examine the output from your MapReduce job. Note you may need to wait a minute before the job output is completely written to the output files. Update path to “RECEIPTS_SOLUTION” if you are running solution.

   ```
   $ cat /home/hdoop/Lab3/RECEIPTS_LAB/OUT/part*
   ```

Here is partial output expected for this exercise:

```
summary  1901_63
summary  1902_77
summary  1903_45
summary  1904_-43
summary  1905_-23
summary  1906_25
summary  1907_87
summary  1908_-57
summary  1909_-89
summary  1910_-18
summary  1911_11
```

Implement Code in the Reducer

In this exercise, you will implement code in the reducer to calculate the mean value. The code has already been provided to calculate minimum and maximum values.

Recall that the mapper code you ran above will produce intermediate results. One such record looks like this:

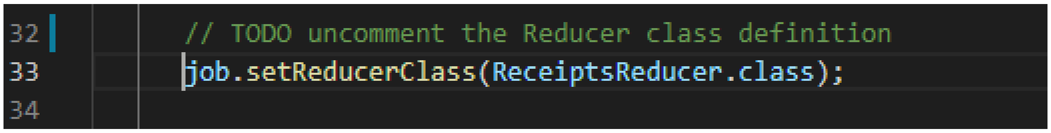
```
summary 1968_-25161
```

When you execute the code for this lab, there will only be one reducer (since there is only one key – “summary”). That reducer will iterate over all the intermediate results and pull out the year and surplus or deficit. Your reducer will keep track of the minimum and maximum values (as temp variables) as well as the year those values occurred. You will also need to keep track of the sum of the surplus or deficit and count of the records in order to calculate the mean value.

1. Open the `ReceiptsReducer.java` source file with your favorite text editor.

   ```
   $ vi ReceiptsReducer.java
   ```

2. Find the `// TODO` statements in the file, and make the changes indicated. Refer to the solutions file as needed for help.
3. Save the `ReceiptsReducer.java` file.
4. Open the `ReceiptsDriver.java` source file with your favorite text editor. Find the line `// TODO comment out the Reducer class definition`. Recall that in the previous section, you commented out the Reducer definition – **in this section, you will need to uncomment it so it will be included.**

   ```
   // TODO uncomment the Reducer class definition
   job.setReducerClass(ReceiptsReducer.class);
   ```

5. Save the `ReceiptsDriver.java` file.

Compile and Run Your Code

1. Execute the `rebuild.sh` script to compile your code.

   ```
   $ chmod +x *.sh
   $ ./rebuild.sh
   ```

2. Execute the `rerun.sh` script to run your code.

   ```
   $ ./rerun.sh
   ```

3. Examine the output from your MapReduce job.

   ```
   $ cat /home/hdoop/Lab3/RECEIPTS_LAB/OUT/part*
   ```

Here is the output expected for this exercise:

```
min(2009):   -1412688.0

max(2000):   236241.0

mean: -93862.0
```

Manage and Test Hadoop MapReduce Applications

Lesson 4: Use the MapReduce API

Lab Overview

The objective of this lab is to write your first complete MapReduce program using the numerical summary pattern we've been focusing on. The lab provides generic templates for the map, reduce, and driver classes. This exercise guides you through how to calculate the minimum, maximum, and mean SAT verbal and math scores over the whole data set.

Summary of Data

This lab examines data sampled from university students across North America. The data set can be downloaded from http://archive.ics.uci.edu/ml/datasets/University.

Not every record contains the same number of fields, but every record starts with the string `(def-instance` and ends with the string `))`. Each record contains information for a single university in the survey. Here is a sample record:

```
(def-instance Adelphi
    (state newyork)
    (control private)
    (no-of-students thous:5-10)
    (male:female ratio:30:70)
    (student:faculty ratio:15:1)
    (sat math 475)
    (expenses thous$:7-10)
    (percent-financial-aid 60)
    (no-applicants thous:4-7)
    (percent-admittance 70)
    (percent-enrolled 40)
    (academics scale:1-5 2)
    (social scale:1-5 2)
    (sat verbal 500)
    (quality-of-life scale:1-5 2)
    (academic-emphasis  business-administration)
    (academic-emphasis biology))
```

Lab 4.3: Write a MapReduce Program

Estimated time to complete: 90 minutes

Prepare

1. Log into the terminal as hdoop.
2. Create a directory as follows:

```
$ su - hdoop
$ mkdir /home/hdoop/Lab4
$ cd /home/hdoop/Lab4
```

3. Copy the source code for the lab from Desktop (run as root user in other terminal):

```
$ cp -r /root/Desktop/apache-hadoop/Lab4/* /home/hdoop/Lab4
$ chown -R hdoop /home/hdoop/Lab4
```

This will create two subdirectories: a UNIVERSITY_LAB directory containing the source files, and a UNIVERSITY_SOLUTION directory that contains the modified files with the correct solutions. You can refer to the files in the UNIVERSITY_SOLUTION directory if you get stuck on a step.

Implement the Mapper Class

1. Change directory into the UNIVERSITY_LAB directory.

```
$ cd UNIVERSITY_LAB
```

2. Open the data file with your favorite text editor.

```
$ vi DATA/university.txt
```

Each record contains an unknown number of fields after the start of the record and before either the sat math or sat verbal field. The sat math field may come before or after the sat verbal field, and one or both of the fields may not be part of the record at all. For example:

```
(def-instance <University Name>
. . .
(sat verbal 500)
. . .
(sat math 475)
. . .))
```

Examine the first few records in the file, then skip to line 1000 or so. Note that the data set is not uniform from beginning to end.

3. Close the data file, and open the UniversityMapper.java source file with your favorite text editor.

```
$ vi UniversityMapper.java
```

4. The `UniversityMapper.java` file contains a number of `TODO` directives. Make the changes necessary to address each `TODO` entry, and then save the file. Compare your results to what is shown in the file in the `UNIVERSITY_SOLUTIONS` directory.

Implement the Reducer Class

Recall that one reducer will be given a list of key-value pairs that looks like this:

```
satv 480 500 530 . . .
```

The other reducer will be given a list of key-value pairs that looks like this:

```
satm 400 500 510 . . .
```

Perform the following steps to complete this lab:

1. Open the `UniversityReducer.java` source file with your favorite text editor.

   ```
   $ vi UniversityReducer.java
   ```

2. Implement each `TODO` in the `UniversityReducer.java` file as follows, just as you did for the `UniversityMapper.java` file. Save your changes. Compare your changes to the file in the `UNIVERSITY_SOLUTIONS` directory.

Implement the Driver Class

1. Open the `UniversityDriver.java` source file with your favorite text editor.

   ```
   $ vi UniversityDriver.java
   ```

2. Implement each `TODO` in the `UniversityDriver.java` file, and save your changes. Compare your changes to the file in the `UNIVERSITY_SOLUTIONS` directory.

Compile and Run the MapReduce Program

1. Launch the rebuild.sh script to recompile the source code:

   ```
   $ chmod +x *.sh
   $ ./rebuild.sh
   ```

   If you get any errors that you can't resolve, it might help to check the output from your map phase by setting `mapred.num.reduce.tasks` to `0` in your configuration.

2. Launch the `rerun.sh` script to execute the code.

   ```
   $ ./rerun.sh
   ```

3. Check the output of the program:

   ```
   $ cat OUT/part-r-00000
   ```

Lesson 5: Manage, Monitor, and Test MapReduce Jobs

Lab 5.1a: Examine Default Job Output

Estimated time to complete: 40 minutes

In this exercise, you will run the `teragen` and `terasort` MapReduce applications from the examples provided in the Hadoop distribution. You will then examine the records produced from running each one.

Prepare the lab files

1. Create a directory for the lab files, and position yourself in that directory:

```
$ su - hdoop
$ mkdir /home/hdoop/Lab5
$ cd /home/hdoop/Lab5
```

2. Download and unzip the lab files into that directory, and position yourself in the directory created:

```
$ cp -r /root/Desktop/apache-hadoop/Lab5/* /home/hdoop/Lab5
$ chown -R hdoop /home/hdoop/Lab5
```

   You should see three directories created when the lab file is unzipped: `SLOW_LAB`, `VOTER_LAB`, and `VOTER_SOLUTION`.

3. Uncompress the data file for the `VOTER_LAB`:

```
$ gunzip VOTER_LAB/DATA/myvoter.csv.gz
```

4. Inject some faulty records into your data set. For example:

```
$ echo "0,anna,14,independent,100,100" >> VOTER_LAB/DATA/myvoter.csv

$ echo "0,anna,25" >> VOTER_LAB/DATA/myvoter.csv
```

Note: Add faulty records in VOTER_SOLUTION if you are running solution.

```
$ echo "0,anna,14,independent,100,100" >> VOTER_SOLUTION/DATA/myvoter.csv

$ echo "0,anna,25" >> VOTER_SOLUTION/DATA/myvoter.csv
```

Run `teragen`

1. Run the `teragen` MapReduce application to generate 1000 records:

```
$ hadoop jar /home/hdoop/hadoop-2.7.0/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.0.jar teragen 1000 /home/hdoop/Lab5/TERA_IN
```

2. Look at the `teragen` job output counters.

Q: Why are there no input or output records for the reducer in the job output?

A: The `teragen` application is a map-only application.

Note: Run following command to download folder to local filesytem from hadoop.

```
hadoop fs -copyToLocal /home/hdoop/Lab5/TERA_IN /home/hdoop/Lab5/TERA_IN
```

3. Examine the files produced by `teragen` and answer the questions below.

   a. What type of file is produced?

   ```
   $ file /home/hdoop/Lab5/TERA_IN/part-m-0000*
   ```

   b. Why is the number of records we generated with `teragen` different than the total number of lines in the files?

   ```
   $ wc -l /home/hdoop/Lab5/TERA_IN/part-m-0000*
   ```

   c. Can you make sense out of the files by looking at them?

   ```
   $ view /home/hdoop/Lab5/TERA_IN/part-m-00000
   ```

Run `terasort`

1. Run the `terasort` application to sort those records you just created and look at the job output.

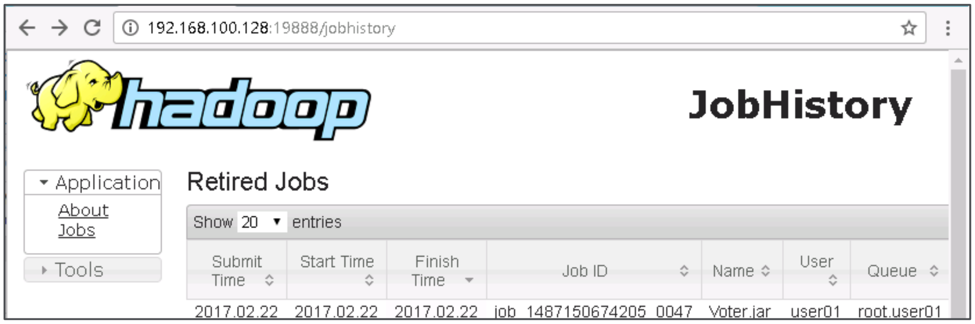
   ```
   $ hadoop jar /home/hdoop/hadoop-2.7.0/share/hadoop/mapreduce/hadoop-
   mapreduce-examples-2.7.0.jar terasort /home/hdoop/Lab5/TERA_IN
   /home/hdoop/Lab5/TERA_OUT
   ```

   ```
   $ hadoop fs -copyToLocal /home/hdoop/Lab5/TERA_OUT /home/hdoop/Lab5/TERA_OUT
   ```

2. Note the application ID of your job. Connect to JobHistoryServer at port 19888:

   ```
   https://<IP address>:19888
   ```

   The JobHistory page displays:



192.168.100.128:19888/jobhistory

Submit Time	Start Time	Finish Time	Job ID	Name	User	Queue
2017.02.22	2017.02.22	2017.02.22	job_1487150674205_0047	Voter.jar	user01	root.user01

3. In the list of displayed jobs, scroll to the bottom to find your job (as a combination of the **Job Name** and **User** fields and **applicationid**). Click the job.
4. Look at the `terasort` standard output to determine the following:
    a. Look at the number of mappers launched. Is this equal to the number of input files?
    b. Look at the number of map and reduce input and output records. When would the number of map input records be different than the number of map output records?
    c. Look at the number of combine input and output records. What does this imply about the `terasort` application?

Lab 5.1b: Use Custom Counters

Estimated time to complete: 50 minutes

In this exercise, you will write the logic to identify a "bad" record in a data set, then define a custom counter to count "bad" records from that data set. This is what a "good" record looks like:

```
1,david   davidson,10,socialist,369.78,5108
```

There are 6 fields total – a primary key, name, age, party affiliation, and two more fields you don't care about. You will implement a record checker that validates that there are exactly 6 fields in the record, and that the third field is a "reasonable" age for a voter.

1. Change to the `VOTER_LAB` directory:

    ```
    $ cd /home/hdoop/Lab5/VOTER_LAB
    ```

2. Open the `VoterDriver.java` file with the `view` command. What character separates the keys from the values in the records? Close the file.
3. Open the `VoterMapper.java` file with your favorite editor. Which character is the value of the record tokenizing on? Keep the file open for the next step.
4. Locate the // TODO statements in the file, and implement the changes necessary to validate the record. Then save the file.
5. Compile and execute the code, using `rebuild.sh` and `rerun.sh`. Based on the minimum, maximum, and mean values for voter ages, what do you conclude about the nature of the data set?
6. Examine the output in your terminal from the job to determine the number of bad records.
    a. How many records have the wrong number of fields?
    b. How many records have a bad age field?
    c. Does the total number of bad records, plus the total number of reduce input records, equal the total number of map input records?

Lab 5.3a: Use Standard Output, Error, and Logging

Estimated time to complete: 50 minutes

In this exercise, you will generate standard error and log messages and then consume them in the MCS.

Modify the mapper to send messages to standard error

In this task, use standard error instead of job counters to keep track of bad records.

1. Open the `VoterMapper.java` file with your favorite editor.
2. Implement the following `TODO`s in the code
   - Instead of incrementing the bad record counter for incorrect number of tokens, write a message to standard error. Include the bad record in the message.

     HINT: Use `System.err.println()`
   - Instead of incrementing the bad record counter for invalid age, write a message to standard error. Include the bad record in the message.

     HINT: Use `System.err.println()`
3. Save the file.
4. Compile and execute the application.

   ```
   $ ./rebuild.sh
   $ ./rerun.sh "-DXmx1024m"
   ```
5. Examine standard error messages for your job as follows:
   a. Log in to the ResourceManager page with:

      ```
      <node IP address>:8088
      ```
   b. In the list of displayed jobs, click the job using Id.
   c. In the opened **Job Overview** page, click the **logs** link. If the page fails to load, replace the string **maprdemo** with the IP address.

Modify the mapper to write messages to syslog

In this task, you will perform the same logic as in the previous task, except you'll write the message to syslog.

1. Open the `VoterMapper.java` file with your favorite editor.
2. Implement the following `TODO`s in the code
   - Instead of incrementing the bad record counter or writing to standard error for incorrect number of tokens, write a message to syslog. Include the bad record in the message.

     HINT: Use `log.error()` from Apache Commons Logging

- Instead of incrementing the bad record counter or writing to standard error for invalid age, write a message to syslog. Include the bad record in the message.

  HINT: Use `log.error()` from Apache Commons Logging

1. Save the file.
2. Compile and execute the application. Replace the `username` variable with your login user name.

```
$ ./rebuild.sh
$ ./rerun.sh "-DXmx1024m"
```

Examine syslog messages for your job in the ResourceManager UI

In this task, you will use the ResourceManager user interface to consume syslog messages.

1. Log in to the ResourceManager user interface with:

```
<node IP address>:8088
```

2. In the list of displayed jobs, click the job using Id.
3. In the opened **Job Overview** page, click the **logs** link. If the page fails to load, replace the string **maprdemo** with the IP address.

Lab 5.3b: Use the Hadoop CLI to Manage Jobs

Estimated time to complete: 30 minutes

Launch a long-running job

1. Change to the `SLOW_LAB` directory.

```
$ cd /home/hdoop/Lab5/SLOW_LAB
```

2. Launch the MapReduce application and specify the sleep time (in ms).

```
$ ./rerun.sh "-DXmx1024m -D my.map.sleep=4000"
```

Get counter values for job

1. Log in to your cluster in a second terminal window.
2. Find the job id for your job:
   a. Go to the ResourceManager page from your browser using:

```
<node IP address>:8088
```

   b. Look for the Application Id based on name and user.

3. Display the job counter for `MAPRFS_BYTES_READ`. Replace the *`jobid`* variable using the output from the previous command. NOTE: Wait till you see that the Job has started, before running this command.

   ```
   $ mapred job -counter <jobID> \
   org.apache.hadoop.mapreduce.FileSystemCounter   MAPRFS_BYTES_READ
   ```

4. Kill the application by selecting the **Kill Application** option.

5. Reload the page to verify that the job no longer exists.

Display job history

1. Open a browser and navigate to JobHistoryServer at port 19888:

   ```
   <IP address>:19888
   ```

2. Scroll through the list and find the job using the **Id**, **Name**, and **User**. Click on the Id to view the job history.

If the job hangs, change the memory allocation for filesystem in the `warden.conf` file:

```
service.command.mfs.heapsize.max=1024
```

Lab 5.4: Test a MapReduce Application

Estimated time to complete: 45 minutes

In this exercise, the code to test the mapper is already provided. You will follow that example to implement the reducer test.

Recall the `VoterMapper` map method emits the key-value pair: (party, age). For example, with input `"1,david davidson,20,socialist,369.78,5108"` you should expect output `(socialist, 20)`.

Test the mapper against different input

1. Change to the `VOTER_LAB` directory.

   ```
   $ cd /home/hdoop/Lab5/VOTER_LAB
   ```

2. View the `mymaptest.dat` file with the `cat` command. The first line of the file is the input record. The second line of the file is the output from the map method for the input.

   ```
   $ cat mymaptest.dat
   ```

3. Test the map method against the test file – you should get a "success" message.

   ```
   $ ./retest.sh map mymaptest.dat
   ```

4. Now edit the test file so that the input and expected output do not match.

5. Test the map method against the test file – this time you should get an exception

```
$ ./retest.sh map mymaptest.dat
```

Implement code to test the reducer

1. View the `myreducetest.dat` file with the `cat` command. The first line of the file is the input record (including the key and list of values). The second line of the file is the output from the reduce method for the associated input.

```
$ cat myreducetest.dat
```

2. Implement the TODO in the *VoterTest.java* file to write the unit test for the reducer.

Build and execute the unit test for the reducer

1. Open the `VoterReducer.java` file to verify that only the key-value pair for the mean is emitted. This is because we are only examining the output for mean in our MRUnit test.

```
$ vi VoterReducer.java
```

2. Rebuild the jar file.

```
$ ./rebuild.sh
```

3. Run the reduce test against the input file.

```
$ ./retest.sh reduce myreducetest.dat
```

4. Make a change in the `myreducetest.dat` file so that the expected output intentionally does not match the expected output. Then retest the reduce method to see what the error looks like.

```
$ ./retest.sh reduce myreducetest.dat
```

Lesson 5 Answer Key

Lab 5.1a – Run Terasort

Step	Instruction or Question	Solution
4a.	Is the number of mappers equal to the number of input files?	Yes, there are two of each.
4b.	When would the number of map input records be different than the number of map output records?	If the map method is doing any sort of filtering (for example, dropping "bad" records).
4c.	What does this imply about the `terasort` application?	The `terasort` application does not use a combiner.

Lab 5.1b – Use Custom Counters

Step	Instruction or Question	Solution
2	What character separates the keys from the values in the records?	The field separator character is a comma.
3	Which character is the value of the record tokenizing on?	A comma.
5	Based on the minimum, maximum, and mean values for voter ages, what do you conclude about the nature of the data set?	The minimum, maximum, and mean values for all parties (democratic, republican, green, etc.) are exactly the same. This is unlikely, and you should investigate to make sure your data is accurate.
6a.	How many records have the wrong number of fields?	The sample data already has one record with the wrong number of fields; the instructions have you add another.
6b.	How many records have a bad age field?	The sample data already has one record with a bad age field; the instructions have you add another.
6c.	Does the total number of records, plus the total number of reduce input records, equal the total number of map input records?	Yes.

Lesson 6: Manage Performance

Lab 6.3 De-Tune a Job and Measure Performance Impact

Estimated time to complete: 30 minutes

Establish a baseline for CPU time

In this task, you will run the `teragen` and `terasort` MapReduce applications from the examples provided in the Hadoop distribution.

1. Create the lab directory.

   ```
   $ mkdir /home/hdoop/Lab6
   ```

2. Run the `teragen` MapReduce application to generate 1,000,000 records.

   ```
   $ hadoop jar /home/hdoop/hadoop-2.7.0/share/hadoop/mapreduce/hadoop-
   mapreduce-examples-2.7.0.jar teragen 1000000 /home/hdoop/Lab6/TERA_IN
   ```

3. Run the `terasort` application to sort those records you just created.

   ```
   $ hadoop jar /home/hdoop/hadoop-2.7.0/share/hadoop/mapreduce/hadoop-mapreduce-
   examples-2.7.0.jar terasort –DXmx1024m –Dmapred.reduce.tasks=2 \
   /home/hdoop/Lab6/TERA_IN   /home/hdoop/Lab6/TERA_OUT_1
   ```

4. Determine the aggregate map phase run time of the job. Connect to the JobHistoryServer using the IP address of the node:

   ```
   http://<IP address>:19888
   ```

5. Run it a few times more to establish a good baseline. Remove the output directory `/home/hdoop/Lab6/TERA_OUT_1` before each run. Here are some values for a few runs. Fill in the aggregate map phase run times for your runs, below the ones given in the table.

Run 1	Run 2	Run 3	Run 4
2m1s=121s	2m30s=150s	2m10s=130s	1m55s=115s

Modify the configuration and determine impact

1. Run the `terasort` application to sort those records again, but this time with a modification in the job parameters.

   ```
   $ hadoop jar /home/hdoop/hadoop-2.7.0/share/hadoop/mapreduce/hadoop-
   mapreduce-examples-2.7.0.jar terasort -DXmx1024m -Dmapred.reduce.tasks=2 \
   -Dio.sort.mb=1 /home/hdoop/Lab6/TERA_IN /home/hdoop/Lab6/TERA_OUT_2
   ```

2. Connect to JobHistoryServer at port 19888.

3. Determine the aggregate time spent in the map phase.

4. Run it a few times more to establish a good test. Change the name of the output directory for each rerun. Here are some values for a few runs: below those, fill in the results for your runs.

Run 1	Run 2	Run 3	Run 4
7m13s=433s	5m5s=305s	5m49s=349s	5m21s=321s

Note: There is a significant difference in the sample times shown in this table between the first run and the rest of the runs. This is one reason we take several samples when benchmarking. Without a reason for this sample, statistically we would probably discard this outlier.

5. It appears that the change has impacted the amount of time spent in the map phase (which makes sense given we are changing the `io.sort.mb` parameter). Calculate the change in performance due to the modification. Here is the calculation with the sample numbers provided: perform the same calculation with your test numbers.

- Average aggregate time, baseline (from step 4 in the previous section):

  = (121 + 150 + 130 + 115) / 4 = 129 seconds

- Average aggregate time, modified (not using outlier from Run 1):

  = (305 + 349 + 321) / 3 = 325 seconds

- Performance differential:

  = (baseline - modified) / baseline)* 100
  = ((129 – 325) / 129) * 100
  = (-196 / 129) * 100 = -151%

In other words, the modified job performs 151% slower than the baseline (takes 151% longer). If the result is a positive number, then the modified job is faster than the baseline job.

Launch Jobs and Advanced Hadoop MapReduce Applications

Lesson 7: Working with Data

Lab 7.3: Run a MapReduce Program Using HBase as Source

Estimated time to complete: 30 minutes

In this exercise, you will create and populate a table in HBase to store the voter data from previous exercises. You will then run a MapReduce program to calculate the usual maximum, minimum, and mean values using data read from that table.

Create an HBase Table Using `importtsv`

In this task, you will use the `importtsv` utility to create an HBase table using the tab-separated `VOTER` data we used in a previous lab. Log in to one of your cluster nodes to perform the steps in this task an use the `hbase` command to create an empty table in MapR-FS.

1. Log into terminal as `hdoop`.
2. Create a directory for this lab, and position yourself there.

   ```
   $ mkdir /home/hdoop/Lab7
   $ cd /home/hdoop/Lab7
   ```

3. Copy the source code for the lab from Desktop (run as root user):

   ```
   $ cp -r /root/Desktop/apache-hadoop/Lab7/* /home/hdoop/Lab7
   ```

   ```
   $ chown -R hdoop /home/hdoop/Lab7
   ```

4. Launch the HBase shell:

   ```
   $ hbase shell
   ```

5. Create the table.

   ```
   hbase> create '/home/hdoop/Lab7/myvoter_table', {NAME => 'cf1'},
   {NAME => 'cf2'}, {NAME => 'cf3'}
   ```

6. Verify the file was created in MapR-FS.

   ```
   hbase> quit
   $ ls -l /home/hdoop/Lab7/myvoter_table
   ```

7. Use the `importtsv` utility to import the data into the HBase table.

   ```
   $ hadoop jar /opt/mapr/hbase/hbase-1.1.1/lib/hbase-server-\
   1.1.1-mapr-1602.jar importtsv -Dimporttsv.columns=\
   HBASE_ROW_KEY,cf1:name,cf2:age,cf2:party,cf3:contribution_amount,\
   cf3:voter_number /home/hdoop/Lab7/myvoter_table \
   /home/hdoop/Lab7/VOTERHBASE_SOLUTION/myvoter.tsv
   ```

8. Use the `hbase` command to validate the contents of the new table.

```
$ echo "scan '/home/hdoop/Lab7/myvoter_table'" | hbase shell
   ROW      COLUMN+CELL
    1      column=cf1:name, timestamp=1406142938710, value=david
           davidson
    1      column=cf2:age, timestamp=1406142938710, value=49
    1      column=cf2:party, timestamp=1406142938710, value=socialist
    1      column=cf3:contribution_amount,   timestamp=1406142938710,
           value=369.78
    1      column=cf3:voter_number,   timestamp=1406142938710,
           value=5108
    10     column=cf1:name, timestamp=1406142938710, value=Oscar
           xylophone
   . . . <output omitted>
   1000000 row(s) in 1113.9850 seconds
```

Run the MapReduce Program to Calculate Statistics

In this task, you will run your MapReduce program and then analyze the results.

1. Change directory to the location of the MapReduce program jar file.

```
$ cd /home/hdoop/Lab7/VOTERHBASE_SOLUTION
```

2. Run the MapReduce program.

```
$ java -cp /opt/mapr/hadoop/hadoop2.7.0/share/hadoop/common\
hadoop-common-2.7.0-mapr-1602.jar:’hbase classpath’:VoterHbase.jar \
VoterHbase.VoterHbaseDriver /home/hdoop/Lab7/myvoter_table \
/home/hdoop/Lab7/OUT
```

3. Analyze the results (min, max, and mean age).

```
$ cat /home/hdoop/Lab7/OUT/part-r-00000 democrat 18.0
democrat 77.0
democrat 47.0
green 18.0
green 77.0
green 47.0
independent 18.0
independent 77.0
independent 47.0
libertarian 18.0
libertarian 77.0
libertarian 47.0
republican   18.0
republican   77.0
republican   47.0
socialist    18.0
socialist    77.0
socialist    47.0
```

Lesson 8: Launching Jobs

Lab 8.3: Write a MapReduce Driver to Launch Two Jobs

Estimated time to complete: 30 minutes

In this exercise, you will modify a MapReduce driver that launches two jobs. The first job calculates minimum, maximum, and mean values for the SAT verbal and math scores. The second job calculates the numerator and denominator for the Spearman correlation coefficient between the verbal and math scores. The driver then calculates the correlation coefficient by dividing the numerator by the square root of the denominator. The code for both MapReduce jobs has been provided.

Pearson's Correlation Coefficient

This statistic is used to determine the level of dependence (or correlation) between two variables. The value of the coefficient ranges from -1 to +1, where:

- +1 means the variables are directly proportional (high positive correlation)
- 0 means there is no correlation between the variables
- -1 means the variables are inversely proportional (high negative correlation)

The formula to calculate this coefficient is given here:

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}$$

Programming Objective

Let X represent the SAT verbal scores and Y represent the SAT math scores. The first MapReduce job calculates the mean values for X and Y, and the second MapReduce job calculates the numerator and the squared value of the denominator. The driver you write must configure and launch both jobs and then calculate the Spearman correlation coefficient.

Copy Lab Files to Cluster

1. Create a directory for this exercise and position yourself in that directory.

```
$ mkdir /home/hdoop/Lab8
$ cd /home/hdoop/Lab8
```

2. Download the lab file to your cluster, and extract the zip file.

```
$ wget http://course-files.mapr.com/DEV3000/DEV302-v5.1-Lab8.zip
$ unzip DEV302-v5.1-Lab8.zip
```

Implement the TODOs and Run the Jobs

In this task, you will configure the `WholeJobDriver.java` file in the `STATISTICS_LAB` directory to configure and launch the two MapReduce jobs and then calculate the correlation coefficient.

1. Open the `WholeJobDriver.java` file with your favorite editor.

   ```
   $ cd DEV302-v5.1-Lab8/STATISTICS_LAB
   $ vi WholeJobDriver.java
   ```

2. Locate the `TODO` statements in the file, and make the changes necessary to address the instructions.

3. Save the file.

4. Compile the java file using `whole_rebuild.sh`, and run it using `whole_rerun.sh`. Your results should be as follows:

   ```
   product_sumofsquares is 243128.0
   var1_sumofsquares is 259871.0
   var2_sumofsquares is 289679.0
   spearman's coefficient is 0.886130250066755
   ```

   You can examine the file `STATISTICS_SOLUTION/WholeJobDriver.java` if you need assistance.

Lab 9: Streaming MapReduce

Lab 9.3: Implement a MapReduce Streaming Application

Estimated time to complete: 30 minutes

In this exercise, you will implement a MapReduce streaming application using the language of your choice (Python or Perl). Guidance will be provided for building the application in the UNIX `bash` shell. We return to the `RECEIPTS` data set to calculate the minimum, maximum, mean and the years associated with the those values.

Copy the Lab Files to Your Cluster

In this task, you will copy and extract the lab files for this exercise.

1. Create a directory for this exercise, and position yourself in that directory.

```
$ mkdir /home/hdoop/Lab9
$ cd /home/hdoop/Lab9
```

2. Download and extract the lab files:

```
$ wget http://course-files.mapr.com/DEV3000/DEV302-v5.1-Lab9.zip
$ unzip DEV302-v5.1-Lab9.zip
```

Implement the Mapper

Implement the mapper in the language of your choice (Python or Perl) based on the following `bash` shell implementation of the same logic:

```
#!/bin/bash
while read record
do
   year=`echo $record | awk  '{print $1}'`
   delta=`echo $record | awk  '{print $4}'`
   printf "summary\t%s_%s\n" "$year" "$delta"
done
```

Implement the Reducer

Implement the reducer in the language of your choice (Python or Perl) based on the following `bash` shell implementation of the same logic:

```
#!/bin/bash -x
count=0
sum=0
max=-2147483647
min=2147483647
```

```
minyear=""
maxyear=""
while read line
do
   value=`echo $line | awk '{print $2}'`
   if [ -n "$value" ]
   then
      year=`echo $value | awk -F_ '{print $1}'`
      delta=`echo $value | awk -F_ '{print $2}'`
   fi
   if [ $delta -lt $min ]
   then
      min=$delta
      minyear=$year
   elif [ $delta -gt $max ]
   then
      max=$delta
      maxyear=$year
   fi
   count=$(( count + 1 ))
   sum=$(( sum + delta ))
done
mean=$(( sum / count ))
printf "min year is %s\n" "$minyear"
printf "min value is %s\n" "$min"
printf "max year is %s\n" "$maxyear"
printf "max value is %s\n" "$max"
printf "sum is %s\n" "$sum"
printf "count is %s\n" "$count"
printf "mean is %d\n" "$mean"
```

Launch the Job

1. Modify the `receipts_driver.sh` script to match the paths to your language choice (Python or Perl), naming convention, and locations for input and output.

```
#!/usr/bin/env bash
USER=`whoami`
# 1) test map script
echo -e "1901 588 525 63 588 525 63" | ./receipts_mapper.sh | od –c
# 2) test reduce script
echo -e "summary\t1901_63" | ./receipts_reducer.sh | od –c
# 3) map/reduce on Hadoop
export   JOBHOME=/user/$USER/9/STREAMING_RECEIPTS
export    CONTRIB=/opt/mapr/hadoop/hadoop-0.20.2/contrib/streaming
export   STREAMINGJAR=hadoop-*-streaming.jar
export   THEJARFILE=$CONTRIB/$STREAMINGJAR
rm -rf $JOBHOME/OUT
```

```
hadoop1 jar $THEJARFILE \
  -mapper 'receipts_mapper.sh' \
  -file receipts_mapper.sh \
  -reducer 'receipts_reducer.sh' \
  -file receipts_reducer.sh \
  -input  $JOBHOME/DATA/receipts.txt  \
  -output  $JOBHOME/OUT
```

2. Launch the MapReduce streaming job.

```
$ ./receipts_driver.sh
```

3. Examine the output.

```
$ cat /home/hdoop/Lab9/OUT/part-r-00000

min year is 2009
min value is -1412688
max year is 2000
max value is 236241
sum is -10418784
count is 111
mean is -93862
```