Hadoop MapReduce

...

CS157B Frank Mock Spring 2017

Some things I Will Discuss

1. Installing Big Data platform - HortonWorks
2. Yelp Data Ingestion Process
3. Briefly Overview of MapReduce
4. How to design MapReduce job
5. Steps to run HelloWorld example (a.k.a. WordCount)
6. Hadoop data types
7. My MapReduce job implementation
8. MapReduce Job Optimization
9. What I learned

Big Data Platform

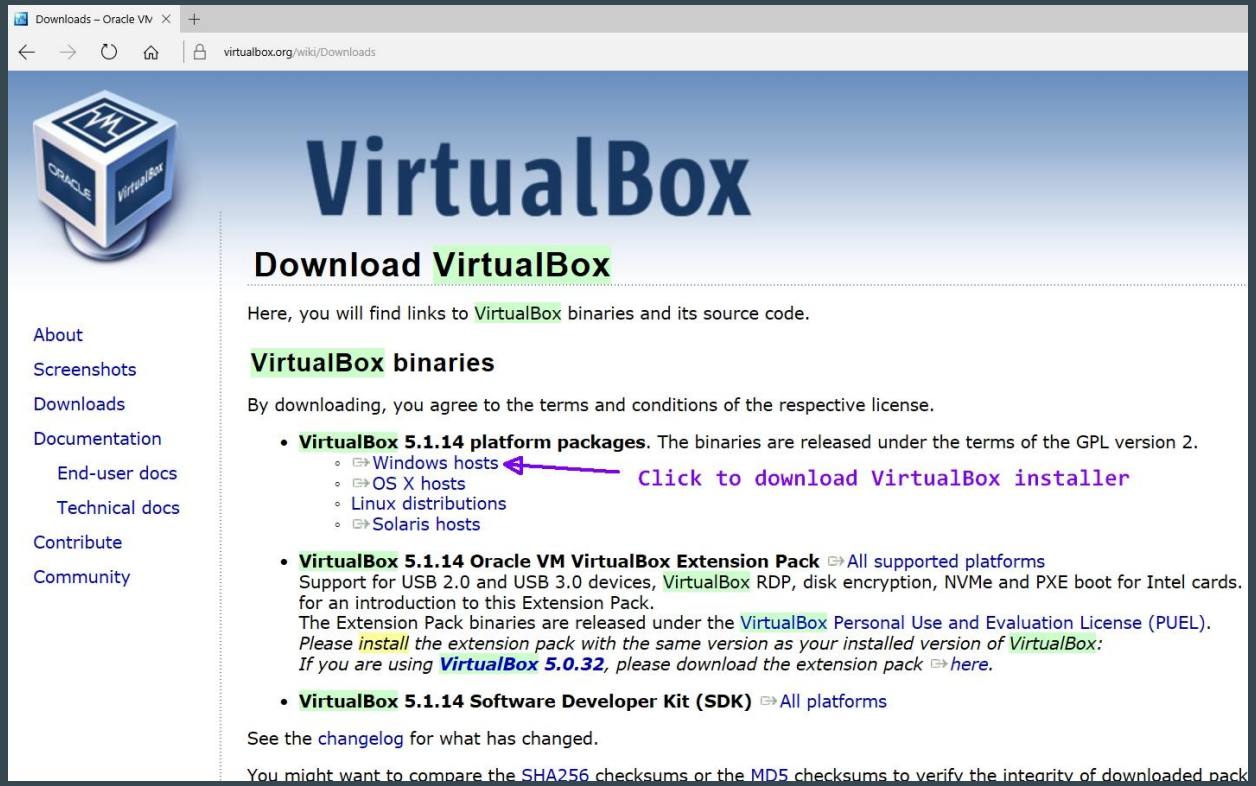
HortonWorks is an all-in-one Hadoop Big Data platform

HortonWorks includes Hadoop MapReduce

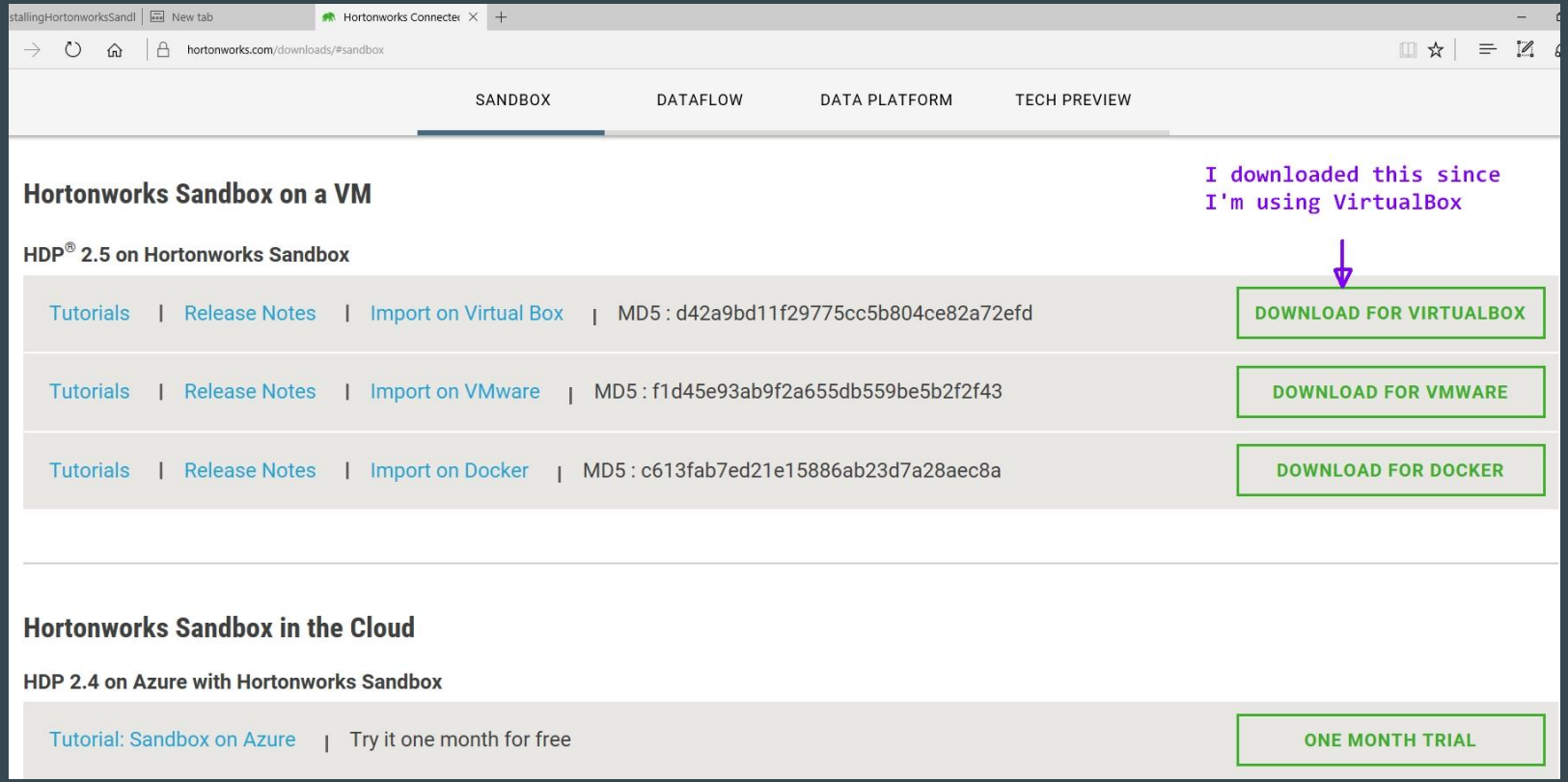
And it includes other higher-level Big Data tools that abstract away the gorey stuff

Like Hive, Pig, etc.

Must Install a Virtual Machine to Use HortonWorks

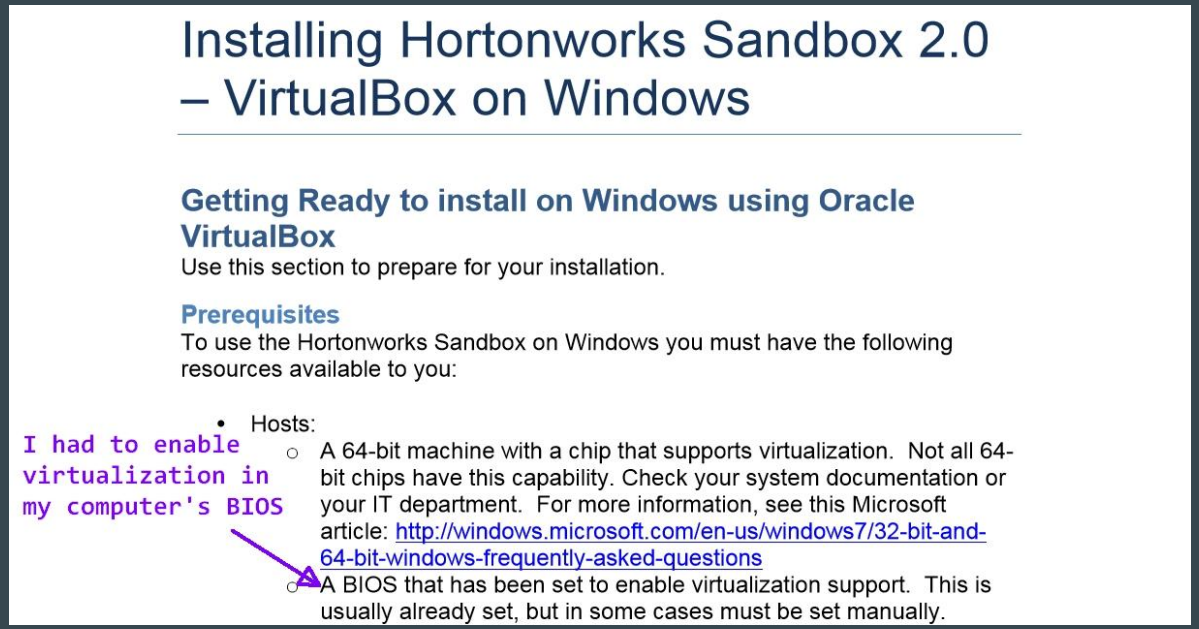


Download Version of HortonWorks for Your VM



Followed Instructions Supplied by HortonWorks

http://hortonworks.com/wp-content/uploads/unversioned/pdfs/InstallingHortonworksSandbox2onWindowsusingVB.pdf



Installing Hortonworks Sandbox 2.0 – VirtualBox on Windows

Getting Ready to install on Windows using Oracle VirtualBox

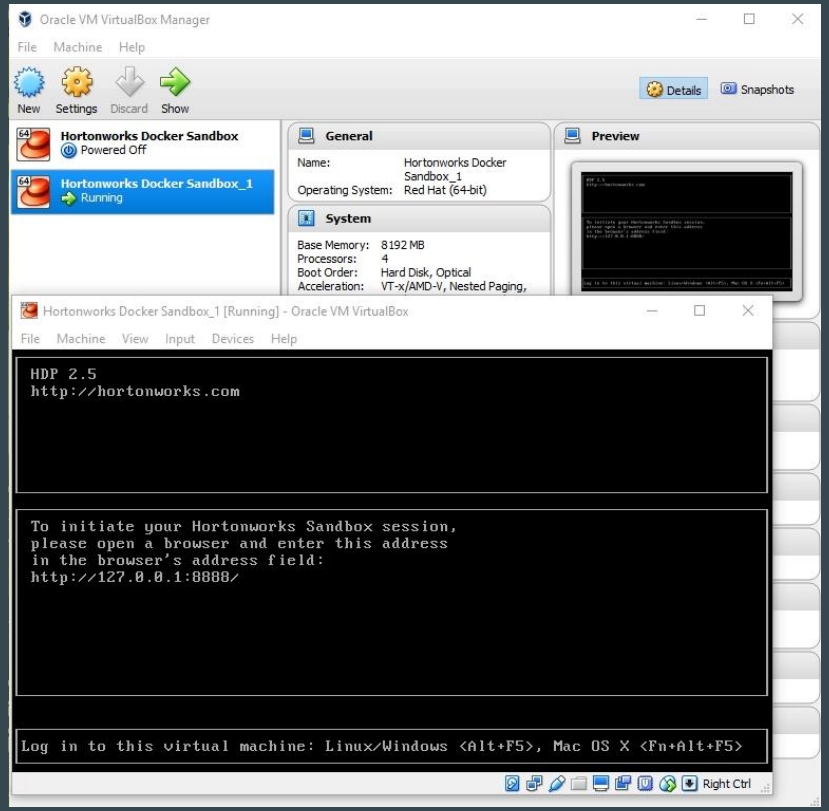
Use this section to prepare for your installation.

Prerequisites

To use the Hortonworks Sandbox on Windows you must have the following resources available to you:

- Hosts:
  - A 64-bit machine with a chip that supports virtualization. Not all 64-bit chips have this capability. Check your system documentation or your IT department. For more information, see this Microsoft article: http://windows.microsoft.com/en-us/windows7/32-bit-and-64-bit-windows-frequently-asked-questions
  - A BIOS that has been set to enable virtualization support. This is usually already set, but in some cases must be set manually.

Running HortonWorks



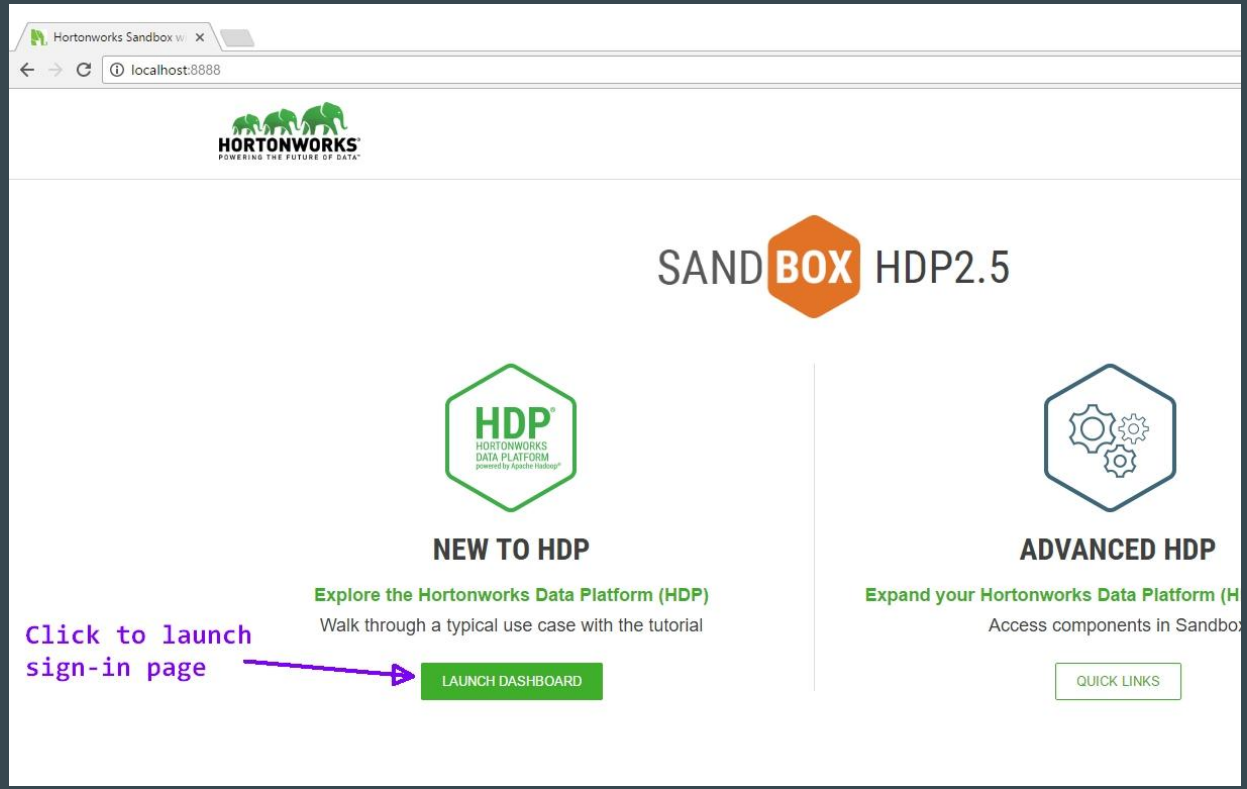
After installation, start the HortonWorks virtual machine in VirtualBox.

It takes a minute to startup - be patient

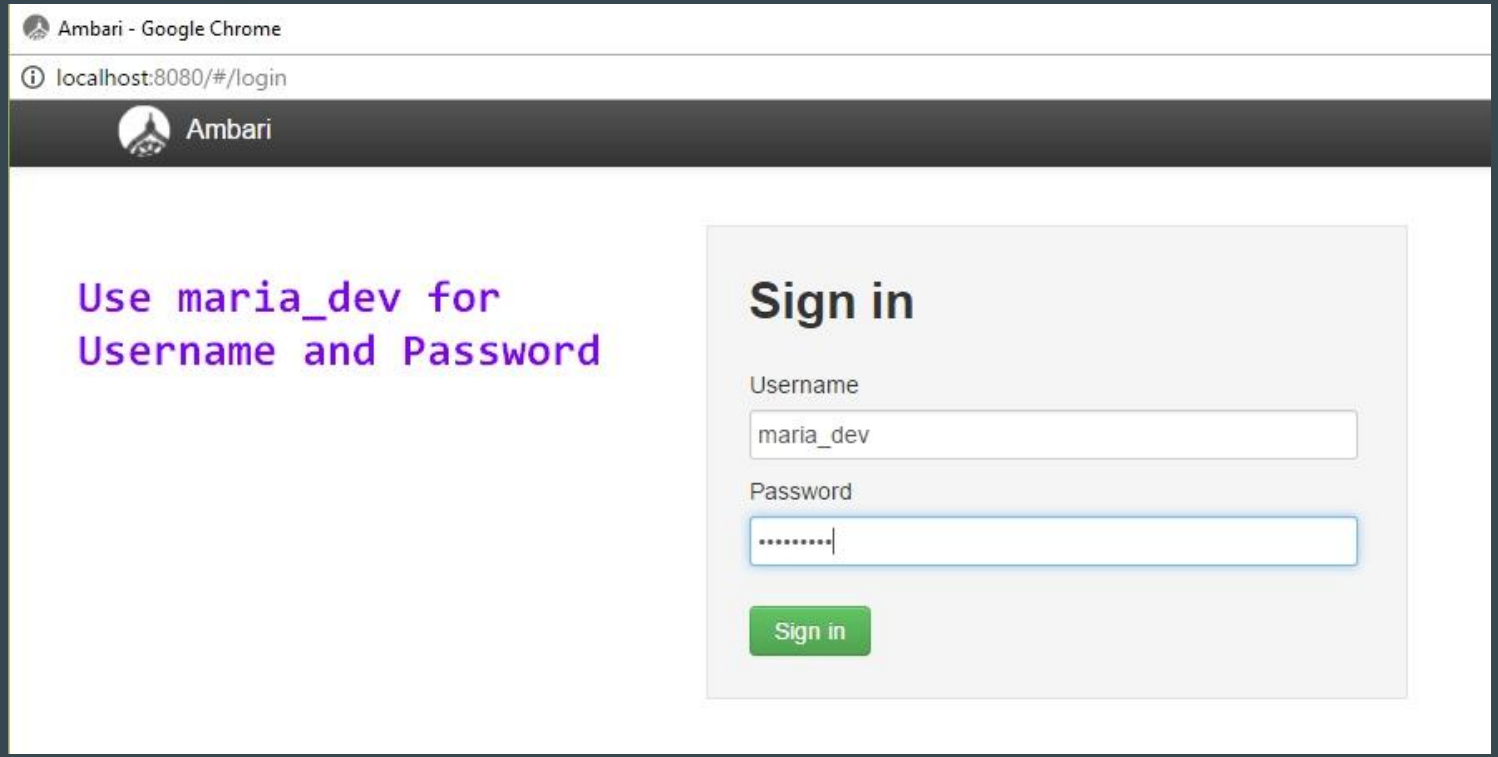
After startup, open a web browser and go to http://127.0.0.1:8888/ to login

Be sure to disable pop-up blocking in your web browser

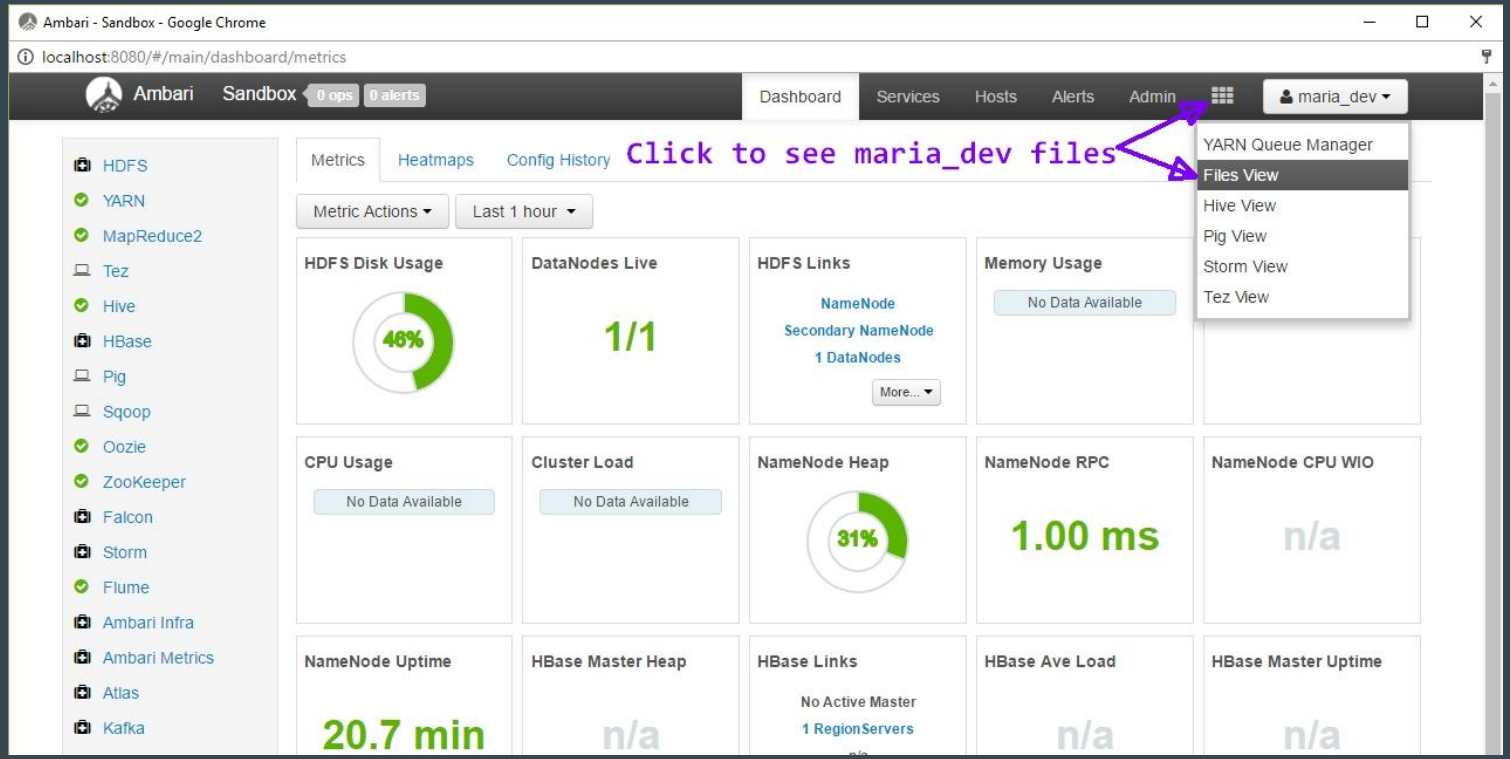
After entering URL in Web Browser



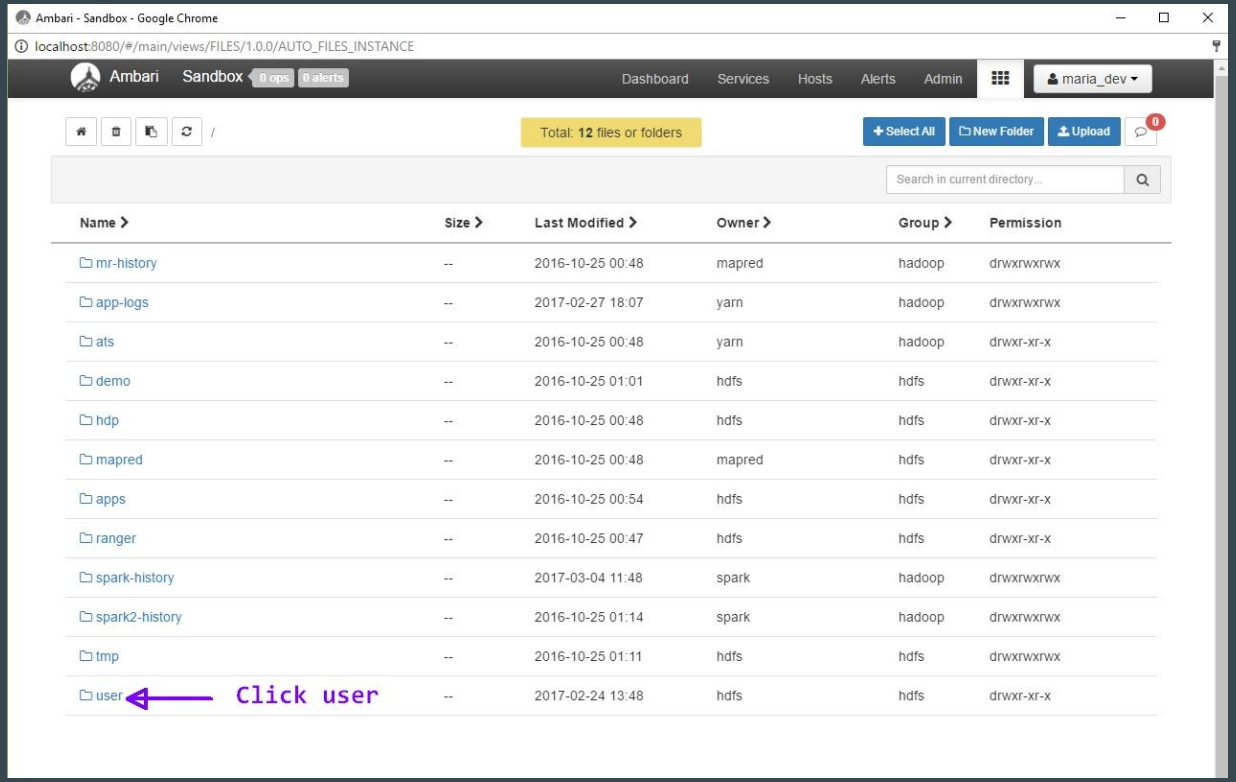
Logging in to HortonWorks Sandbox



Sandbox Dashboard

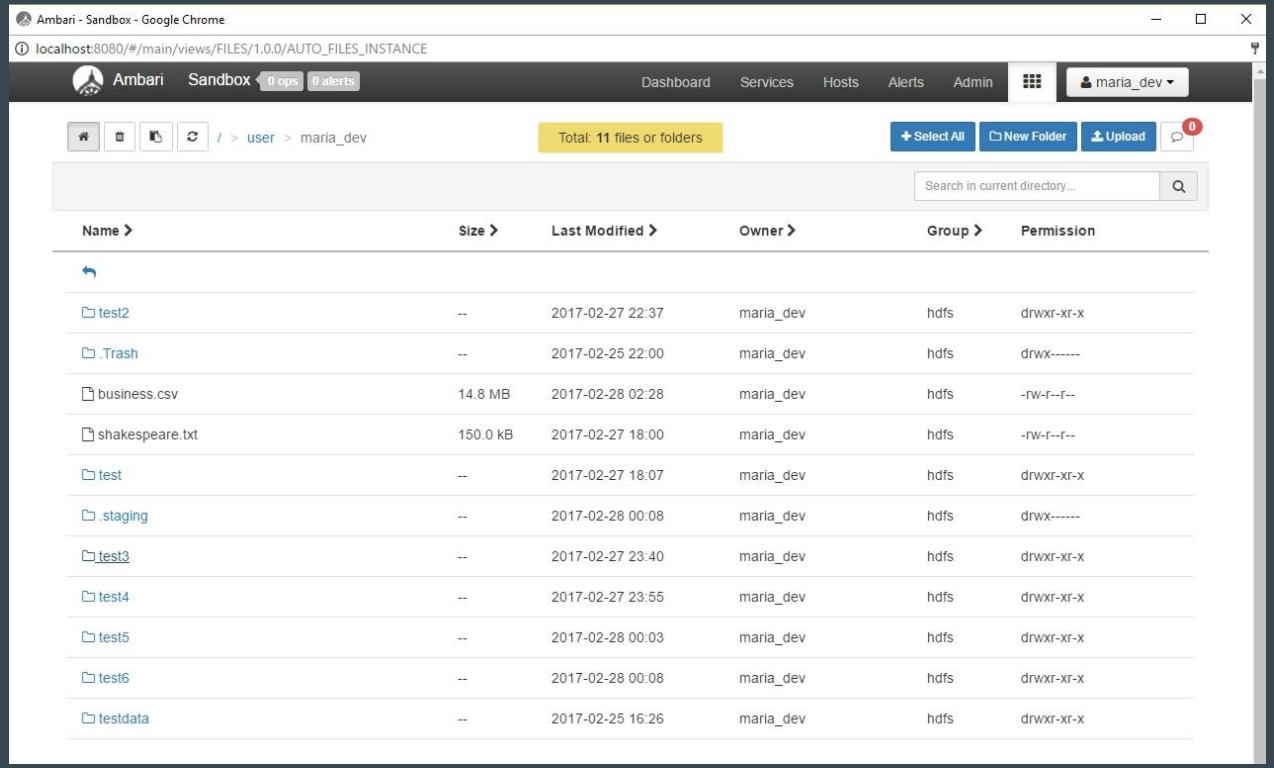


Navigating the File System



Click user and on the next page click maria_dev

maria_dev Directories and Files



Created Files From Yelp MySQL Data

I decided to re-create a subset of the Yelp data since it was ‘clean data’

Saved data as CSV files

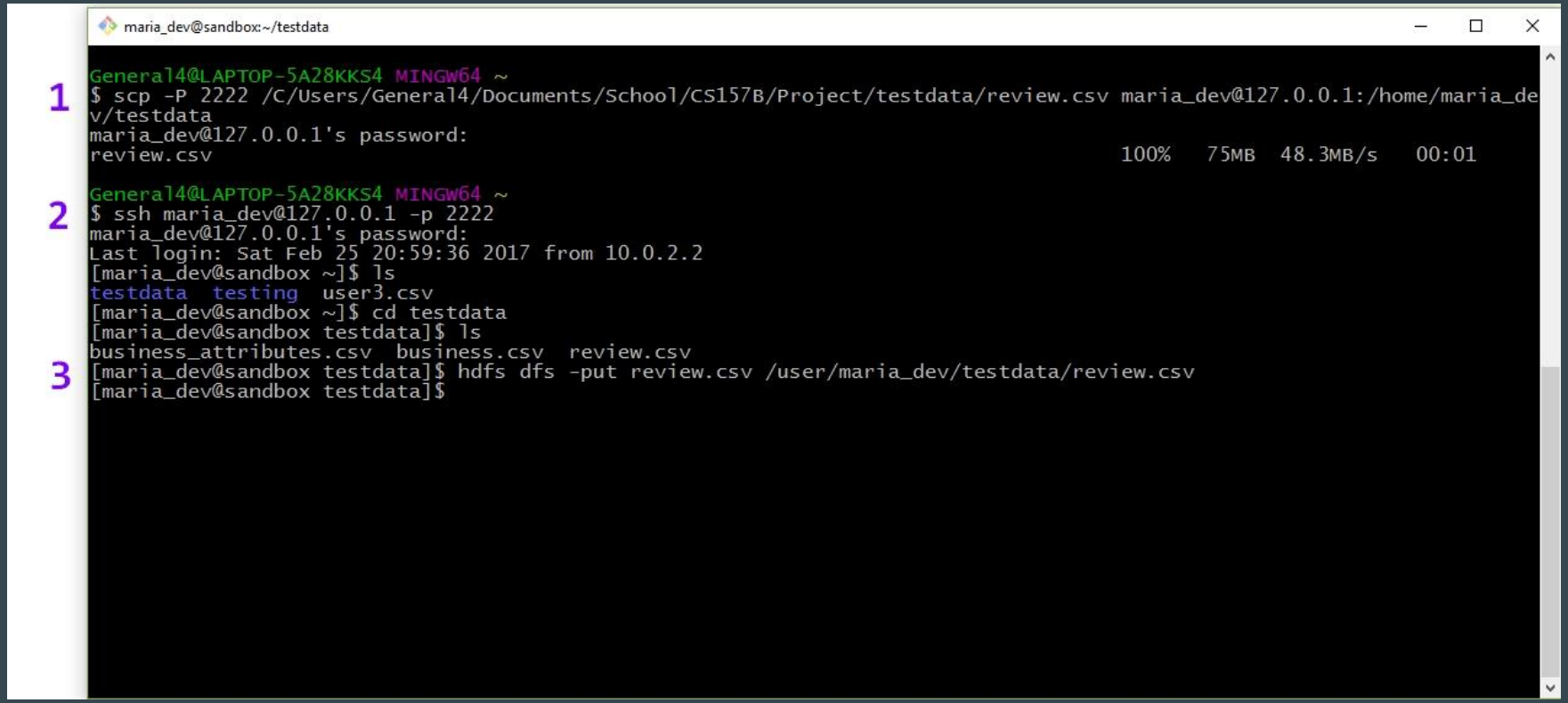
```
mysql> (SELECT 'user_id', 'name', 'review_count', 'yelping_since', 'useful', 'funny', 'cool', 'fans', 'average_stars', 'compli
compliment_funny', 'compliment_writer', 'compliment_photos', 'type')
    -> Union
    -> (SELECT user_id, name, review_count, yelping_since, useful, funny, cool, fans, average_stars, compliment_hot, complimen
, compliment_photos, type
    -> FROM user
    -> LIMIT 100000
    -> Into Outfile 'C:/ProgramData/MySQL/MySQL Server 5.6/Uploads/user.csv'
    -> Fields Enclosed By '"' Terminated By ',' Escaped By '"'
    -> Lines Terminated By '\r\n');
Query OK, 100001 rows affected (1 min 36.65 sec)

mysql> _
```

Getting Yelp data into HortonWorks

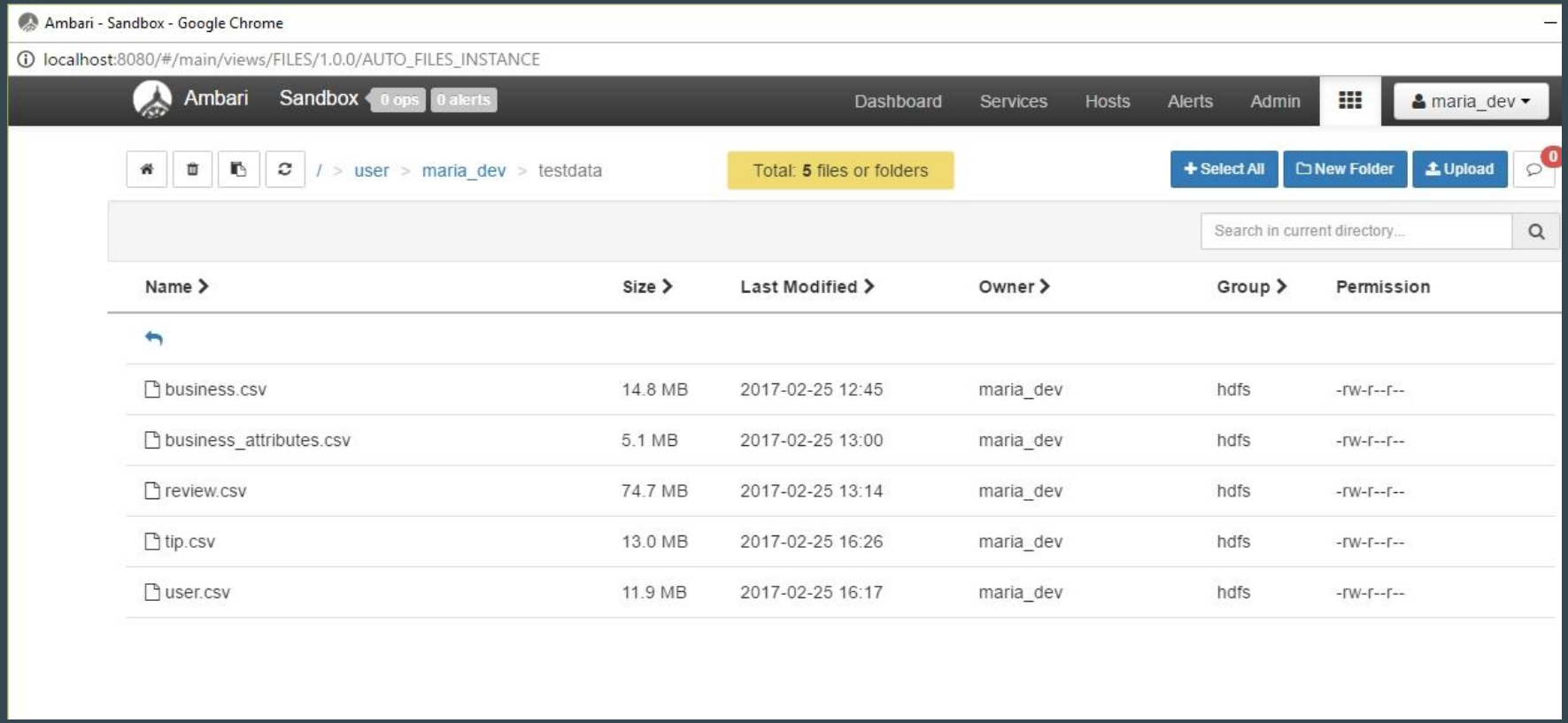
- Loaded a subset of the Yelp data - 100,000 records from each table
- Opted to use the command-line instead of GUI for data ingestion
- Learned that transferring a single file to Sandbox is 3-step process

1. `scp -P 2222 <local path of file> < user: path on remote HDFS>`
2. `ssh maria_dev@127.0.0.1 -p 2222` to login to Hadoop HDFS
3. `hdsf dfs -put <file_name> <dest. path in SandBox>`

Command-Line File Transfer a 3-Step Process



Yelp files in Sandbox



What is MapReduce?

Programming model designed for ‘Big Data’ batch processing

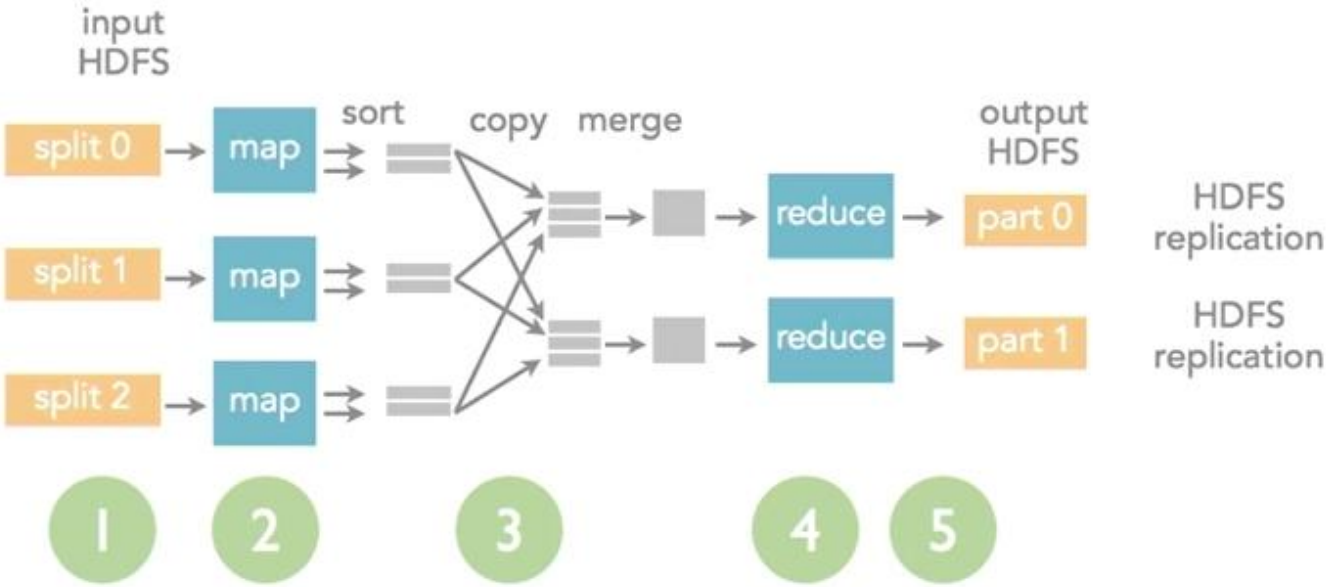
Originally created out of research by Google - (designed to index the internet)

There are two parts:

- Map
- Reduce

To use MapReduce to solve problem you must decompose your problem into a Map task and a Reduce task. Based on the problem, reduction may not be necessary.

Visualizing MapReduce



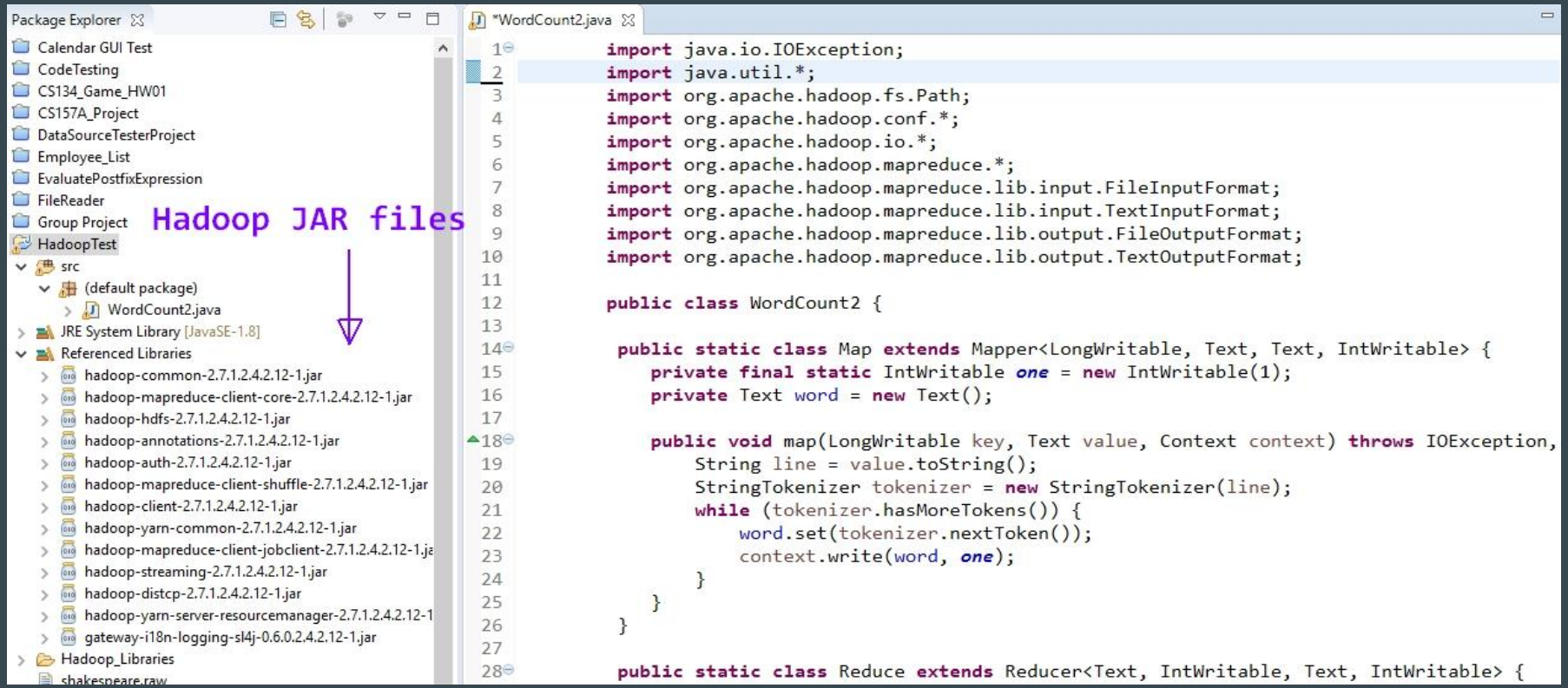
First Run Practice MapReduce Job

Did the HelloWorld MapReduce tutorial on HortonWorks

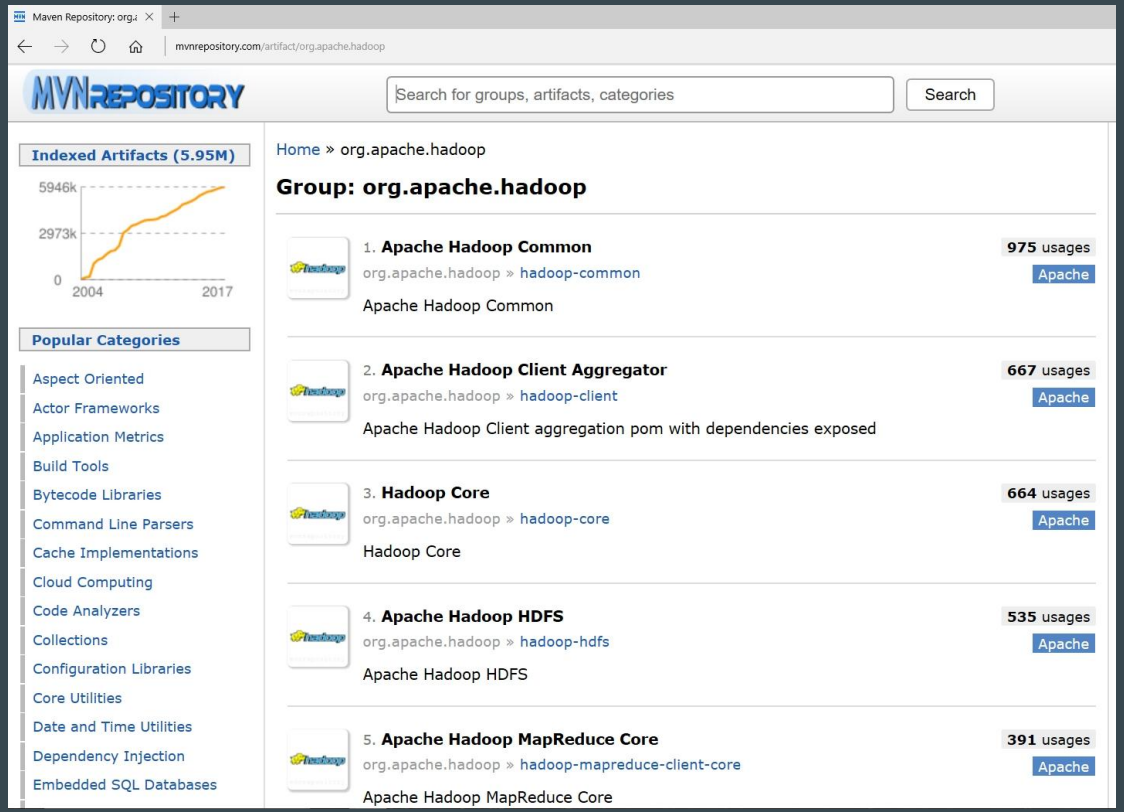
https://hortonworks.com/hadoop-tutorial/introducing-apache-hadoop-developers/

MapReduce Java source code can be copied from above URL

Must add lots of JAR files to project buildpath



http://mvnrepository.com/artifact/org.apache.hadoop



The Jar files can be found here

```
39
40        public static void main(String[] args) throws Exception {
41            Configuration conf = new Configuration();
42
43            Job job = new Job(conf, "WordCount2");
44            job.setJarByClass(WordCount2.class);
45
46            job.setOutputKeyClass(Text.class);
47            job.setOutputValueClass(IntWritable.class);
48
49            job.setMapperClass(Map.class);
```

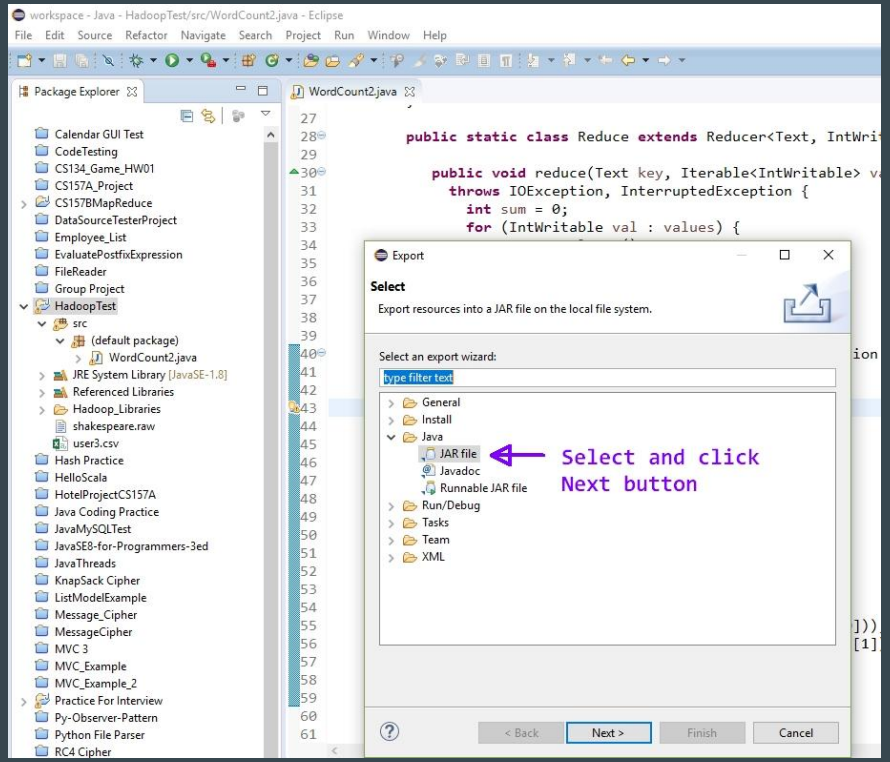
This line is missing from example code. Hadoop finds relevant JAR by looking for the JAR file containing this class.

Problems @ Javadoc Declaration Console

```
<terminated> PostalCodeRating [Java Application] C:\Program Files (x86)\Java\jre1.8.0_111\bin\javaw.exe (Feb 28, 2017, 3:31:54 PM)
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/commons/logging/LogFactory
        at org.apache.hadoop.conf.Configuration.<clinit>(Configuration.java:177)
        at PostalCodeRating.main(PostalCodeRating.java:79)
Caused by: java.lang.ClassNotFoundException: org.apache.commons.logging.LogFactory
        at java.net.URLClassLoader.findClass(Unknown Source)
        at java.lang.ClassLoader.loadClass(Unknown Source)
        at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
        at java.lang.ClassLoader.loadClass(Unknown Source)
        ... 2 more
```

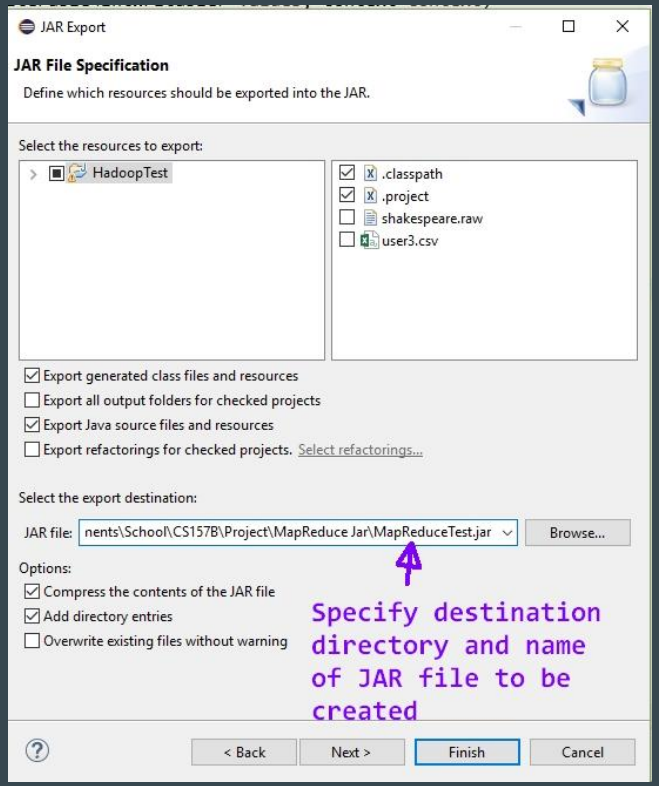
Doesn't run in Eclipse, but the JAR runs in Hadoop

How to make a JAR File in Eclipse



1. Right-Click the project name and select Export
2. Select Jar File and click Next

How to make a JAR File in Eclipse (continued)



3. Tell Eclipse where to create the JAR and give it a name.

4. Click Finish

5. Next, transfer the JAR to the HDFS

How to Run a JAR file (MapReduce Job) in Hadoop

Login to sandbox terminal and use the `jar` Hadoop command.

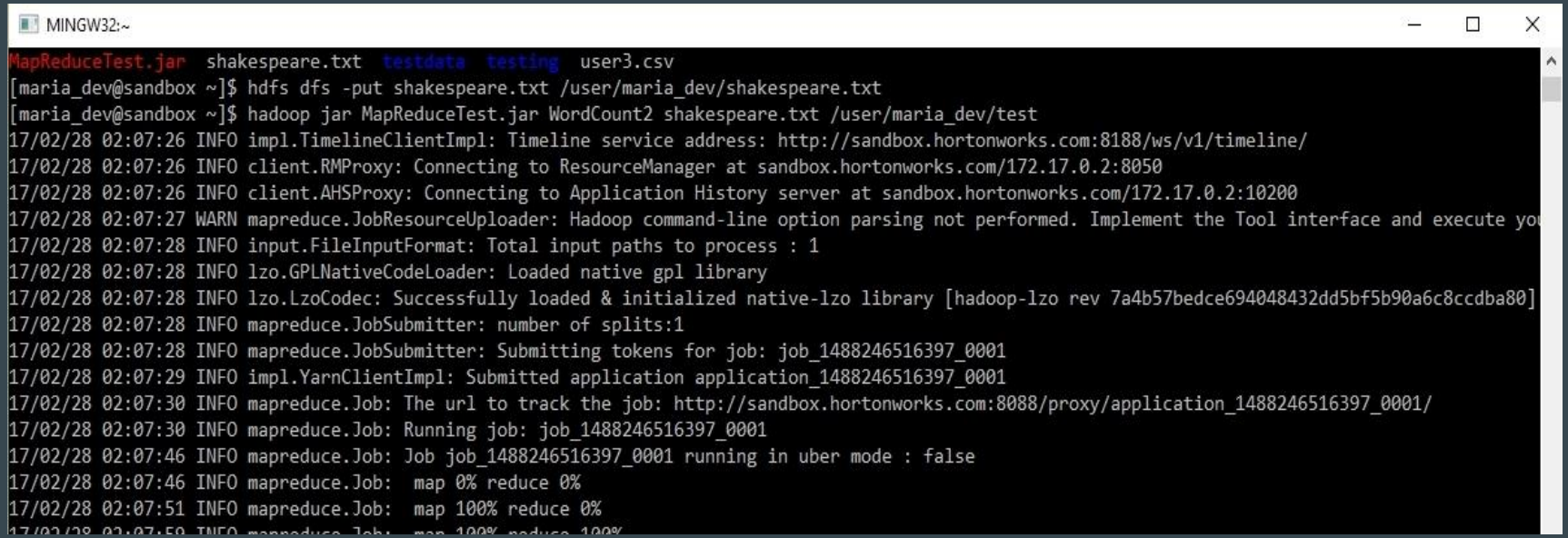
Usage: `hadoop jar <jar_file_name> [mainClass] args…`

```
hadoop jar MapReduceTest.jar WordCount2 shakespeare.txt /user/maria_dev/test
```

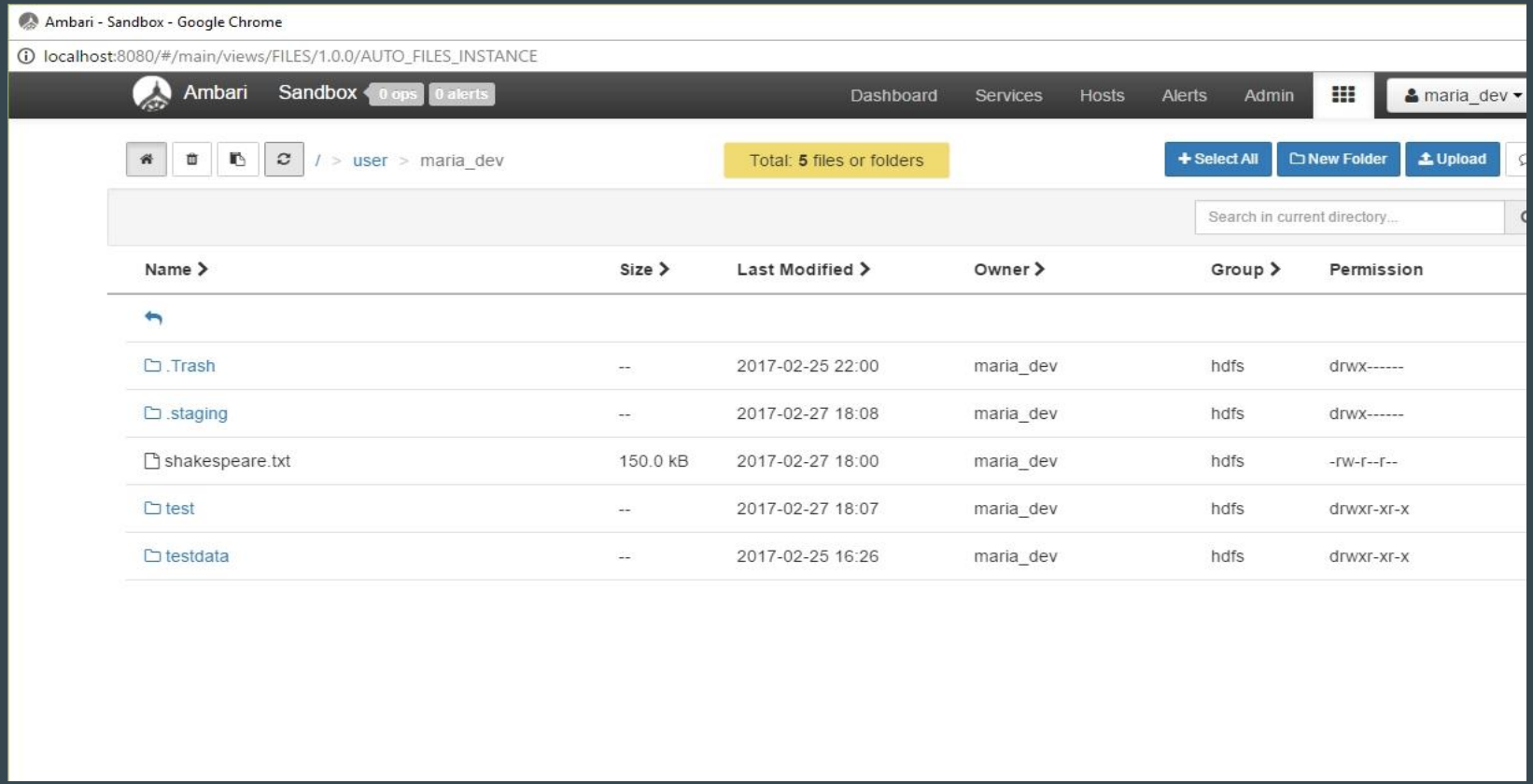
This directory should not already exist

Job will abort if it does. Prevents loss of previous data analysis

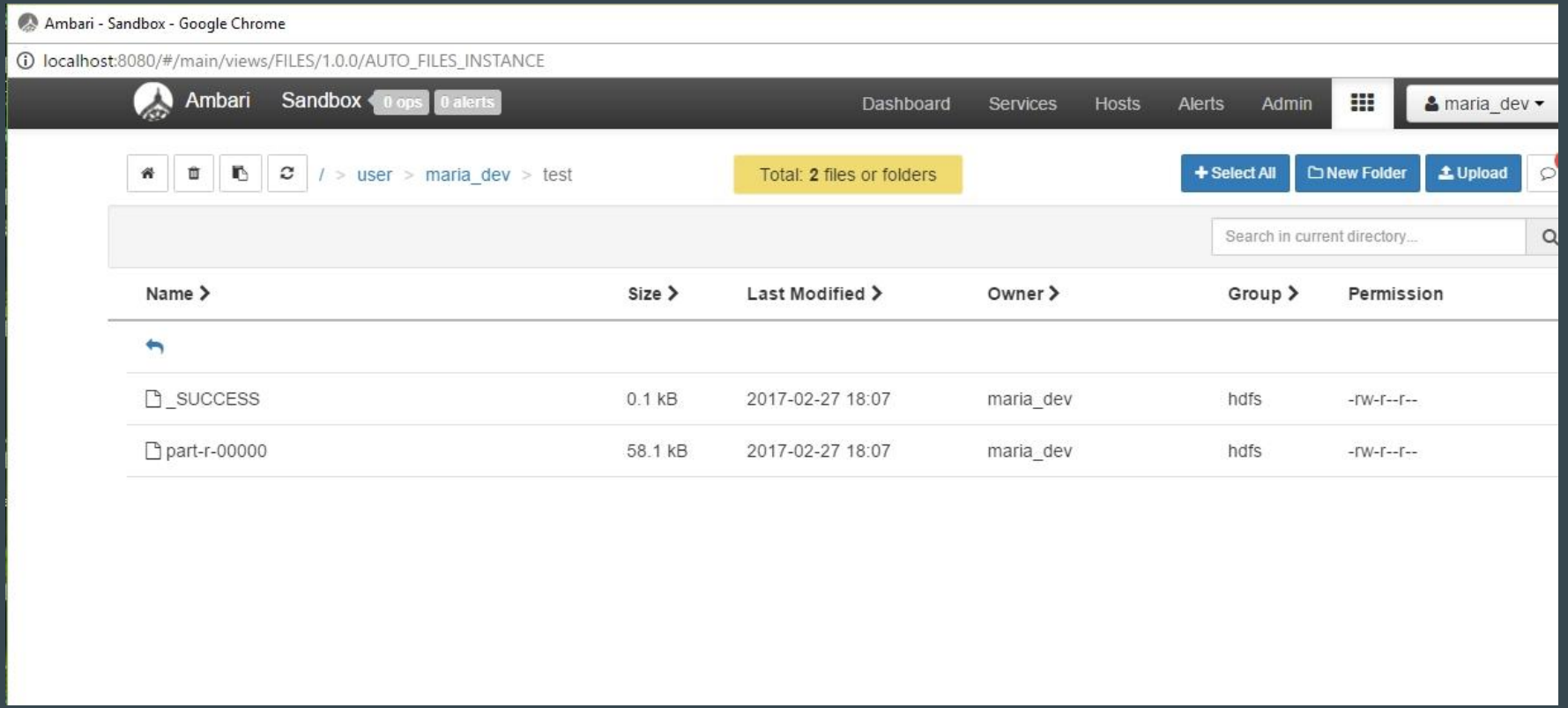
Transfer to HDFS and Run the Test MapReduce Job

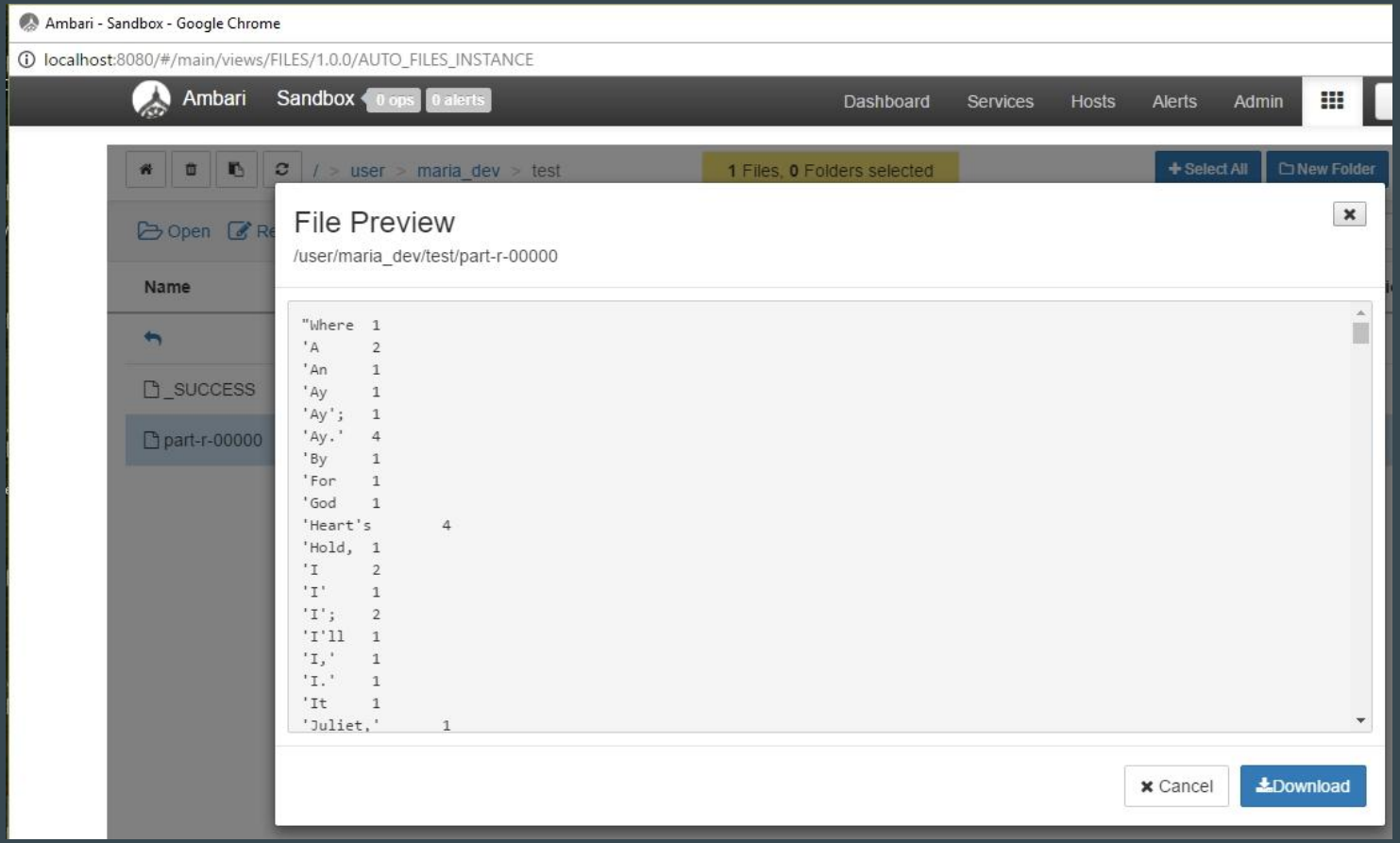


MapReduce output is in test directory



Success only means the test finished




Ambari - Sandbox - Google Chrome
localhost:8080/#/main/views/FILES/1.0.0/AUTO_FILES_INSTANCE
Ambari
Sandbox
0 ops
0 alerts
Dashboard
Services
Hosts
Alerts
Admin
/ > user > maria_dev > test
1 Files, 0 Folders selected
+ Select All
New Folder
Open
Name
_SUCCESS
part-r-00000
File Preview
/user/maria_dev/test/part-r-00000
"Where 1
'A 2
'An 1
'Ay 1
'Ay'; 1
'Ay.' 4
'By 1
'For 1
'God 1
'Heart's 4
'Hold, 1
'I 2
'I' 1
'I'; 2
'I'll 1
'I,' 1
'I.' 1
'It 1
'Juliet,' 1
Cancel
Download

Now implement my own MapReduce job

Coding a MapReduce Job

Problem: Find average review rating for all businesses in a specific zip code

Written as a SQL statement

```
SELECT postal_code, AVG(stars), count(*) AS business_count
FROM business
GROUP BY postal_code
ORDER BY business_count DESC
LIMIT 100;
```

MySQL Results

```
mysql> SELECT postal_code, AVG(stars), count(*) AS business_count
    -> FROM business
    -> GROUP BY postal_code
    -> ORDER BY business_count DESC
    -> LIMIT 100;
+-------------+------------------+----------------+
| postal_code | AVG(stars)       | business_count |
+-------------+------------------+----------------+
| 89109       | 3.5757068452380953 |           2688 |
| 85251       | 3.9410935738444195 |           1774 |
| 85281       | 3.67276814386641   |           1557 |
| 89119       |  3.489802855200544 |           1471 |
...

| 85253       | 3.9275568181818183 |            352 |
| 85248       | 3.690201729106628  |            347 |
| 85051       | 3.3771676300578033 |            346 |
| 89183       | 3.6246376811594203 |            345 |
+-------------+------------------+----------------+
100 rows in set (18.30 sec)
```

Using MapReduce

First break problem into two tasks:

- Map Task
- Reduce Task

Start by observing the data
I wrote a small Java program to see what I’m working with

```java
// MapReduce Job - Find average review rating for all businesses in a specific zip code
public class AvgRatingBusinessZipcode {
    public static final String BUSINESS_CSV_LOCATION =
            "C:\\Users\\General4\\workspace\\YelpMapReduce\\business.csv";

    private static void readInFile(String location) {
        try {
            BufferedReader reader = new BufferedReader(new FileReader(location));
            Scanner in = new Scanner(reader);
            String line = "";
            int lineCount = 0;
            while(in.hasNextLine() && lineCount++ < 10) {
                System.out.println(in.nextLine());
            }
            System.out.println();
            in.close();
        }
        catch(Exception ex) {
            ex.printStackTrace();
```

Must extract data from this mess. Fun, right?

Problems | Javadoc | Declaration | Console

<terminated> AvgRatingBusinessZipcode [Java Application] C:\Program Files (x86)\Java\jre1.8.0_111\bin\javaw.exe (Feb 28, 2017, 10:44:40 AM)

```
"business_id","name","neighborhood","address","city","state","postal_code","latitude","longitude","stars","review_count","is_open","type"
"--6MefnULPED_I942VcFNA","John's Chinese BBQ Restaurant","","328 Highway 7 E, Chalmers Gate 11, Unit 10","Richmond Hill","ON","L4B 3P7","43.840905","-79.39
"--7zmmkVg-IMGaXbuVd0SQ","Primal Brewery","","16432 Old Statesville Rd","Huntersville","NC","28078","35.437086","-80.843688","4","32","1","business"
"--9e1ONYQuAa-CB_Rrw7Tw","Delmonico Steakhouse","The Strip","3355 Las Vegas Blvd S","Las Vegas","NV","89109","36.123183","-115.16919","4","1311","1","busin
"--9QQLMTbFzLJ_oT-ON3Xw","Great Clips","","1835 E Guadalupe Rd, Ste 106","Tempe","AZ","85283","33.3616642","-111.9096233","3","8","1","business"
"--ab39IjZR_xUf81WyTyHg","Famous Footwear","","1800 E Rio Salado Pky 110, Tempe Marketplace","Tempe","AZ","85281","33.4300928","-111.9049649","4","9","1","
"--cgVkbWTiga3OYTkymKqA","Eazor's Auto Salon","","616 Long Rd","Pittsburgh","PA","15235","40.45314","-79.83886","5","6","1","business"
"--cjBEbXMI2obtaRHNSFrA","Howl at the Moon","Downtown","125 7th St","Pittsburgh","PA","15222","40.443888","-80.000223","3","39","1","business"
"--cZ6Hhc9F7VkKXxHMVZSQ","Pio Pio","Dilworth","1408 E Blvd","Charlotte","NC","28203","35.1998528426","-80.8448199047","4","259","1","business"
"--DaPTJW3-tB1vP-PfdTEg","Sunnyside Grill","Corso Italia","1218 Saint Clair Avenue W","Toronto","ON","M6E","43.6778069","-79.4446742","3.5","32","1","busin
```

Zero in on target data

```
18            while(in.hasNextLine() && lineCount++ < 10) {
19                String aLine = in.nextLine();
20                System.out.println(aLine);
21                aLine = aLine.replace("\"", "");
22                String[] colValue = aLine.split(",");
23
24                for(String n : colValue){
25                    System.out.println(n);
26                }
```

Problems @ Javadoc Declaration Console

```
<terminated> AvgRatingBusinessZipcode [Java Application] C:\Program Files (x86)\Java\jre1.8.0_111\bin\javaw.exe (Feb 28, 2017, 4:02:07 PM)
"--cjBEbXMI2obtaRHNSFrA","Howl at the Moon","Downtown","125 7th St","Pittsburgh","PA","15222","40.443888","-80.000223","3","39","1","busine
--cjBEbXMI2obtaRHNSFrA
Howl at the Moon
Downtown
125 7th St
Pittsburgh
PA
15222
40.443888
-80.000223
3
39
1
business
"--cZ6Hhc9F7VkKXxHMVZSQ","Pio Pio","Dilworth","1408 E Blvd","Charlotte","NC","28203","35.1998528426","-80.8448199047","4","259","1","busine
--cZ6Hhc9F7VkKXxHMVZSQ
Pio Pio
Dilworth
1408 E Blvd
Charlotte
```

Index 6

Index 9

Target data always at these indexes

Next write code using Hadoop MapReduce framework

Welcome to Apache™ Hadoop®!

PDF

What Is Apache Hadoop?

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

The project includes these modules:

- **Hadoop Common**: The common utilities that support the other Hadoop modules.
- **Hadoop Distributed File System (HDFS™)**: A distributed file system that provides high-throughput access to application data.
- **Hadoop YARN**: A framework for job scheduling and cluster resource management.
- **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets.

Other Hadoop-related projects at Apache include:

- **Ambari™**: A web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters which includes support for Hadoop HDFS, Hadoop MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig and Sqoop. Ambari also provides a dashboard for viewing cluster health such as heatmaps and ability to view MapReduce, Pig and Hive applications visually alongwith features to diagnose their performance characteristics in a user-friendly manner.
- **Avro™**: A data serialization system.
- **Cassandra™**: A scalable multi-master database with no single points of failure.

Documentation for Hadoop Data Types

Hadoop provides its own set of basic types that are optimized for network serialization

They can be found at: org.apache.hadoop.io package

Hadoop Wrapper Data Types

Hadoop	→	Java	Necessary Import
Text	→	String	import org.apache.hadoop.io.Text;
IntWritable	→	Integer	import org.apache.hadoop.io.IntWritable;
LongWritable	→	Long	import org.apache.hadoop.io.LongWritable;
FloatWritable	→	Float	import org.apache.hadoop.io.FloatWritable;

Inputs and Outputs

		(KEY, VALUE)	(Input Type, Output Type)	
Mapper Input	=	(Line byte offset, Text Line)	(LongWritable, Text)	default
Mapper Output	=	(PostalCode, Stars)	(Text, FloatWritable)	
Reducer Input	=	(PostalCode, Stars)	(Text, FloatWritable)	
Reducer Output	=	(PostalCode, Avg(stars))	(Text, FloatWritable)	

Visualize the function inputs and outputs before coding

Map input < 0, "--ab39IjZR_xUf81WyTyHg","Famous Footwear","","1800 E Rio Salado ... "3","4","1","business" >
...
Map input < 100, "--cgVkbWTiga3OYTkymKqA","Eazor's Auto Salon","","616 Long Rd", ... "4","58","1","business" >

Map output <94582, 3.5 >
Map output <94582, 2.0 >
...
Map output <94582, 3.0 >
Map output <94582, 4.5 >

These will be grouped and sorted before being sent to reducer function

Reducer input < 94582, [3.5, 2.0, ... , 3.0, 4.5] > ← Reducer will get an input like this for each unique postal code

Reducer output 94582, 3.25 ← Entry in output file for each distinct postal code

The Map Function

```
public static class MapperClass
    extends Mapper<Object, Text, Text, FloatWritable> {

    Text postalCode = new Text();
    FloatWritable stars = new FloatWritable();

    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {

        String line = value.toString(); // Convert to Java String

        line = line.replaceAll("\"", ""); // Remove double quotes
        String[] colValue = line.split(","); // Each token separated by comma

        postalCode.set(colValue[6]); // assign the 6th element to postalcode
        try {
            stars.set(Float.valueOf(colValue[9])); // assign 9th element to stars
        }
        catch (NumberFormatException e) {
            e.printStackTrace();
        }
        // Write output key, value pair to the context object
        context.write(postalCode, stars);
    }
}
```

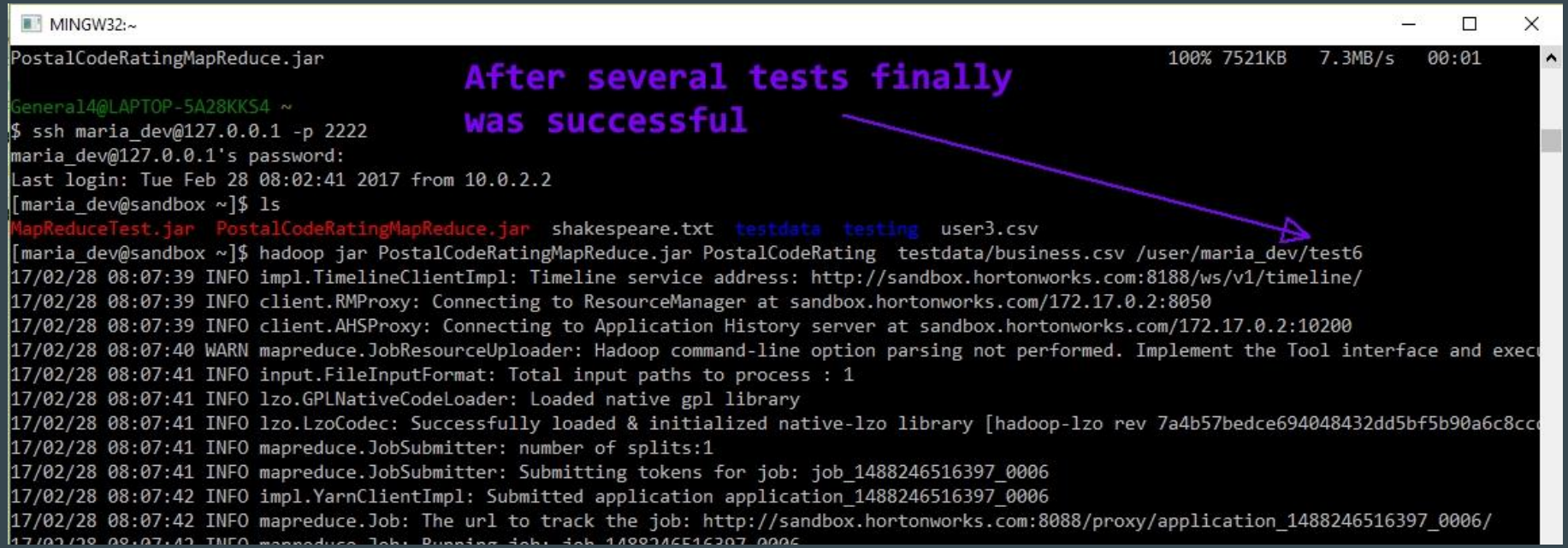
The Reducer

```
public static class ReducerClass
    extends Reducer<Text, FloatWritable, Text, FloatWritable> {

    FloatWritable postalCodeRating = new FloatWritable();

    public void reduce(Text key, Iterable<FloatWritable> values, Context context)
            throws IOException, InterruptedException {
        // Record sum and count of stars
        float sum = 0;
        float count = 0;
        // Sum all the stars that share the same postal code
        for (FloatWritable val : values) {
            sum += val.get();
            count++;
        }
        // Compute the average
        postalCodeRating.set(sum / count);
        // Write output pair
        context.write(key, postalCodeRating);
    }
}
```

The MapReduce Job Runner

```
public static void main(String[] args) throws Exception {
    // Check for correct number of arguments
    if (args.length != 2) {
        System.out.println("usage: [input] [output]");
        System.exit(-1);
    }
    // Create and configure MapReduce job
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "PostalCodeRating");
    job.setJarByClass(PostalCodeRating.class);
    // Set Mapper and Reducer Class
    job.setMapperClass(MapperClass.class);
    job.setReducerClass(ReducerClass.class);
    // Set output types for keys and values
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(FloatWritable.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(FloatWritable.class);
    // Set input and output format class
    job.setInputFormatClass(TextInputFormat.class); //default
    job.setOutputFormatClass(TextOutputFormat.class);
    // Set input and output files from arguments
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    // Send job to cluster and wait for job to complete
    job.waitForCompletion(true);
}
```

Run Job in Hadoop



```
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=15515127
                HDFS: Number of bytes written=136326
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=4697
                Total time spent by all reduces in occupied slots (ms)=4005
                Total time spent by all map tasks (ms)=4697
                Total time spent by all reduce tasks (ms)=4005
                Total vcore-milliseconds taken by all map tasks=4697
                Total vcore-milliseconds taken by all reduce tasks=4005
                Total megabyte-milliseconds taken by all map tasks=1174250
                Total megabyte-milliseconds taken by all reduce tasks=1001250
        Map-Reduce Framework
                Map input records=100001
                Map output records=100001
                Map output bytes=964985
                Map output materialized bytes=1164993
                Input split bytes=137
                Combine input records=0
                Combine output records=0
                Reduce input groups=10560
                Reduce shuffle bytes=1164993
                Reduce input records=100001
                Reduce output records=10560
                Spilled Records=200002
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=422
                CPU time spent (ms)=5110
                Physical memory (bytes) snapshot=326160384
                Virtual memory (bytes) snapshot=3891957760
                Total committed heap usage (bytes)=154664960
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
```

How long did the job take?

I'm not sure, about 10 seconds.

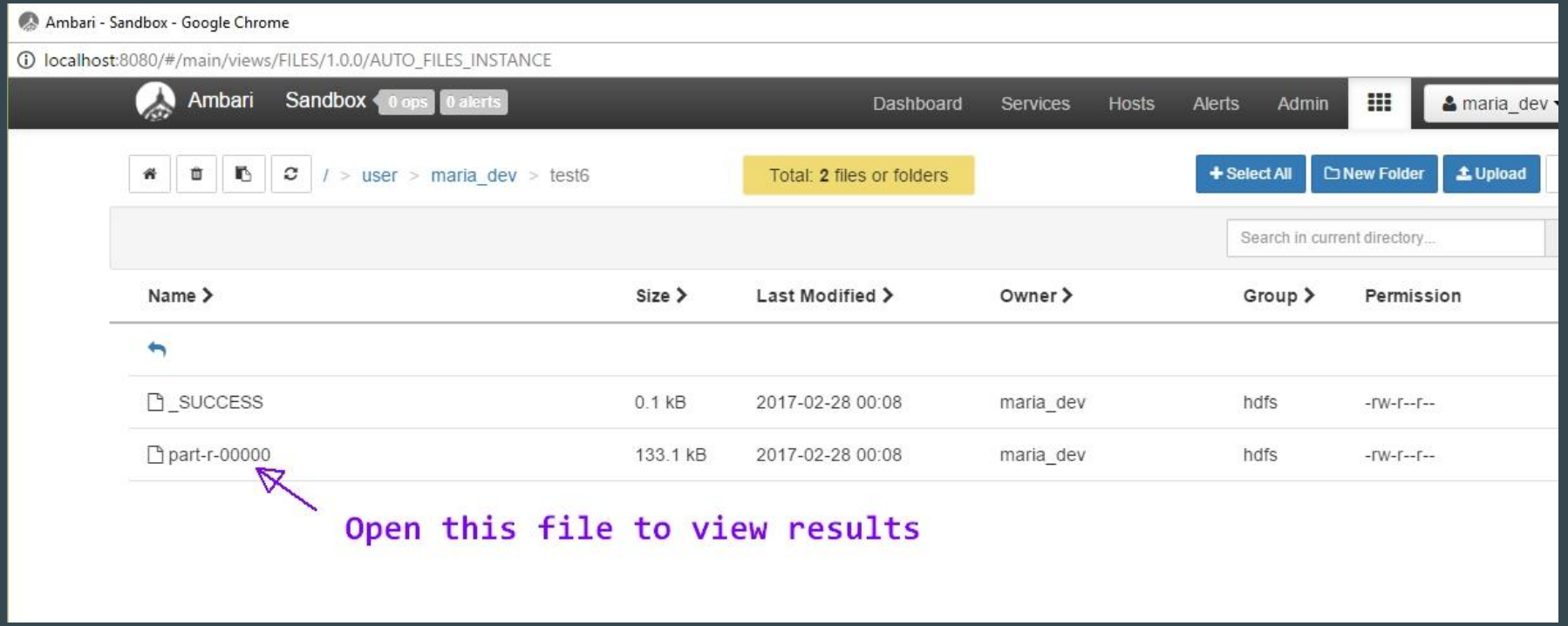
Time spent by all maps 4697 ms

Time spent by all reducers 4005 ms

Default configuration was used for this MapReduce job, which means only a single reducer is used.

Map output data size was less than HDFS block size so only one reduce task required

View the MapReduce Job Results



File Preview

/user/maria_dev/test6/part-r-00000

```
85048   3.5801528
85050   3.840708
85051   3.2222223
85053   3.516892
85054   3.7346938
85058   4.0
85060   4.125
85064   4.5
85066   4.5
85067   3.6666667
85068   4.6666665
85070   4.0
85072   1.0
85073   3.2727273
85075   2.5
85078   4.75
85080   4.1666665
85082   4.5
85083   4.4333334
85085   3.6394231
```

Cancel Download

Improve MapReduce Performance

- Increase the number of reducers for the job. But, why?
  - It makes the reducer phase shorter because you get more parallelism
  - Hadoop does this automatically if the map output > HDFS block size
  - Number of reducers can be set manually (shown in next slide)
- Map jobs should run longer than the reducer
  - If map job time is greater than or equal, that is a sign the MapReduce job is not correctly optimized, or input data is very small (as was the case with my example).
  - Remember, map output gets partitioned. The number of partitions is equal to the number of reducers. Very large map output increases the number of partitions.

Improve MapReduce Performance (continued)

- Does very large Map input data mean Hadoop will spawn more reducers?
  - No! The **output** of the map job governs the number of reducers, not the map input data
  - Depending on the query, map functions may produce relatively small output data
  - Larger map input data only means more map tasks will be created by Hadoop.
- How do you set the number of reducers?
  - The MapReduce API will try and derive the correct number of reducers to use
  - However, you can set it manually `job.setNumReduceTasks(25);`
  - Too many reduce tasks can cause problems too. You must fine tune the amount.

Improve MapReduce Performance (continued)

- How many reducers do you choose?
  - One, rule of thumb is to aim for reducers that each run for about 5 minutes, and which produce at least one HDFS block's worth of output (This assumes you're working with 'Big Data')

- Remember, input data to Map is expected to be much larger than my example
  - Increasing the number of reducers for my example would be a bad idea since my input data was only 14 mb, well below the 128 mb block size.

Example of what not to do

My input data was very small but I will increase my reducers to 25 anyway

I added the following code to my job runner in the main function:

```
job.setNumReduceTasks(25);
```

```
I then transferred the JAR file to hadoop and ran the job
again. Let's see if I truly killed MapReduce performance.
```

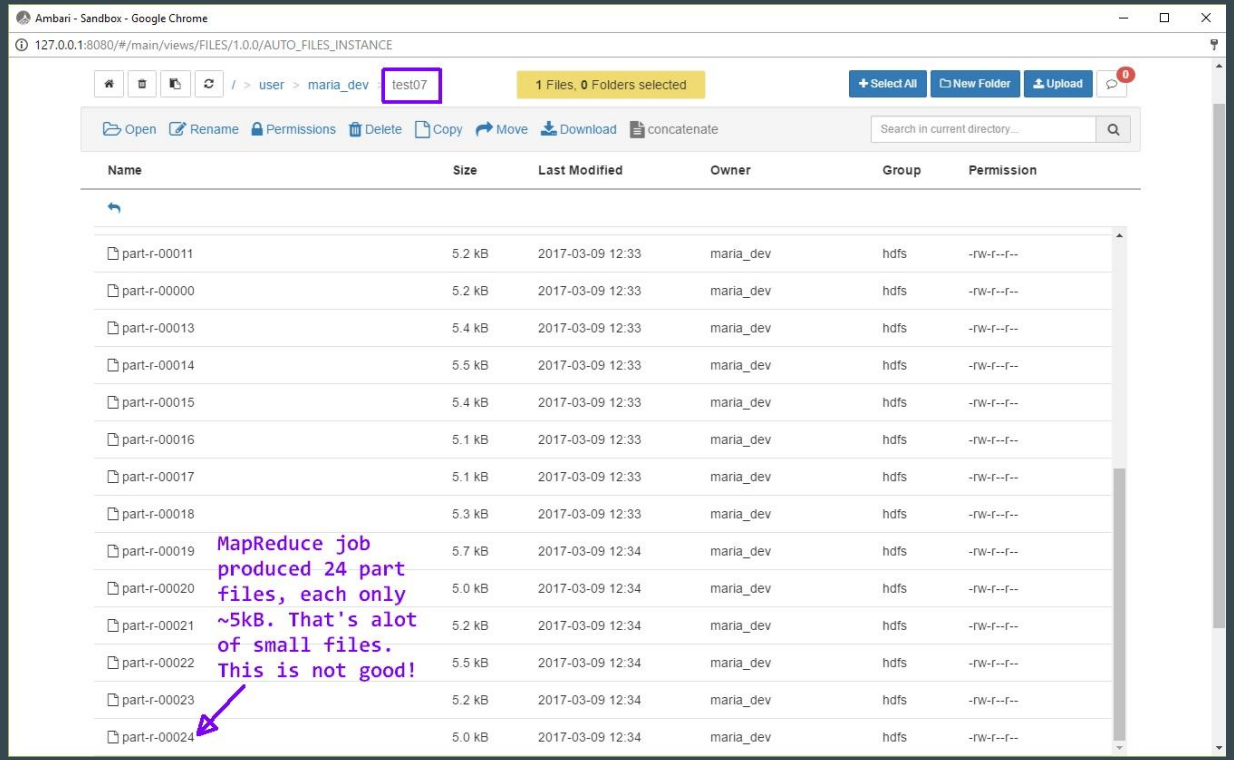
Example of what not to do (continued)

```
        HDFS: Number of write operations=50
Job Counters
        Launched map tasks=1
        Launched reduce tasks=25
        Data-local map tasks=1
        Total time spent by all maps in occupied slots (ms)=5486
        Total time spent by all reduces in occupied slots (ms)=403716
        Total time spent by all map tasks (ms)=5486
        Total time spent by all reduce tasks (ms)=403716
        Total vcore-milliseconds taken by all map tasks=5486
        Total vcore-milliseconds taken by all reduce tasks=403716
        Total megabyte-milliseconds taken by all map tasks=1371500
        Total megabyte-milliseconds taken by all reduce tasks=100929000
Map-Reduce Framework
        Map input records=100001
        Map output records=100001
        Map output bytes=964985
        Map output materialized bytes=1165137
        Input split bytes=137
        Combine input records=0
        Combine output records=0
        Reduce input groups=10560
        Reduce shuffle bytes=1165137
        Reduce input records=100001
        Reduce output records=10560
        Spilled Records=200002
```

Now I have 25 Reducers

Reducers ran 5X longer than the mapper

Example of what not to do (continued)



Job produced 24 part files, each of which is ~ 5kB, well below the default block size of 128 mb.

That is a lot of unnecessary disk operations

When running a MapReduce job on actual ‘Big Data’ - make sure the part files are approximately the size of the HDFS block files

Scaling Out Example

Now, I will demonstrate a MapReduce job that works on much larger input data

Query:

Find all businesses that have over 1000 reviews and have a star rating greater than 4

Input data is review3.txt, which is 2.1 GB

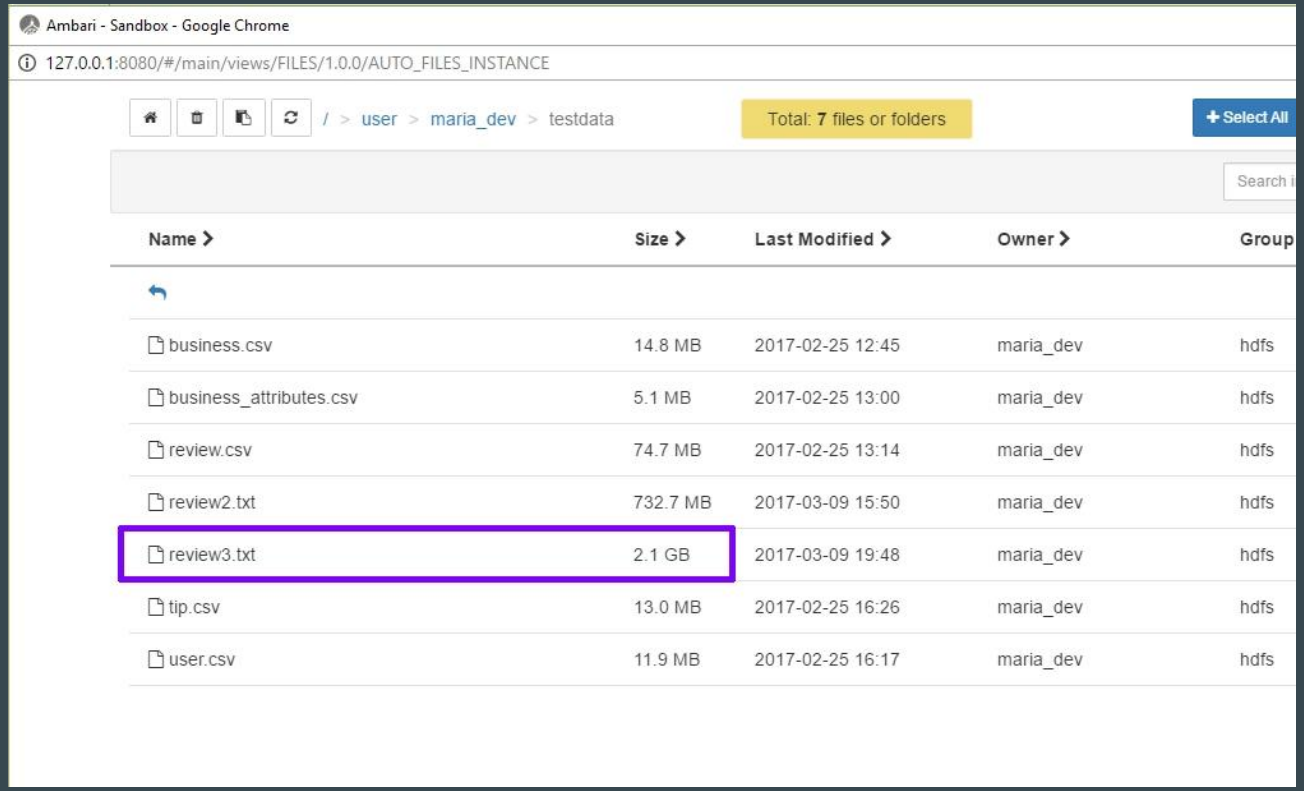
Since my input is larger, I expect Hadoop to spawn more map tasks for increased parallelism

Scaling Out Example (continued) MySQL Solution

```
mysql> SELECT business_id, AVG(stars) AS starAVG, COUNT(*) AS reviewCount
    -> FROM review
    -> GROUP BY business_id
    -> HAVING reviewCount > 1000 AND starAVG > 4 LIMIT 25;
+------------------------+---------+-------------+
| business_id            | starAVG | reviewCount |
+------------------------+---------+-------------+
| --9e1ONYQuAa-CB_Rrw7Tw |  4.0953 |        1311 |
| -ed0Yc9on37RoIoG2ZgxBA |  4.0045 |        1117 |
| 0FUtlsQrJI7LhqDPxLumEw |  4.1688 |        1410 |
| 0NmTwqYEQiKErDv4a55obg |  4.1138 |        1028 |
| 0W4lkclzZThpx3V65bVgig |  4.0674 |        1603 |
| 2iTsRqUsPGRH1li1WVRvKQ |  4.3959 |        1225 |
| 3BCsAgo_1i4xMuTyLKMLRQ |  4.2906 |        1311 |
| 3GEEy7RP6e4bT4LAiWFMFQ |  4.2161 |        1462 |
| 3kdSl5mo9dWC4clrQjEDGg |  4.4659 |        2303 |
| 3154GTr8-E3XPbIxnF_sAA |  4.2856 |        1390 |
| 3N9U549Zse8UP-MwKZAjAQ |  4.2307 |        1036 |
| 4JNXUYY8wbaaDmk3BPzlWw |  4.1140 |        6414 |
| 4k3R1MAMd46DZ_JyZU0lMg |  4.1647 |        1008 |
| 5shgJB7a-2_gdnzc0gsOtg |  4.2727 |        1313 |
| 7sb2FYLS2sejZKxRYF9mtg |  4.4955 |        1443 |
| 7sPNbCx7vGAaH7SbNPZ6oA |  4.0658 |        2735 |
| 9a3DrZvpYxVs3k_qwlCNSw |  4.2656 |        1457 |
| A-uZAD4zP3rRxb44WUGV5w |  4.6825 |        1134 |
| aLcFhMe6DDJ430zelCpd2A |  4.0323 |        1145 |
| awI4hHMfa7H0Xf0-ChU5hg |  4.4437 |        1785 |
| BH9z7IJ4zydAqgwsbqoVZQ |  4.3463 |        1311 |
| CiYLq33nAyghFkUR15pP-Q |  4.3154 |        1151 |
| d10IxZPirVJlOSpdRZJczA |  4.3515 |        1391 |
| DkYS3arLOhA8si5uUEmHOw |  4.2668 |        4655 |
| eLFfWcdb7VkqNyTONksHiQ |  4.2761 |        1619 |
+------------------------+---------+-------------+
25 rows in set (36.95 sec)
```

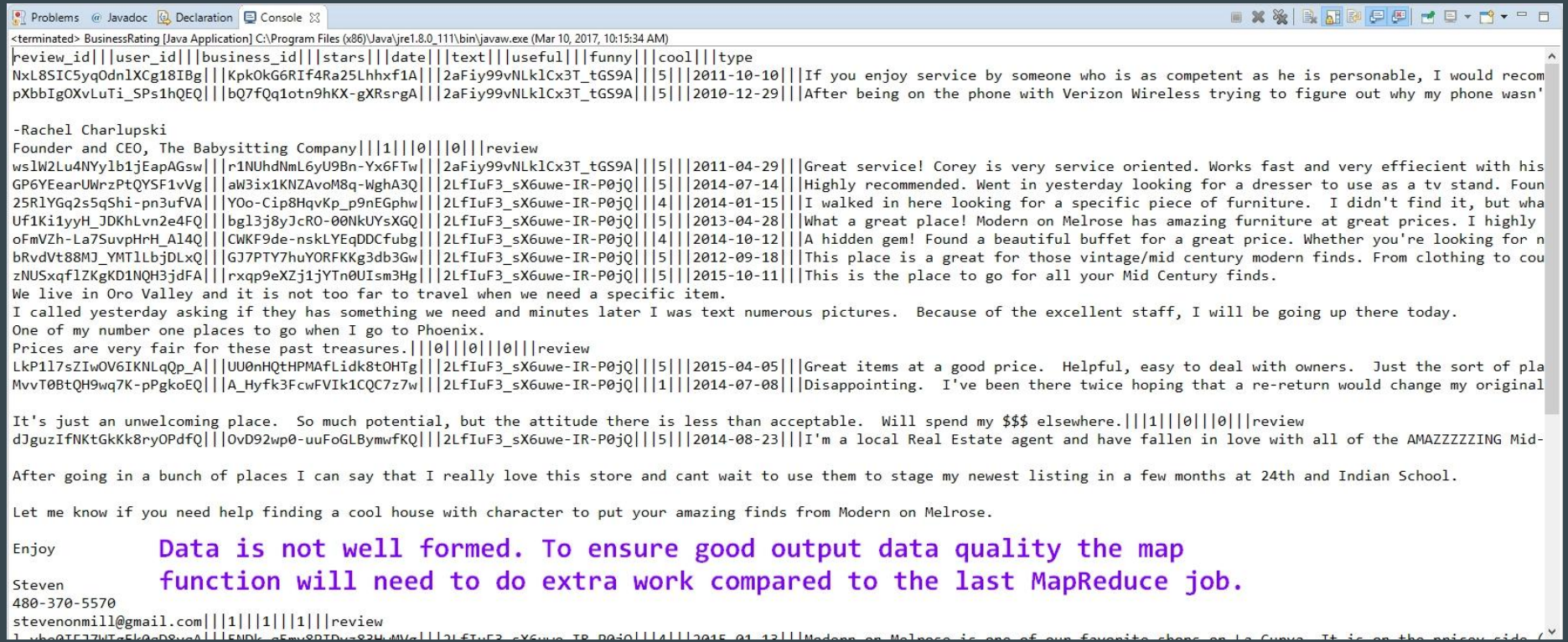
As before, here how you would solve the query using MySQL

Scaling Out Example (continued)



The input Data is 2.1 GB

Scaling Out Example (continued)



Scaling Out Example (continued)

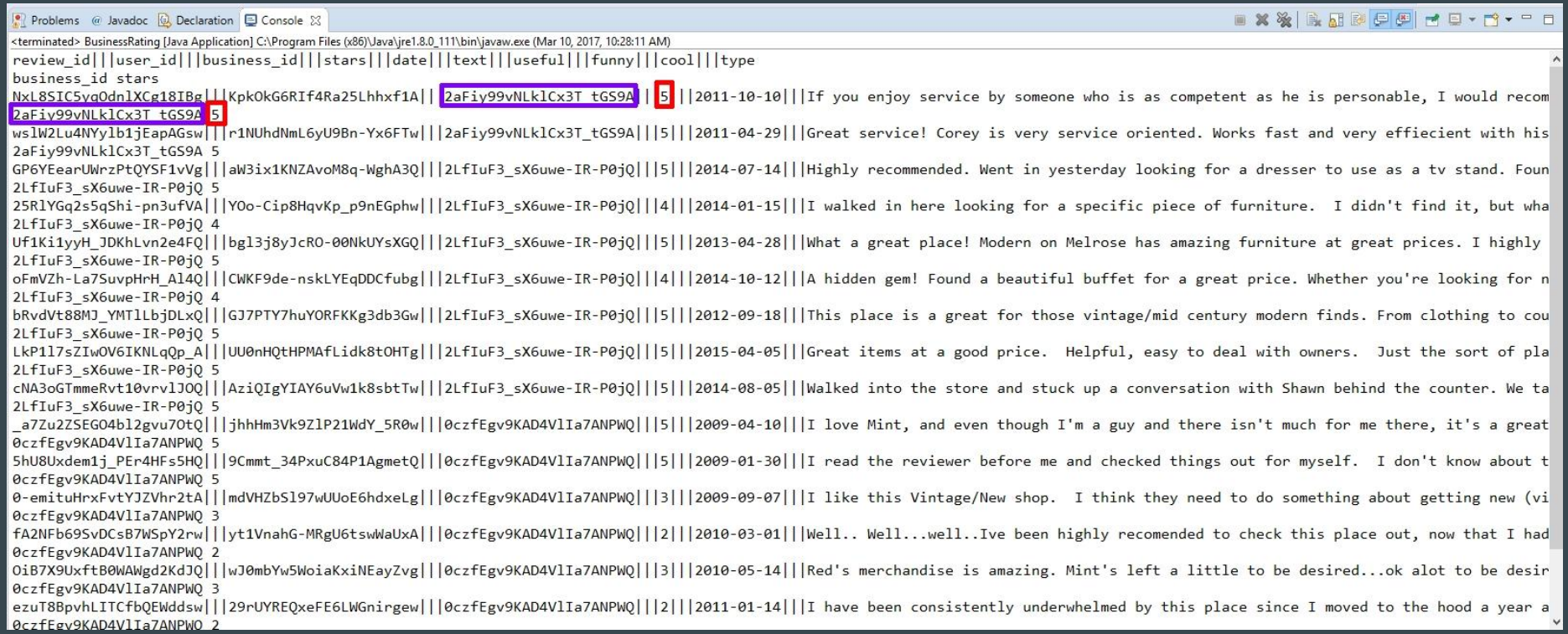
The values is review3.txt were delimited by three pipe symbols |||

First, I identified the target data I would need - business_id, stars

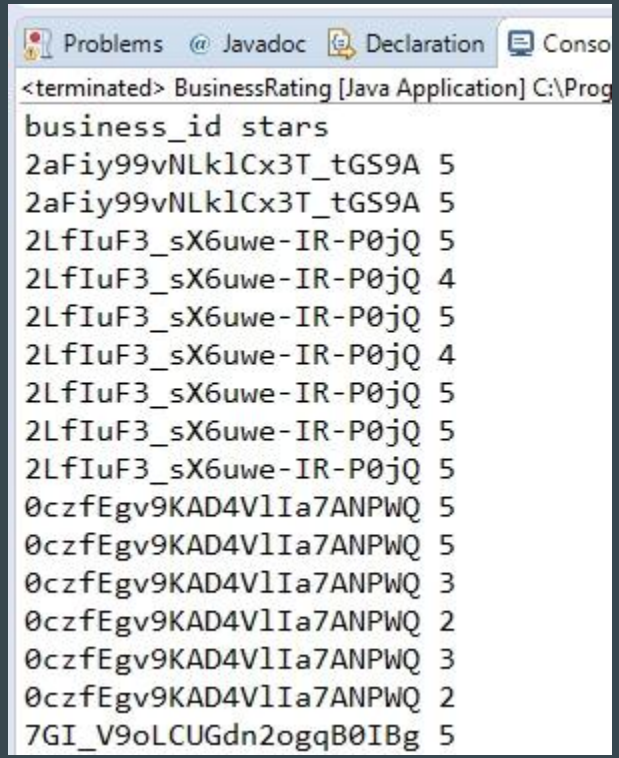
I wrote a java program to help zero in on the target data

I choose only those lines that were well formed and throughout the rest

Scaling Out Example (continued) Zero in on target data



Scaling Out Example (continued)



I captured the target data.

This is what my Map function will need to do

Now it is time to write code using the MapReduce API

Scaling Out Example (continued) The Map Function

```
    Text business_id = new Text();
    FloatWritable stars = new FloatWritable();

    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {

        String line = value.toString(); // Convert to Java String

        line = line.replace("\"", ""); // Remove double quotes
        String delimiters = "(\\|\\|\\|)"; // Column values separated by |||
        String[] colValue = line.split(delimiters); // Place values is an array

        // The array should have a size of 10 if it is correct
        int arraySize = colValue.length;

        // extract the third and fourth elements which are
        // business_id and stars respectively
        if(arraySize == 10){
            business_id.set(colValue[2]); // assign the 3rd element to business_id
            try {
                stars.set(Float.valueOf(colValue[3])); // assign 4th element to stars
            }
            catch (NumberFormatException e) {
                e.printStackTrace();
            }
        }

        // Write output key, value pair to the context object
        context.write(business_id, stars);
    }
}
```

Scaling Out Example (continued) The Reduce Function

```
FloatWritable business_rating = new FloatWritable();

public void reduce(Text key, Iterable<FloatWritable> values, Context context)
        throws IOException, InterruptedException {
    // Record sum and count of stars
    float sum = 0;
    float count = 0;
    // Sum all the stars that share the same postal code
    for (FloatWritable val : values) {
        sum += val.get();
        count++;
    }
    // Compute the average
    float avgRating = (sum/count);

    // Only want business with over 1000 reviews and a rating > 4
    if(count > 1000 && avgRating > 4){
        business_rating.set(avgRating);
        // Write output pair
        context.write(key, business_rating);
    }
}
```

Scaling Out Example (continued) The Job Runner

```java
// The Driver Code
public static void main(String[] args) throws Exception {
    // Check for correct number of arguments
    if (args.length != 2) {
        System.out.println("usage: [input] [output]");
        System.exit(-1);
    }
    // Create and configure MapReduce job
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "BusinessRating");
    job.setJarByClass(BusinessRating.class);
    // Set Mapper and Reducer Class
    //job.setNumReduceTasks(25); // Explicitly set number of reducers
    job.setMapperClass(MapperClass.class);
    job.setReducerClass(ReducerClass.class);
    // Set output types for keys and values
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(FloatWritable.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(FloatWritable.class);
    // Set input and output format class
    job.setInputFormatClass(TextInputFormat.class); //default
    job.setOutputFormatClass(TextOutputFormat.class);
    // Set input and output files from arguments
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    // Send job to cluster and wait for job to complete
    job.waitForCompletion(true);
}
```

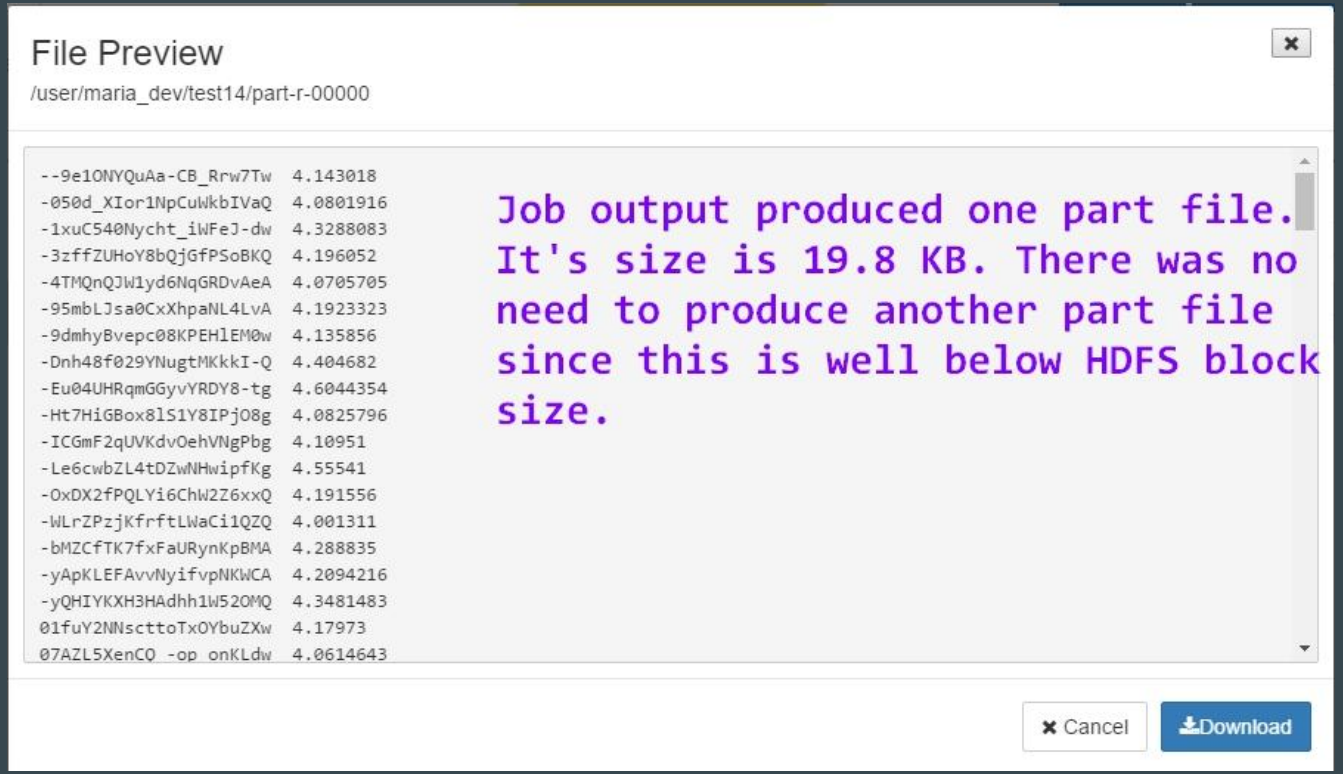
Scaling Out Example (continued) The Results

```
File System Counters
        FILE: Number of bytes read=310473543
        FILE: Number of bytes written=623702148
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=2301147471
        HDFS: Number of bytes written=20238
        HDFS: Number of read operations=57
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
Job Counters
        Launched map tasks=18
        Launched reduce tasks=1
        Data-local map tasks=18
        Total time spent by all maps in occupied slots (ms)=837885
        Total time spent by all reduces in occupied slots (ms)=78331
        Total time spent by all map tasks (ms)=837885
        Total time spent by all reduce tasks (ms)=78331
        Total vcore-milliseconds taken by all map tasks=837885
        Total vcore-milliseconds taken by all reduce tasks=78331
        Total megabyte-milliseconds taken by all map tasks=209471250
        Total megabyte-milliseconds taken by all reduce tasks=19582750
Map-Reduce Framework
        Map input records=10706094
        Map output records=10706094
        Map output bytes=289061337
        Map output materialized bytes=310473633
        Input split bytes=2448
        Combine input records=0
```

Since input data was much larger this time, it was divided into block size chuncks - One Map Task per block

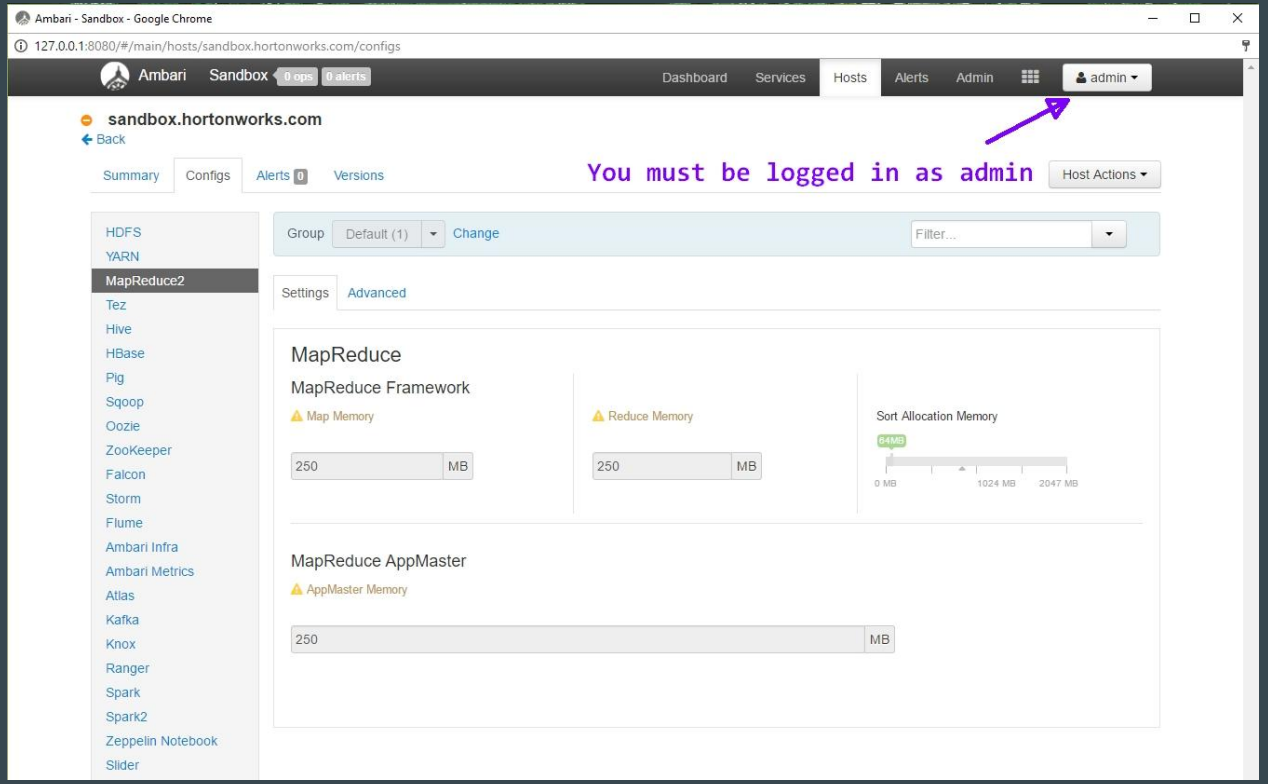
Better time ratio

Scaling Out Example (continued) The Results



Output file shows all the businesses that have over 1000 reviews and have a rating greater than 4

Boosting Performance Using Ambari GUI



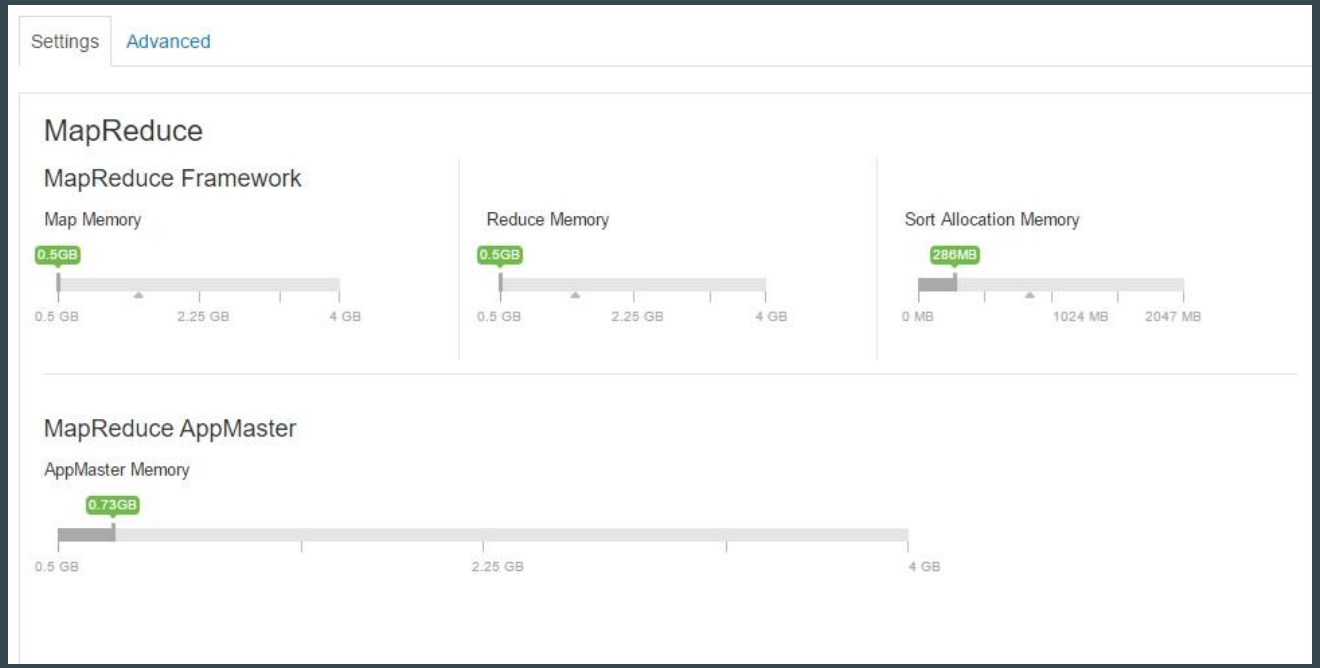
You must login to Ambari as the administrator

Notice that all the MapReduce memory settings are at the default of 250 MB.

This value is below the recommended value.

I will boost performance by increasing this value into the recommended range.

Boosting Performance Using Ambari GUI (continued)



Now the settings are in the recommended range.

I had to restart the MapReduce service and then I ran my MapReduce job again.

Boosting Performance Using Ambari GUI (continued)

```
        HDFS: Number of write operations=2
Job Counters
        Launched map tasks=18
        Launched reduce tasks=1
        Data-local map tasks=18
        Total time spent by all maps in occupied slots (ms)=774192
        Total time spent by all reduces in occupied slots (ms)=82228
        Total time spent by all map tasks (ms)=774192
        Total time spent by all reduce tasks (ms)=82228
        Total vcore-milliseconds taken by all map tasks=774192
        Total vcore-milliseconds taken by all reduce tasks=82228
        Total megabyte-milliseconds taken by all map tasks=193548000
        Total megabyte-milliseconds taken by all reduce tasks=20557000
Map-Reduce Framework
        Map input records=10706094
        Map output records=10706094
        Map output bytes=289061337
        Map output materialized bytes=310473633
        Input split bytes=2448
        Combine input records=0
        Combine output records=0
        Reduce input groups=110740
        Reduce shuffle bytes=310473633
        Reduce input records=10706094
        Reduce output records=624
        Spilled Records=21412188
        Shuffled Maps =18
        Failed Shuffles=0
        Merged Map outputs=18
        GC time elapsed (ms)=45055
        CPU time spent (ms)=213130
        Physical memory (bytes) snapshot=3486101504
        Virtual memory (bytes) snapshot=36732022784
        Total committed heap usage (bytes)=2268594176
Shuffle Errors
```

CPU time spent before optimization 213130 ms

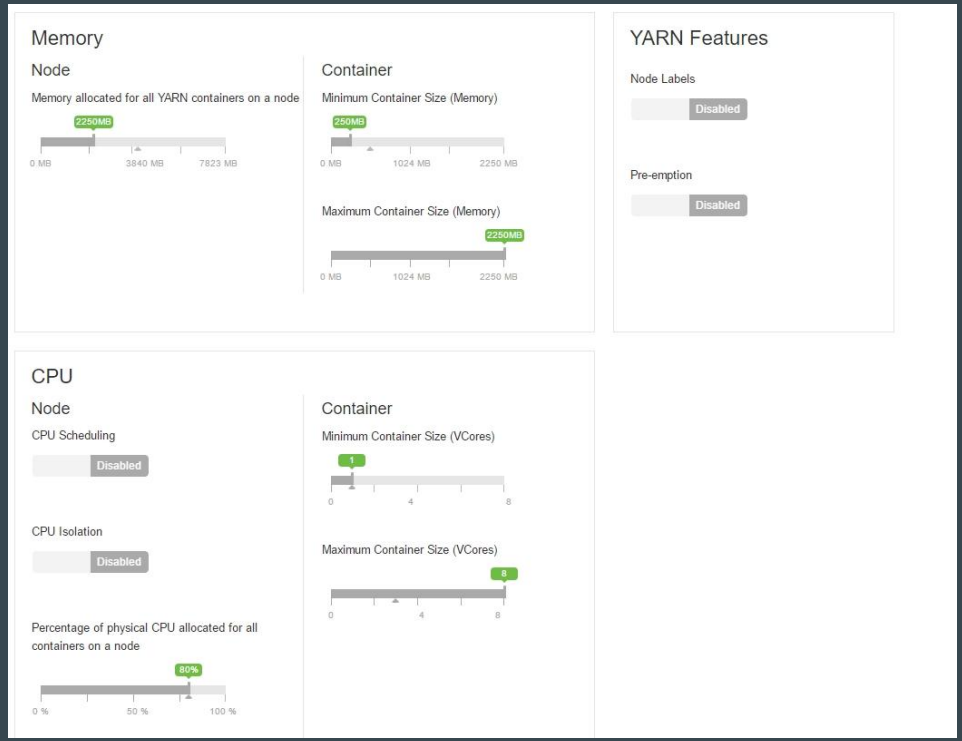
Boosting Performance Using Ambari GUI (continued)

```
        Total megabyte-milliseconds taken by all map tasks=115046912
        Total megabyte-milliseconds taken by all reduce tasks=32114688
    Map-Reduce Framework
        Map input records=10706094
        Map output records=10706094
        Map output bytes=289061337
        Map output materialized bytes=310473633
        Input split bytes=2448
        Combine input records=0
        Combine output records=0
        Reduce input groups=110740
        Reduce shuffle bytes=310473633
        Reduce input records=10706094
        Reduce output records=624
        Spilled Records=21412188
        Shuffled Maps =18
        Failed Shuffles=0
        Merged Map outputs=18
        GC time elapsed (ms)=9178
        CPU time spent (ms)=168230
        Physical memory (bytes) snapshot=9801912320
        Virtual memory (bytes) snapshot=53080670208
        Total committed heap usage (bytes)=9102688256
    Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
    File Input Format Counters
        Bytes Read=2301145023
    File Output Format Counters
        Bytes Written=20238
```

Previous 213130 ms
Current 168230 ms
22% Faster!

Wow, a small change caused a big boost in performance!

Scaling UP



Minimum virtual cores = 1

I will increase to 7 and run the job again.

Let’s see if this improves performance of the MapReduce job

Scaling Up

I increased the VCores amount.

Now Minimum VCores = 7

Scaling Up - After increasing min. VCore amount

```
        Total vcore-milliseconds taken by all reduce tasks=61572
        Total megabyte-milliseconds taken by all map tasks=115852288
        Total megabyte-milliseconds taken by all reduce tasks=31524864
Map-Reduce Framework
        Map input records=10706094
        Map output records=10706094
        Map output bytes=289061337
        Map output materialized bytes=310473633
        Input split bytes=2448
        Combine input records=0
        Combine output records=0
        Reduce input groups=110740
        Reduce shuffle bytes=310473633
        Reduce input records=10706094
        Reduce output records=624
        Spilled Records=21412188
        Shuffled Maps =18
        Failed Shuffles=0
        Merged Map outputs=18
        GC time elapsed (ms)=8283
        CPU time spent (ms)=165680
        Physical memory (bytes) snapshot=9689694208
        Virtual memory (bytes) snapshot=53111750656
        Total committed heap usage (bytes)=9175564288
Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
File Input Format Counters
```


Before Optimization 168230 ms
After Optimization 165680 ms
Negligible improvement due to relatively small MapReduce job

Improvements

Write a **Combiner** function as an optimization

The combiner function would work on the output of the Map function

It could sum the star ratings and feed reducer the sum and number of businesses

Optimized reducer input < 94582, [345, 75] >

Sum of stars

number of businesses

The reducer would only have to compute the average for each postal code

Maintain Data Quality

- My MapReduce job worked on clean, historical data
- At Enterprise level, new ‘dirty’ data coming in all the time
- Remember, garbage in = garbage out
- Veracity - the 4th V of Big Data
- Turn Big Data into smaller data by filtering and cleaning
- Preprocessing data before inputting into MapReduce job ensures high quality data output

How do you maintain data integrity?

- Preprocessing (cleaning, filtering) does not necessarily ensure data integrity
- Can't impose tuple based constraints or attribute constraints like with MySQL
- Solution -
  - Determine the data type violations that may cause data integrity problems
  - Further parse data in Map function to check for data type violations

Example: In the map function it may be a good idea to check that the star ratings are numeric and non-negative.

Other examples: In map function, by using regular expressions check that a phone number, zip code, or age attributes are valid.

Optimization Using File Compression

Two benefits of using file compression:

1. Reduces the space required to store data files
2. Speeds up data transfer across network or reading to/from disk

If Input Data Files are Compressed

- The bytes read from HDFS is reduced
- They will be decompressed automatically by MapReduce job

Infrequently Viewed Job Output Data Should Be Stored In Compressed Format

- Saves disk space

Compress Map Output Data (Intermediate Compression)

- Jobs run faster by reducing the volume of data transferred to the Reducer Functions

Warning!

Use a compression format the supports file splitting!

If you do not the MapReduce job will be very inefficient.

Data locality optimization will be lost!

To Enable gzip Map Output Compression (intermediate compression)

In the main function and the following to the configuration settings:

```
Configuration conf = new Configuration();

conf.setBoolean(Job.MAP_OUTPUT_COPRESS, true);

conf.setClass(Job.MAP_OUTPUT_COMPRESS_CODEC, GzipCodec.class,  \

    CompressionCodec.class);

Job job = new Job(conf);
```

What I learned

Writing MapReduce jobs is not easy!

Task decomposition can be tricky

Understand the inputs and outputs before coding

Functional program thinking

You must remember what Hadoop does for you (grouping and sort)

I’d prefer to work at a higher-level of abstraction

References

- Hortonworks Sandbox
  - Getting Started with HDF Sandbox
  - Learning the Ropes of the Hortonworks Sandbox
  - Manage Files on HDFS via CLI/Ambari Files View
- MapReduce
  - Introducing Apache Hadoop to Java Developers
  - Apache's MapReduce Tutorial
  - Apache Hadoop API Documentation
  - Hadoop The Definitive Guide (4th Edition) by Tom White ← Very Good Book