Tutorial on Hadoop HDFS and MapReduce

Hortonworks

Table Of Contents

Introduction

In this tutorial, you will execute a simple **Hadoop MapReduce** job. This **MapReduce** job takes a semi-structured log file as input, and generates an output file that contains the log level along with its frequency count.

Our input data consists of a semi-structured **log4j** file in the following format:

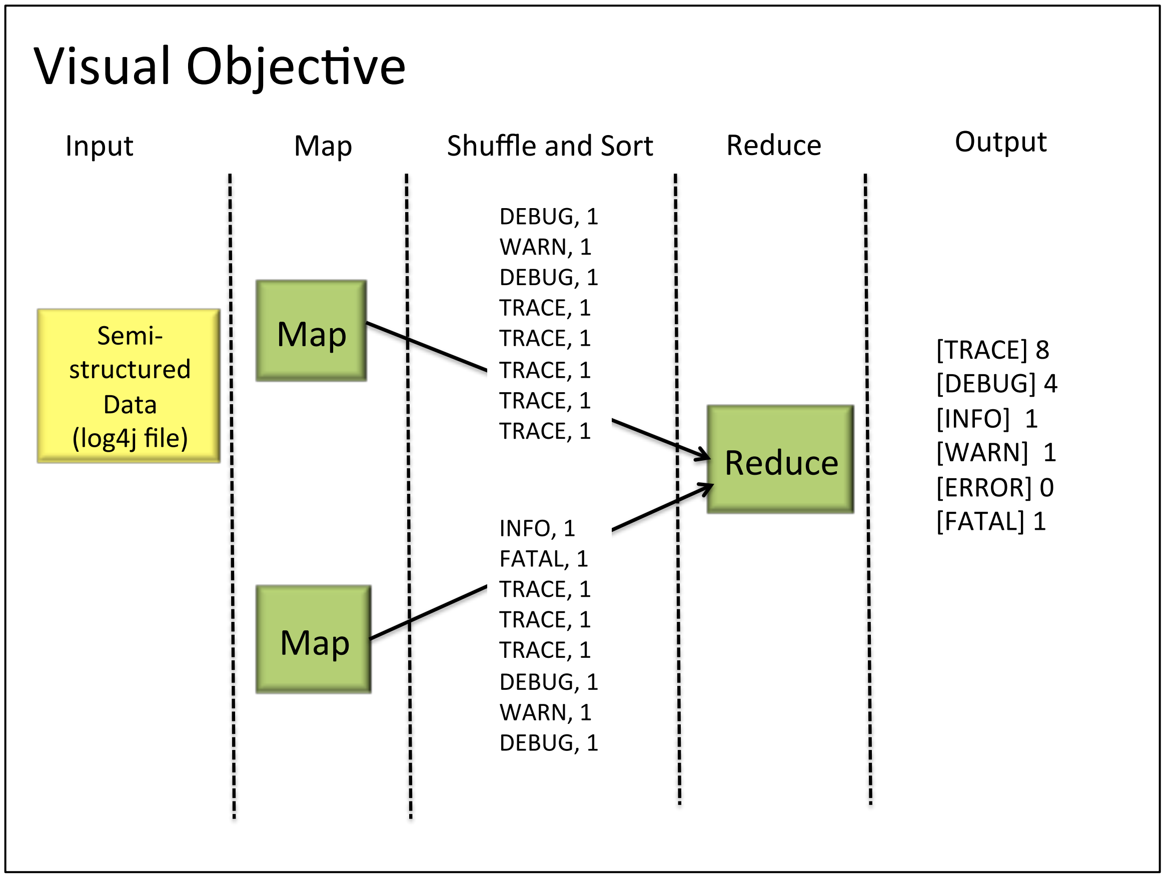
```
. . . . . . . . . . .

2012-02-03 20:26:41 SampleClass3 [TRACE] verbose detail for id
1527353937
java.lang.Exception: 2012-02-03 20:26:41 SampleClass9 [ERROR]
incorrect format for id 324411615
      at com.osa.mocklogger.MockLogger#2.run(MockLogger.java:83)
2012-02-03 20:26:41 SampleClass2 [TRACE] verbose detail for id
191364434
2012-02-03 20:26:41 SampleClass1 [DEBUG] detail for id 903114158
2012-02-03 20:26:41 SampleClass8 [TRACE] verbose detail for id
1331132178
2012-02-03 20:26:41 SampleClass8 [INFO] everything normal for id
1490351510
2012-02-03 20:32:47 SampleClass8 [TRACE] verbose detail for id
1700820764
2012-02-03 20:32:47 SampleClass2 [DEBUG] detail for id 364472047
2012-02-03 20:32:47 SampleClass7 [TRACE] verbose detail for id
1006511432
2012-02-03 20:32:47 SampleClass4 [TRACE] verbose detail for id
1252673849
2012-02-03 20:32:47 SampleClass0 [DEBUG] detail for id 881008264
2012-02-03 20:32:47 SampleClass0 [TRACE] verbose detail for id
1104034268
2012-02-03 20:32:47 SampleClass6 [TRACE] verbose detail for id
1527612691
java.lang.Exception: 2012-02-03 20:32:47 SampleClass7 [WARN]
problem finding id 484546105
      at com.osa.mocklogger.MockLogger#2.run(MockLogger.java:83)
2012-02-03 20:32:47 SampleClass0 [DEBUG] detail for id 2521054
2012-02-03 21:05:21 SampleClass6 [FATAL] system problem at id
1620503499
. . . . . . . . . . . . . .
```

The output data will be put into a file showing the various log4j log levels along with its frequency occurrence in our input file. A sample of these metrics is displayed below:

```
[TRACE] 8
[DEBUG] 4
[INFO]  1
[WARN]  1
[ERROR] 1
[FATAL] 1
```

This tutorial takes about 30 minutes to complete and is divided into the following five tasks:

The visual representation of what you will accomplish in this tutorial is shown in the figure.



The Use Case

Generally, all applications save errors, exceptions and other coded issues in a log file so administrators can review the problems, or generate certain metrics from the log file data. These log files usually get quite large in size, containing a wealth of data that must be processed and mined.

Log files are a good example of **big data**. Working with big data is difficult using relational databases with statistics and visualization packages. Due to the large amounts of data and the computation of this data, parallel software running on tens, hundreds, or even thousands of servers is often required to compute this data in a reasonable time. **Hadoop** provides a **MapReduce framework** for writing applications that process large amounts of structured and semi-structured data in parallel across large clusters of machines in a very reliable and fault-tolerant manner.

In this tutorial, you will use an semi-structured, application log4j log file as input, and generate a **Hadoop MapReduce** job that will report some basic statistics as output.

Pre-Requisites

Ensure that these pre-requisites have been met prior to starting the tutorial.

- Access to Hortonworks Virtual Sandbox—This tutorial uses a hosted solution that runs in an Amazon Web Services (AWS) EC2 environment. It provides a packaged environment for demonstration and trial use of the Apache Hadoop ecosystem on a single node (pseudo-distributed mode).

- The Virtual Sandbox is accessible as an Amazon Machine Image (AMI) and requires that you have an account with AWS.

- Working knowledge of Linux OS.

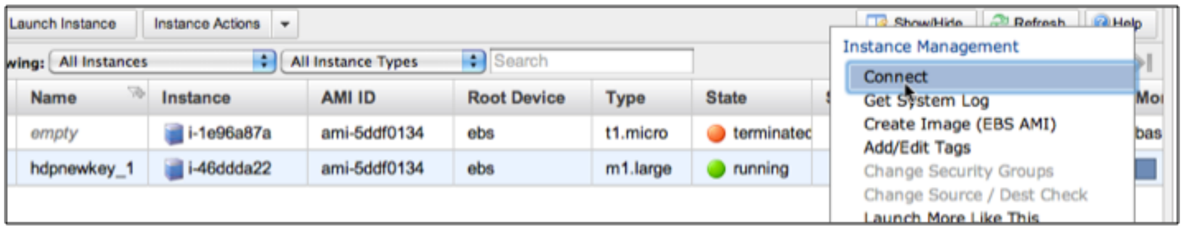
For help in getting an AWS account and configuring to the Hortonworks Virtual Sandbox, refer to Using the Hortonworks Virtual Sandbox.

Task 1: Access Your Hortonworks Virtual Sandbox

Note: If you have the Hortonworks Virtual Sandbox up and running, are connected to the instance via SSH, and verified that HDP services have been started, you can skip to Task 2.

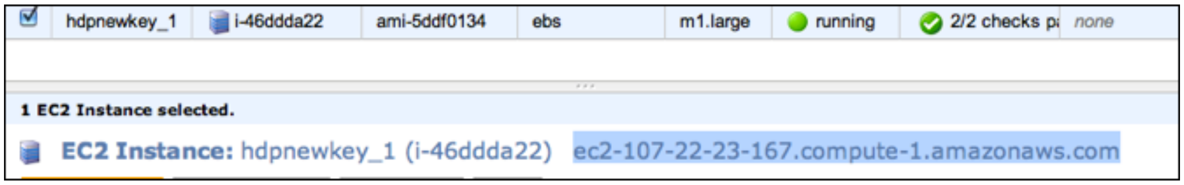
STEP 1: If you have not done so, access your AWS Management console. (https://console.aws.amazon.com/console/home and choose **EC2**)

On the AWS Management Console, go to the **My Instances** page. Right click on the row containing your instance and click **Connect**.



Name	Instance	AMI ID	Root Device	Type	State
empty	i-1e96a87a	ami-5ddf0134	ebs	t1.micro	terminated
hdpnewkey_1	i-46ddda22	ami-5ddf0134	ebs	m1.large	running

STEP 2: Copy the AWS name of your instance.



hdpnewkey_1	i-46ddda22	ami-5ddf0134	ebs	m1.large	running	2/2 checks p	none

1 EC2 Instance selected.

EC2 Instance: hdpnewkey_1 (i-46ddda22) ec2-107-22-23-167.compute-1.amazonaws.com

STEP 3: Connect to the instance using SSH. From the SSH client on your local client machine, enter:

```
ssh -i <Full_Path_To_Key_Pair_File> root@<EC2 Instance Name>
```

In the example above you would enter: ec2-107-22-23-167.compute-1.amazonaws.com.

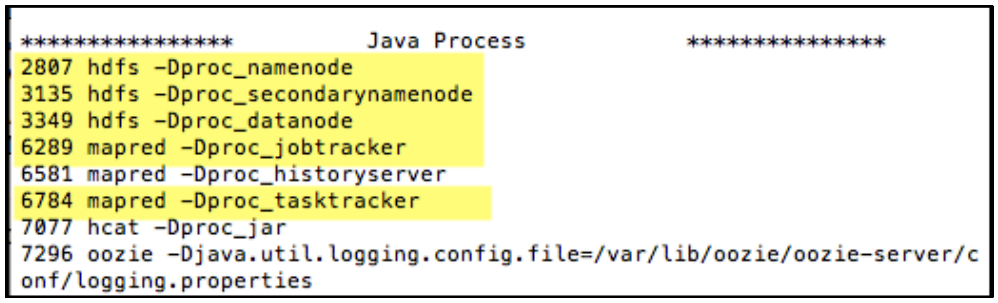
STEP 4: Verify that HDP services are started. MapReduce requires several services should have been started if you completed the pre-requisite "Using The Hortonworks Virtual Sandbox".

These services are:

- NameNode
- JobTracker
- SecondaryNameNode
- DataNode
- TaskTracker

Use an editor of your choice to view the hdp-stack-start-<date>-<time>.log file (located here: /root). (Use **ls** to get the actual <date><time>)

This file provides a list of all the Hadoop services that have started successfully. For example, the following screenshot provides the output of the tail end of this log file:



```
****************          Java Process          ***************
2807 hdfs -Dproc_namenode
3135 hdfs -Dproc_secondarynamenode
3349 hdfs -Dproc_datanode
6289 mapred -Dproc_jobtracker
6581 mapred -Dproc_historyserver
6784 mapred -Dproc_tasktracker
7077 hcat -Dproc_jar
7296 oozie -Djava.util.logging.config.file=/var/lib/oozie/oozie-server/c
onf/logging.properties
```

If the services have not been started, recall that the command to start HDP services is **/etc/init.d/hdp-stack start.**

Task 2: Create the MapReduce job

STEP 1: Change to the directory containing the tutorial:

```
# cd tutorial
```

STEP 2: Examine the **MapReduce** job by viewing the contents of the **Tutorial1.java** file:

```
# more Tutorial1.java
```

Note: Press spacebar to page through the contents or enter **q** to quit and return to the command prompt.

This program (shown below) defines a Mapper and a Reducer that will be called.

```
//Standard Java imports
import java.io.IOException;
import java.util.Iterator;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

//Hadoop imports
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;

/**
 * Tutorial1
 *
 */
public class Tutorial1
{
      //The Mapper
      public static class Map extends MapReduceBase implements
Mapper<LongWritable, Text, Text, IntWritable>
      {
            //Log levels to search for
            private static final Pattern pattern =
Pattern.compile("(TRACE)|(DEBUG)|(INFO)|(WARN)|(ERROR)|(FATAL)");
            private static final IntWritable accumulator = new
IntWritable(1);

            private Text logLevel = new Text();
            public void map(LongWritable key, Text value,
OutputCollector<Text, IntWritable> collector, Reporter reporter)
                        throws IOException
                        {
                  // split on space, '[', and ']'
                  final String[] tokens = value.toString().split("[
\\[\\]]");
                  if(tokens != null)
                  {
                        //now find the log level token
                        for(final String token : tokens)
                        {
                              final Matcher matcher =
pattern.matcher(token);
                              //log level found
                              if(matcher.matches())
                              {
                                    logLevel.set(token);
                                    //Create the key value pairs
                                    collector.collect(logLevel,
accumulator);
                              }
                        }
                  }
            }
      }

      //The Reducer
      public static class Reduce extends MapReduceBase implements
Reducer<Text, IntWritable, Text, IntWritable>
      {
            public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> collector,
                        Reporter reporter) throws IOException
            {
                  int count = 0;
```

```
                //code to aggregate the occurrence
                while(values.hasNext())
                {
                      count += values.next().get();
                }
                System.out.println(key +  "\t" + count);

                collector.collect(key, new IntWritable(count));
          }
    }

    //The java main method to execute the MapReduce job
    public static void main(String[] args) throws Exception
    {
          //Code to create a new Job specifying the MapReduce class
          final JobConf conf = new JobConf(Tutorial1.class);
          conf.setOutputKeyClass(Text.class);
          conf.setOutputValueClass(IntWritable.class);
          conf.setMapperClass(Map.class);
          // Combiner is commented out – to be used in bonus activity
          //conf.setCombinerClass(Reduce.class);
          conf.setReducerClass(Reduce.class);
          conf.setInputFormat(TextInputFormat.class);
          conf.setOutputFormat(TextOutputFormat.class);

          //File Input argument passed as a command line argument
          FileInputFormat.setInputPaths(conf, new Path(args[0]));

          //File Output argument passed as a command line argument
          FileOutputFormat.setOutputPath(conf, new Path(args[1]));

          //statement to execute the job
          JobClient.runJob(conf);
    }
}
```

Note: You can also examine the contents of the file using a text editor such as vi. Exit vi by typing Esc : q ! return (in sequence).

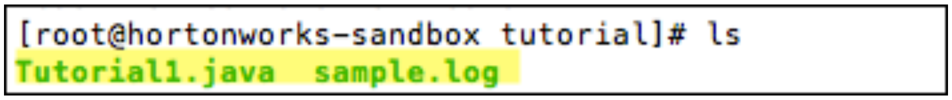
STEP 4: Compile the java file:

```
# javac -classpath /usr/share/hadoop/hadoop-core-*.jar Tutorial1.java
```

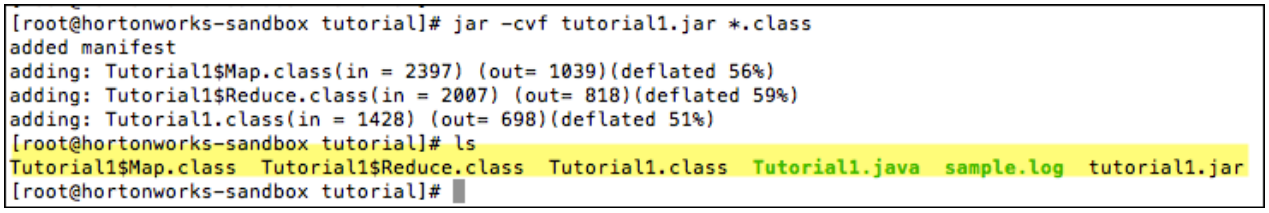
STEP 5: Create a **tutorial1.jar** file containing the Hadoop **class** files:

```
# jar -cvf tutorial1.jar *.class
```

Notice the results before and after executing the **jar** command, including verifying the existence of the **sample.log** file in the non-HDFS directory structure (used later).



```
[root@hortonworks-sandbox tutorial]# ls
Tutorial1.java  sample.log
```



```
[root@hortonworks-sandbox tutorial]# jar -cvf tutorial1.jar *.class
added manifest
adding: Tutorial1$Map.class(in = 2397) (out= 1039)(deflated 56%)
adding: Tutorial1$Reduce.class(in = 2007) (out= 818)(deflated 59%)
adding: Tutorial1.class(in = 1428) (out= 698)(deflated 51%)
[root@hortonworks-sandbox tutorial]# ls
Tutorial1$Map.class  Tutorial1$Reduce.class  Tutorial1.class  Tutorial1.java  sample.log  tutorial1.jar
[root@hortonworks-sandbox tutorial]#
```

Task 3: Import the input data in HDFS and Run MapReduce

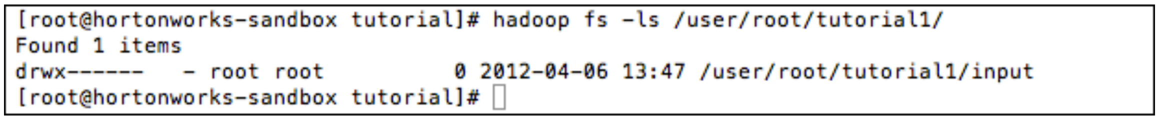
The MapReduce job reads data from HDFS. In this task, we will place the sample.log file data into HDFS where MapReduce will read it and run the job.

STEP 1: Create an input directory in **HDFS**:

```
# hadoop fs -mkdir tutorial1/input/
```

STEP 2: Verify that the input directory has been created in the Hadoop file system:

```
# hadoop fs -ls /user/root/tutorial1/
```



```
[root@hortonworks-sandbox tutorial]# hadoop fs -ls /user/root/tutorial1/
Found 1 items
drwx------   - root root          0 2012-04-06 13:47 /user/root/tutorial1/input
[root@hortonworks-sandbox tutorial]#
```

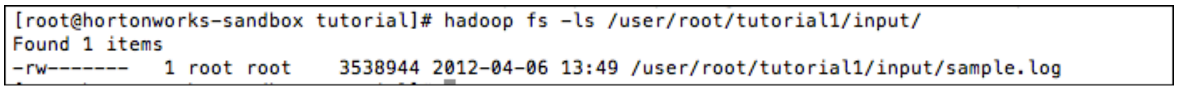
STEP 3: Load the **sample.log** input file into **HDFS**:

```
# hadoop fs -put sample.log /user/root/tutorial1/input/
```

Note: You are also creating the input directory in this step.

STEP 4: Verify that the sample.log has been loaded into HDFS:

```
# hadoop fs -ls /user/root/tutorial1/input/
```



```
[root@hortonworks-sandbox tutorial]# hadoop fs -ls /user/root/tutorial1/input/
Found 1 items
-rw-------   1 root root    3538944 2012-04-06 13:49 /user/root/tutorial1/input/sample.log
```

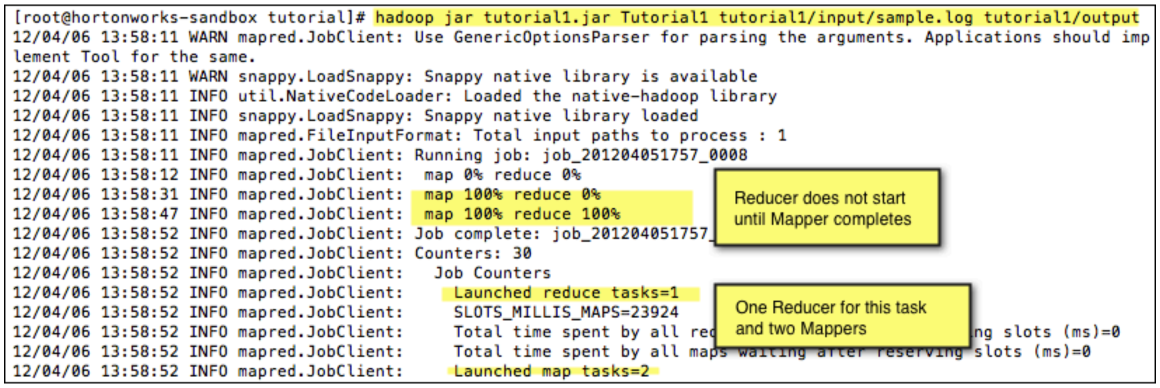
STEP 5: Run the **Hadoop MapReduce** job

In this step, we are doing a number of things, as follows:

- Calling the Hadoop program
- Specifying the jar file (tutorial1.jar)
- Indicating the class file (Tutorial1)
- Specifying the input file (tutorial1/input/sample.log), and output directory (tutorial1/output)
- Running the MapReduce job

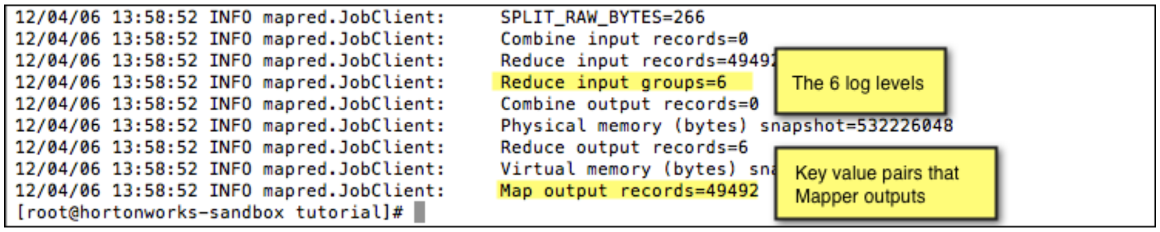
```
# hadoop jar tutorial1.jar Tutorial1 tutorial1/input/sample.log tutorial1/output
```

The Reduce programs begin to process the data when the Map programs are 100% complete. Prior to that, the Reducer(s) queries the Mappers for intermediate data and gathers the data, but waits to process. This is shown in the following screenshot.


Reducer does not start until Mapper completes
One Reducer for this task and two Mappers

```
[root@hortonworks-sandbox tutorial]# hadoop jar tutorial1.jar Tutorial1 tutorial1/input/sample.log tutorial1/output
12/04/06 13:58:11 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications should imp
lement Tool for the same.
12/04/06 13:58:11 WARN snappy.LoadSnappy: Snappy native library is available
12/04/06 13:58:11 INFO util.NativeCodeLoader: Loaded the native-hadoop library
12/04/06 13:58:11 INFO snappy.LoadSnappy: Snappy native library loaded
12/04/06 13:58:11 INFO mapred.FileInputFormat: Total input paths to process : 1
12/04/06 13:58:11 INFO mapred.JobClient: Running job: job_201204051757_0008
12/04/06 13:58:12 INFO mapred.JobClient:  map 0% reduce 0%
12/04/06 13:58:31 INFO mapred.JobClient:  map 100% reduce 0%
12/04/06 13:58:47 INFO mapred.JobClient:  map 100% reduce 100%
12/04/06 13:58:52 INFO mapred.JobClient: Job complete: job_201204051757_
12/04/06 13:58:52 INFO mapred.JobClient: Counters: 30
12/04/06 13:58:52 INFO mapred.JobClient:   Job Counters
12/04/06 13:58:52 INFO mapred.JobClient:     Launched reduce tasks=1
12/04/06 13:58:52 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=23924
12/04/06 13:58:52 INFO mapred.JobClient:     Total time spent by all red                          ng slots (ms)=0
12/04/06 13:58:52 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
12/04/06 13:58:52 INFO mapred.JobClient:     Launched map tasks=2
```

The next screen output shows Map output records=80194, and Reduce output records=6. As you can see, the Reduce program condensed the set of intermediate values that share the same key (DEBUG, ERROR, FATAL, and so on) to a smaller set of values.



```
12/04/06 13:58:52 INFO mapred.JobClient:     SPLIT_RAW_BYTES=266
12/04/06 13:58:52 INFO mapred.JobClient:     Combine input records=0
12/04/06 13:58:52 INFO mapred.JobClient:     Reduce input records=4949
12/04/06 13:58:52 INFO mapred.JobClient:     Reduce input groups=6
12/04/06 13:58:52 INFO mapred.JobClient:     Combine output records=0
12/04/06 13:58:52 INFO mapred.JobClient:     Physical memory (bytes) snapshot=532226048
12/04/06 13:58:52 INFO mapred.JobClient:     Reduce output records=6
12/04/06 13:58:52 INFO mapred.JobClient:     Virtual memory (bytes) sn
12/04/06 13:58:52 INFO mapred.JobClient:     Map output records=49492
[root@hortonworks-sandbox tutorial]#
```

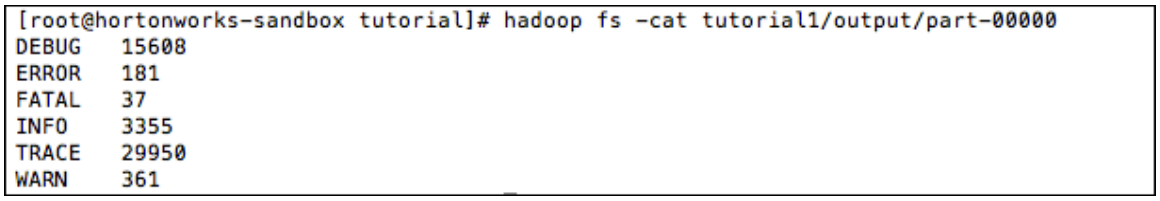
Task 4: Examine the MapReduce job's output on HDFS

STEP 1: View the output of the **MapReduce** job on **HDFS**:

```
# hadoop fs -cat tutorial1/output/part-00000
```

Note: By default, Hadoop creates files begin with the following naming convention: "part-00000". Additional files created by the same job will have the number increased.

After executing the command, you should see the following output:



```
[root@hortonworks-sandbox tutorial]# hadoop fs -cat tutorial1/output/part-00000
DEBUG   15608
ERROR   181
FATAL   37
INFO    3355
TRACE   29950
WARN    361
```

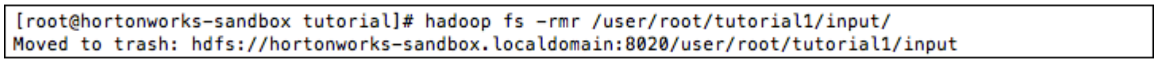
Notice that after running MapReduce that the data types are now totaled and in a structured format.

Task 5: Tutorial Clean Up

The clean up task applies to this tutorial only; it is not performed in the actual deployment. In this task, you will delete input and output directories so that if you like, you can run the tutorial again. A way to gracefully shut down processes when exiting the VM is also shown below.

STEP 1: Delete the input directory and recursively delete files within the directory:

```
# hadoop fs –rmr /user/root/tutorial1/input/
```



```
[root@hortonworks-sandbox tutorial]# hadoop fs -rmr /user/root/tutorial1/input/
Moved to trash: hdfs://hortonworks-sandbox.localdomain:8020/user/root/tutorial1/input
```

STEP 2: Delete the output directory and recursively delete files within the directory:

```
# hadoop fs –rmr /user/root/tutorial1/output/
```

Congratulations! You have successfully completed this tutorial.