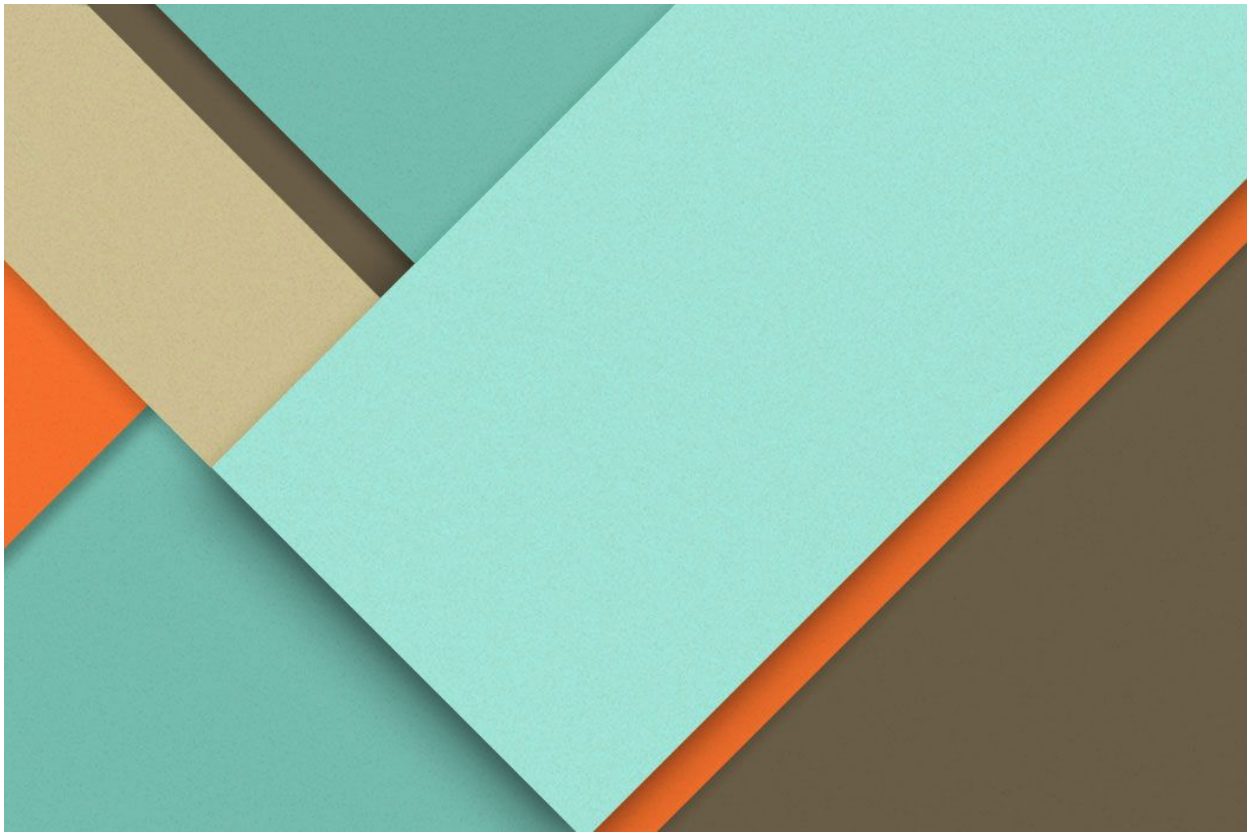


Project 2

12.07.2016

Ms. Saatchi Nandwani | Mr. Akshat Sehgal

MGS 655 Distributed Computing and Big Data

Team Composition and Contributions

TASK	CONTRIBUTOR
CCR Cluster Setup	Saatchi
Gathering Inputs	Akshat/Saatchi
Map Reduce Program to compute Trigrams	Akshat
Empirical Results and Analysis	Saatchi
Project Report	Akshat/Saatchi

Connection to CCR Cluster

1. We updated the HDFS NameNode, Resource Manager and MapReduce JobHistory hosts and port numbers in the configuration files namely *core-site.xml, hdfs-site.xml, yarn-site.xml and mapred-site.xml*.
2. In addition to this, we had to setup a VPN connection using *Cisco AnyConnect* to access the host *vpn.buffalo.edu* using the group *UBVPN*.
   We performed our mapreduce job using an *ubuntu 16.04 system*. So we used an open source version to establish the VPN connection named *OpenConnect*.

Data Sets

We are dealing with 39 articles as we get sufficient number of trigrams for our empirical analysis. We decided this number on the basis of last two digits of the Person ID 50198939.

We chose an online text file merging utility for merging all individual articles into one file: http://www.ofoct.com/merge-text-files-online

As per the project requirements, we replicated 2 individual articles and merged them manually to our final input file. (the same we used for Project 1)

Design Issues

Setting the Number of Mappers

We were not able to make YARN assign multiple mappers to our MapReduce job. However, we were able to set the number of reducers.

Reason

The Default Block Size of the Hadoop Configuration deployed on CCR is 128 MB. Our final input file size is 762 kb in size which is very much lesser than the Block Size of the Hadoop Configuration deployed on CCR.

Hence YARN, deploys only 1 mapper for our MapReduce job.

How we reached to this conclusion?

1. We tried to change the number of mappers in code using standard hadoop api as follows, however the API to set number of mappers in the following manner is deprecated in newer version of Hadoop:

   Job.setNumMapTasks = 2

2. We tried changing the number of mappers using the command line by adding the following parameter to the *jar* command:

   –D mapreduce.job.maps=2

3. Its mentioned in the Hadoop Documentation that the number of splits in the file govern the number of mappers deployed. We tried modifying the split size of the file using the command line by adding the below parameter to the jar command:

   -Dmapreduce.input.fileinputformat.split.minsize
   -Dmapreduce.input.fileinputformat.split.maxsize

How we handled this issue?

We executed the MapReduce code using 1 Reducer first and then using 2 Reducers. We did not set/change the number of mappers

We compared the results/run-times for these cases to answer Part 4b in the grading criteria.

Empirical Results and Discussions

Brief Summary of Results

Our Map Algorithm converts the input file into a string and splits this string using a regular expression. As per our algorithm, we are counting special characters along with the standard space as delimiters to count only english language trigrams.

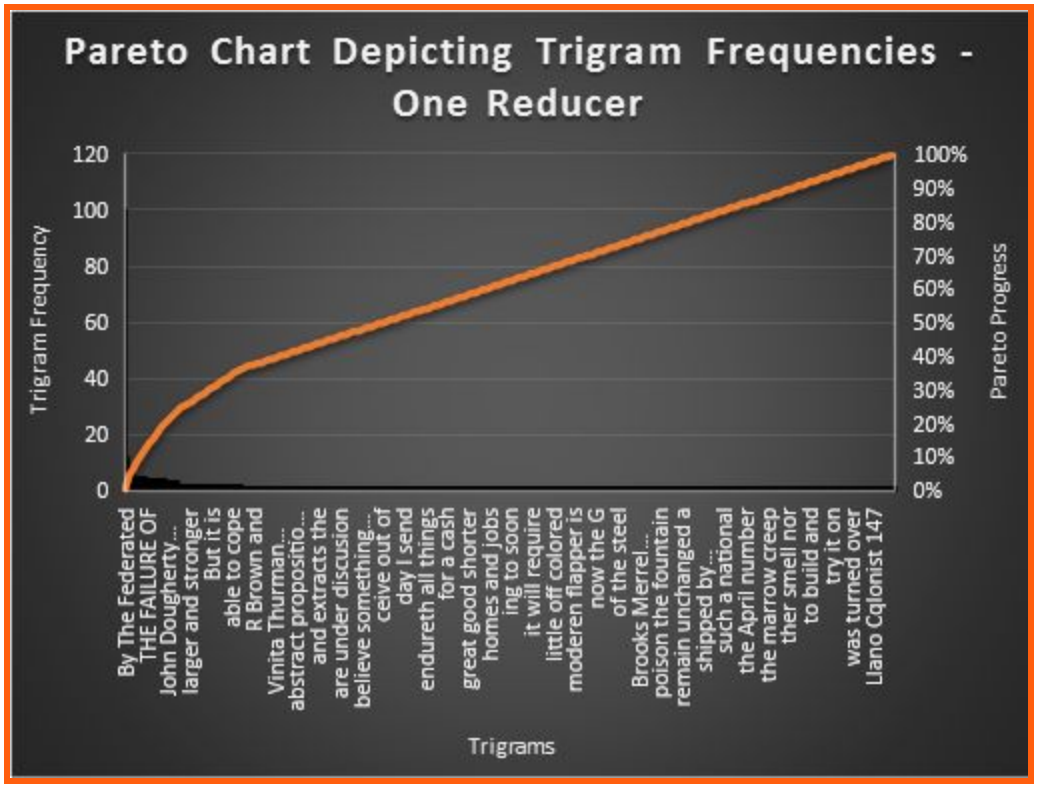
Our Reduce Algorithm counts the number of occurrences of each unique trigram in the input file.

//TO DO - explain the results, which words occur frequently together

The subsequent sections present a graphical representation of our empirical analysis of the two cases mentioned in the problem statement.

1. **One Reducer:**

Here is a Pareto Chart representing the frequencies of the Trigrams calculated:



Here is a list of top 25 Trigrams identified:

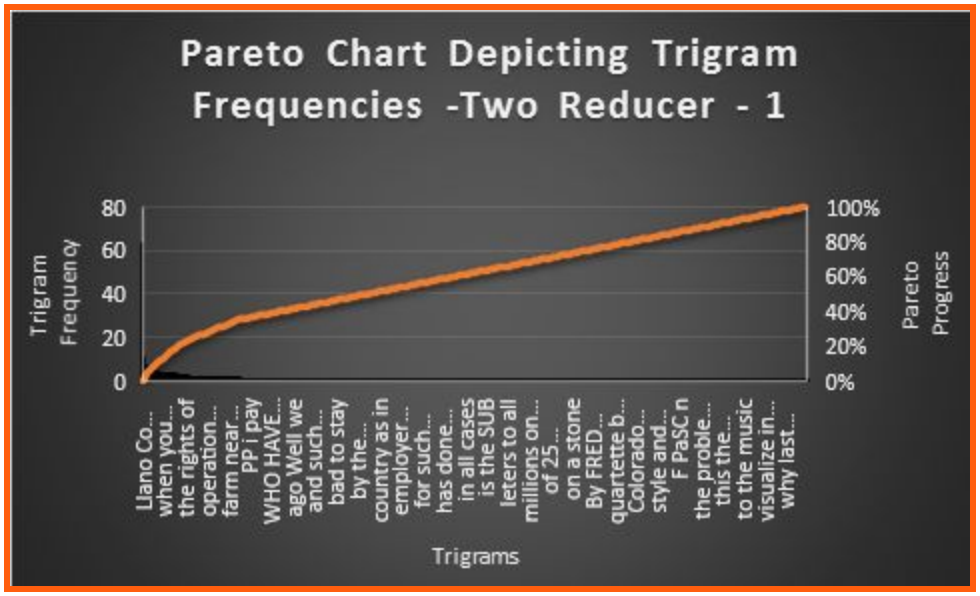
By The Federated	100
The Federated Press	81
Llano Co operative	63
of the Colony	58
Co operative Colony	53
one of the	34
as well as	32
the Colony is	28

a co operative	27
of the world	27
the United States	27
worked at the	25
THE LLANO PUBLICATIONS	24
the Llano Co	24
DOLLAR UP CLUB	23
CO OPERATIVE COLONY	22
Co operative Farm	22
LLANO CO OPERATIVE	22
be able to	21
the co operative	21
American Co operator	20
and it is	20
in the world	20
to the Colony	20
The American Co	19

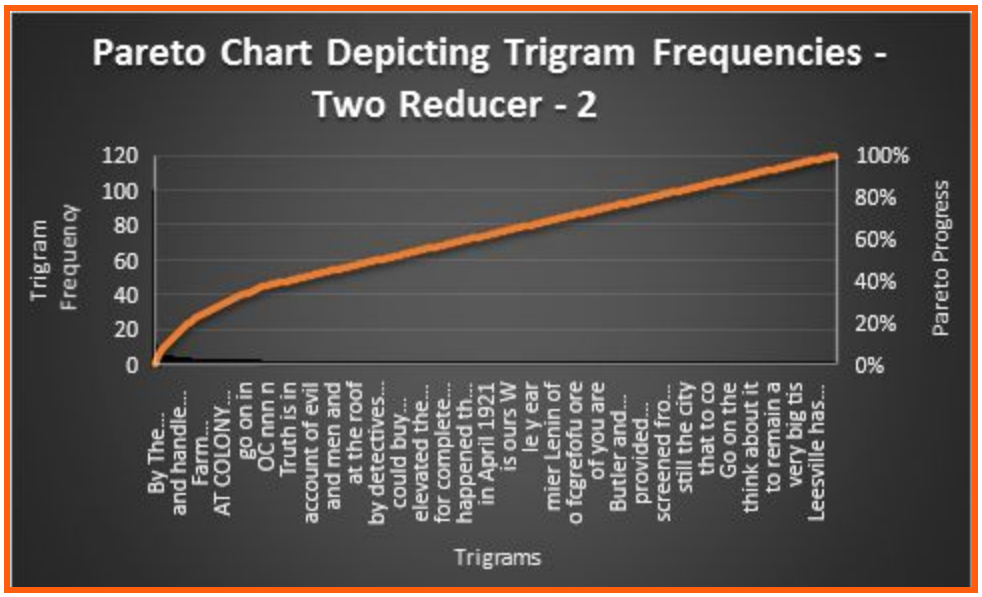
2. **Two Reducers:**

   We got two output files in this case.

   Here is a Pareto Chart representing the frequencies of the Trigrams calculated in the first output file:



Here is a Pareto Chart representing the frequencies of the Trigrams calculated in the first output file:



PLEASE NOTE:

Number of Trigrams identified by first reducer = 36445
Number of Trigrams identified by second reducer = 36298
Total = 72743 = One Reducer Case
No discrepancies in output observed.

Number Of Unique Trigrams Identified

Our MapReduce program to count the number of Trigrams identified *72743 unique trigrams over a corpus of data that included 41 articles of english language, out of which 39 were distinct and 2 were replicated* as per the problem statement.

Affect In The Program Run Time By Changing Number Of Mappers and Reducers

As explained in the design issues, we could not change the number of mappers.

However, we could modify the number of reducers using the command line.

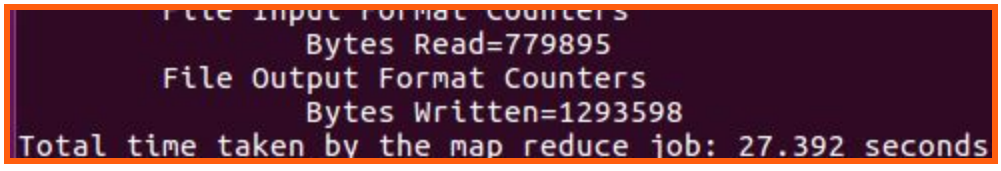
Our program calculates the program runtime by adding the below code to the main():

```
double startTime = (double) System.currentTimeMillis();
....
.....
if(job.waitForCompletion(true)){
                double endTime = System.currentTimeMillis();
                System.out.println("Total time taken by the map reduce job: "+(endTime-startTime)/1000+" seconds");
                System.exit(0);
        }
        else{
                System.exit(1);
        }
```

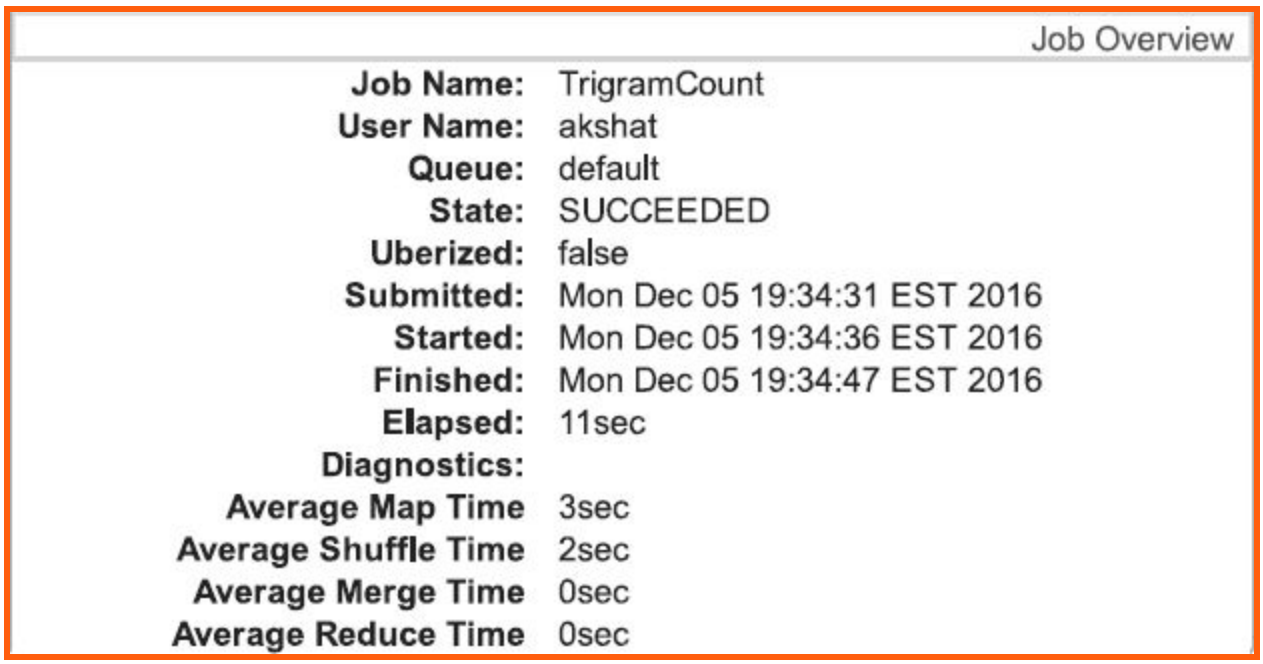
Here are the run times calculated in the two cases:

1. **One Reducer:**

   Time Calculated by our program: *27.392 seconds*



Time Calculated by MapReduce JobHistory: *11 seconds*

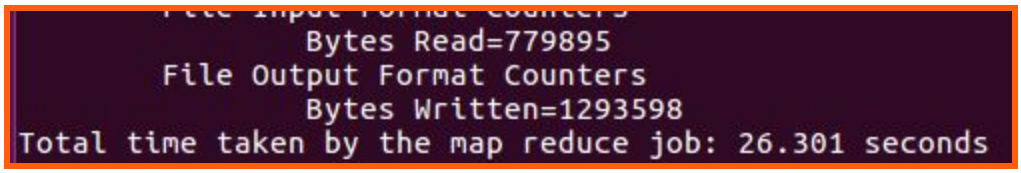


Job Overview

Job Name:	TrigramCount
User Name:	akshat
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Mon Dec 05 19:34:31 EST 2016
Started:	Mon Dec 05 19:34:36 EST 2016
Finished:	Mon Dec 05 19:34:47 EST 2016
Elapsed:	11sec
Diagnostics:	
Average Map Time	3sec
Average Shuffle Time	2sec
Average Merge Time	0sec
Average Reduce Time	0sec

2. **Two Reducers:**

Time Calculated by our program: *26.301 seconds*

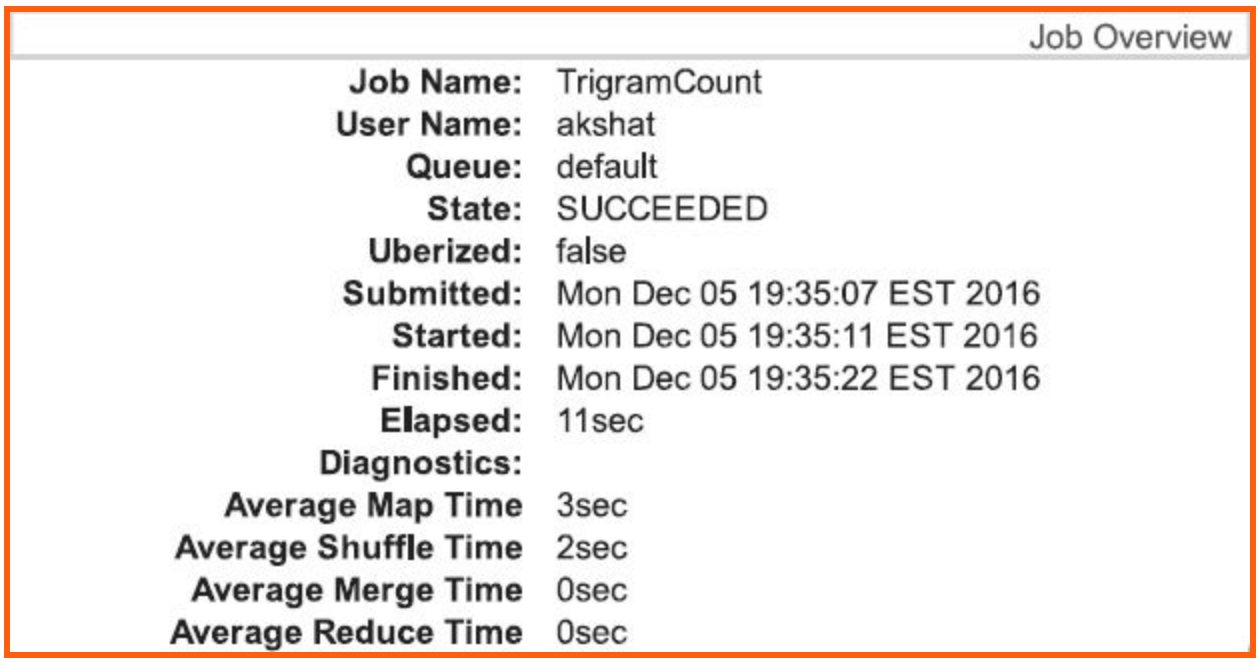


```
                Bytes Read=779895
        File Output Format Counters
                Bytes Written=1293598
Total time taken by the map reduce job: 26.301 seconds
```

Conclusion:

As we can see above, there is no considerable change in the run time of the programs in our case by changing the number of reducers.

Time Calculated by MapReduce JobHistory: *11 seconds*



Job Overview	
Job Name:	TrigramCount
User Name:	akshat
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Mon Dec 05 19:35:07 EST 2016
Started:	Mon Dec 05 19:35:11 EST 2016
Finished:	Mon Dec 05 19:35:22 EST 2016
Elapsed:	11sec
Diagnostics:	
Average Map Time	3sec
Average Shuffle Time	2sec
Average Merge Time	0sec
Average Reduce Time	0sec

Wait Times Before Job Processing

We did not observe any substantial wait times before our job was processed except one time when the wait time increased to about 3 hours due to duplication of datanode on the ccr cluster.

Practical Experiences

While running our MapReduce job on the CCR via the Ubuntu 16.04 platform, we observed a peculiar issue where in we saw that container launch was failing as per the below logs:

Diagnostics: Exception from container-launch.

Container id: container_1480947435876_0067_02_000001

Exit code: 1

Stack trace: ExitCodeException exitCode=1:

at org.apache.hadoop.util.Shell.runCommand(Shell.java:582)

at org.apache.hadoop.util.Shell.run(Shell.java:479)

at org.apache.hadoop.util.Shell$ShellCommandExecutor.execute(Shell.java:773)

at org.apache.hadoop.yarn.server.nodemanager.DefaultContainerExecutor.launchContainer(DefaultContainerExecutor.java:212)

Upon digging up further on the issue, we found that the JDK version used to build our jar was different than the one used to deploy Hadoop on CCR.

We installed a different version of JDK that is compatible with the JDK version of Hadoop deployment on CCR.

Also, given the corpus of data that we selected, we noticed that data contained a lot of special character data that we found would be redundant for our study. To get rid of these special characters, we used split() of Java String class as follows in our mapper code:

```
String[] wordArray = textFileString.split("\\W+");
```

Fortunately, we were able to solve the above challenges faced within due time using the perfect solutions and no workarounds.

//TODO - What would you have liked to do if you had infinite resources available?