Hadoop Exercise to Create an Inverted Index

Objectives:

- **Creating an Inverted Index of words occurring in a set of web pages**
- **Get hands-on experience in GCP App Engine**

We'll be using a subset of 74 files from a total of 408 files (text extracted from HTML tags) derived from the Stanford WebBase project that is available here. It was obtained from a web crawl done in February 2007. It is one of the largest collections totaling more than 100 million web pages from more than 50,000 websites. This version has been cleaned for the purpose of this assignment.

These files will be placed in a bucket on your Google cloud storage and the Hadoop job will be instructed to read the input from this bucket.

1. Uploading the input data into the bucket
   a. Get the data files from either of the links below
   http://csci572.com/2022Spring/hw3/DATA.zip

   https://drive.google.com/drive/u/1/folders/1Z4KyaIluddPGVkIm6dUjkpD_FiXyNIcq

   You should use your USC account to get access to the data from the Google Drive link. Compressed full data is around 1.1GB. Uncompressed, it is 3.12 GB of data for the files for this project. So on balance you should download the zipped file, not the folder.

   b. Unzip the contents. You will find two folders inside named '**development**' and '**fulldata**'. Each of the folders contains the actual data (txt files). We suggest you use the development data initially while you are testing your code. Using the fulldata will take up to few minutes for each run of the Map-Reduce job and you may risk spending all your cloud credits while testing the code.
   c. Click on 'Dataproc 'in the left navigation menu under . Next, locate the address of the default **Google cloud storage staging** bucket for your cluster in the Figure 1 below. If you've previously disabled billing, you need to re-enable it before you can upload the data. Refer to the "**Enable and Disable Billing account"** section to see how to do this.
   d.

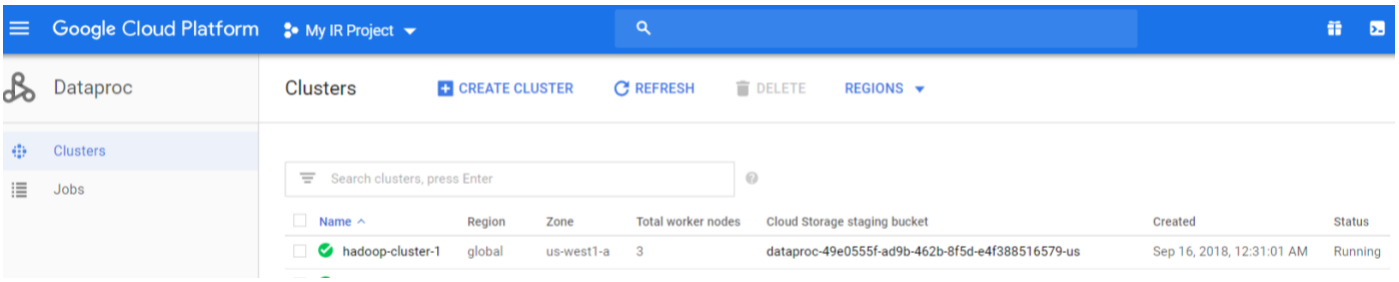

Figure 1: The default Cloud Storage bucket.

e. Go to the storage section in the left navigation bar and select your cluster's default bucket from the list of buckets. At the top you should see menu items UPLOAD FILES, UPLOAD FOLDER, CREATE FOLDER, etc (Figure 2). Click on the UPLOAD FOLDER button and upload the `dev_data` folder and `full_data` folder individually. This will take a while, but there will be a progress bar (Figure 3). You may not see this progress bar as soon as you start the upload but, it will show up eventually.

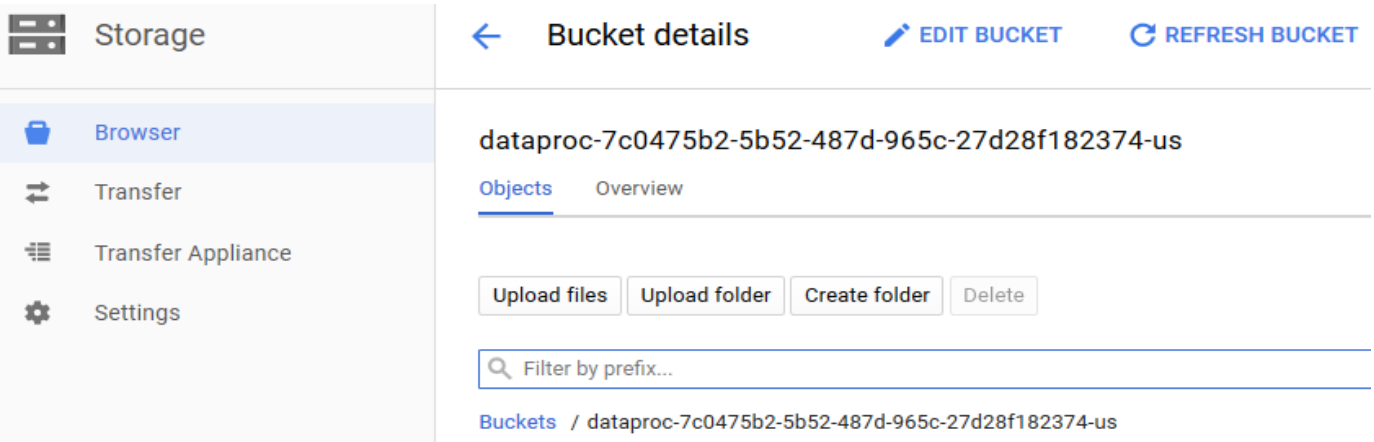

Figure 2: Cloud Storage Bucket.

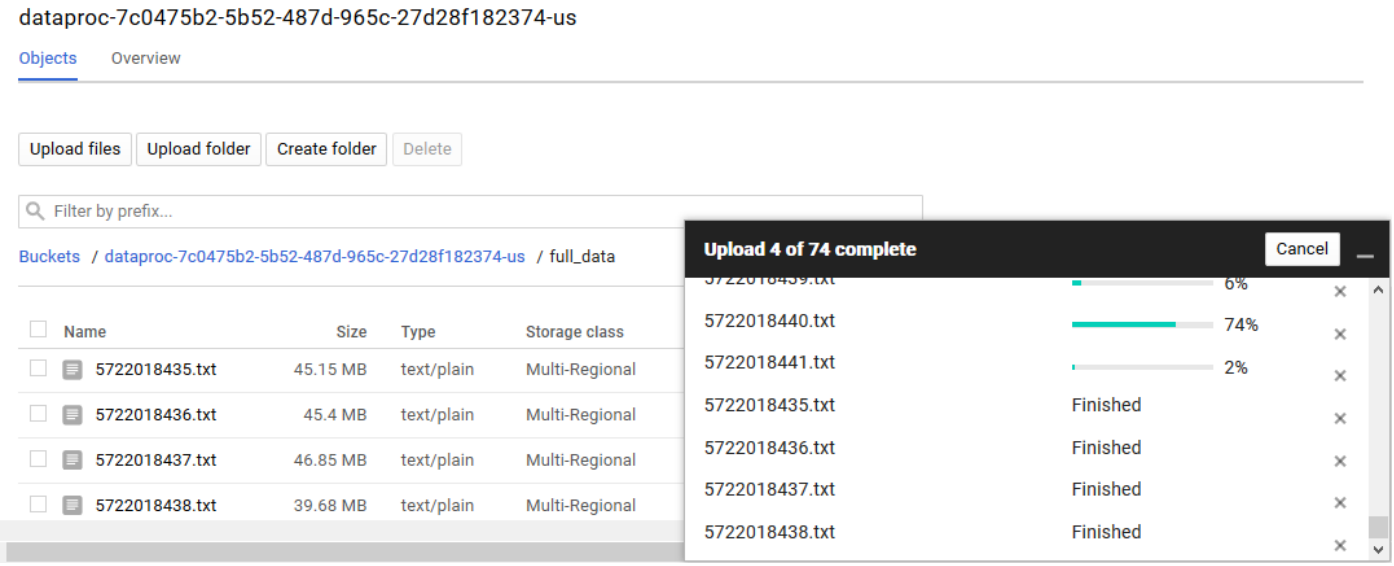

Figure 3: Progress of uploading

Inverted Index Implementation using Map-Reduce

Now that you have the cluster and the files in place, you need to write the actual code for the job. As of now, Google Cloud allows us to submit jobs via the UI, only if they are packaged as a jar file. The following steps are focused on submitting a job written in Java via the Cloud console UI.

Refer to the examples below and write a Map-Reduce job in Java that creates an Inverted Index given a collection of text files. You can very easily tweak a **word-count example** to create an inverted index instead (**Hint**: Change the mapper to output `word` **`docID`** instead of `word count` and in the reducer use a **HashMap**).

For examples of Map-Reduce Jobs go to Google and enter the query
"map reduce tutorial examples"
The YouTube videos are especially good, and most of them are short and easy to digest.
And alternate source of examples can be found at
https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

The example in the following pages explains a Hadoop word count implementation in detail. It takes one text file as input and returns the word count for every word in the file. Refer to the comments in the code for explanation.

The Mapper Class:

```
/*
This is the Mapper class. It extends the Hadoop's Mapper class.
This maps input key/value pairs to a set of intermediate(output) key/value pairs.
Here our input key is a LongWritable and input value is a Text.
And the output key is a Text and value is an IntWritable.
*/
class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable>
{
    /*
    Hadoop supported data types. This is a Hadoop specific datatype that is used to handle
    numbers and Strings in a hadoop environment. IntWritable and Text are used instead of
    Java's Integer and String datatypes.
    Here 'one' is the number of occurances of the 'word' and is set to the value 1 during the
    Map process.
    */
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException
    {
        //Reading input one line at a time and tokenizing.
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);

        //Iterating through all the words available in that line and forming the key value pair.
        while (tokenizer.hasMoreTokens())
        {
            word.set(tokenizer.nextToken());
            /*
            Sending to output collector(Context) which in-turn passes the output to Reducer.
            The output is as follows:
                'word1' 1
                'word1' 1
                'word2' 1
            */
            context.write(word, one);
        }
    }
}
```

The Reducer Class:

```java
/*
This is the Reducer class. It extends the Hadoop's Reducer class.
This maps the intermediate key/value pairs we get from the mapper to a set
of output key/value pairs, where the key is the word and the value is the word's count.
Here our input key is a Text and input value is a IntWritable.
And the output key is a Text and value is an IntWritable.
*/
class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable>
{
    /*
    Reduce method collects the output of the Mapper and adds the 1's to get the word's count.
    */
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException
    {
        int sum = 0;
        /*
        Iterates through all the values available with a key and add them together and give the
        final result as the key and sum of its values
        */
        for (IntWritable value : values)
        {
            sum += value.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

Main Class

```java
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.*;
public class WordCount
{
    public static void main(String[] args)
        throws IOException, ClassNotFoundException, InterruptedException {
        if (args.length != 2) {
            System.err.println("Usage: Word Count <input path> <output path>");
            System.exit(-1);
        }
        //Creating a Hadoop job and assigning a job name for identification.
        Job job = new Job();
        job.setJarByClass(WordCount.class);
        job.setJobName("Word Count");
        //The HDFS input and output directories to be fetched from the Dataproc job submission console.
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        //Providing the mapper and reducer class names.
        job.setMapperClass(WordCountMapper.class);
        job.setReducerClass(WordCountReducer.class);
        //Setting the job object with the data types of output key(Text) and value(IntWritable).
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.waitForCompletion(true);
    }
}
```

The input data is cleaned, that is all the \n\r s is removed but one or more \t might still be present (which needs to be handled). There will be punctuation and you are required to handle this in your code. Replace all the occurrences of special

characters and numerals by space character, convert all the words to the lowercase. **Single '\t' separates the key (Document ID) from the value (Document)**. The input files are in a key value format as below:

```
DocumentID      document
```

Sample document:

```
5722018411  A look at the most publicized aspects of the strike-- economics, stress, and management-- shows how these issues obscured and distorted the
controllers' main concern of workplace control and helps explain why problems persist in the ATC workforce.  It also demonstrates how management and labor's
focus on economic issues since World War II has bankrupted labor's discourse and limited its ability to address concerns outside of a narrow range of
concerns.These perceptions were in part responsible for the overwhelming public approval of Reagan's handling of the strike; 65% in a public opinion poll;
mail, according to one representative, ran 1000 to 1 in favor of the administration.Most strikers denied that money was a critical component in their decision
to strike.  Yet Poli insisted that his demands, headed by a pay raise, reflected the desires of his constituency.  Arthur Shostak, who conducted five surveys
of PATCO members in 1979 and 1980 backs up Poli's assertion that salary was important to the strikers.  It is tempting to concede then that workers did see the
```

The mapper's output is expected to be as follows:

```
aspect 5722018411
distorted 5722018411
economics 5722018411
economics 5722018411
management 5722018411
publicized 5722018411
```

The above example indicates that the word `aspect` occurred 1 time in the document with docID `5722018411` and `economics` 2 times.

The reducer takes this as input, aggregates the word counts using a HashMap and creates the Inverted index. The format of the index should be as follows: **Single '\t' separates the word from the docID and its count. For each docID:count, a single space separates them.**

```
word['\t']docID:count[space]docID:count[space]docID:count...
```

```
answer  5722018453:2 5722018483:1
antecedence 5722018502:1 5722018435:1
asterisks.  5722018417:1 5722018504:2 5722018447:1
beautiful   5722018439:7 5722018417:2 5722018416:3 5722018438:5 5722018437:1 5722018415:1 5722018414:2 5722018435:3
bind    5722018419:6 5722018417:39 5722018416:1
chunking    5722018507:1 5722018502:1
```

The above is just a sample and shows a portion of the inverted index created by the reducer.

Ways to write code

1. you can use the **VI** or **nano** editors that come pre-installed on the master node. You can test your code on the cluster itself. Be sure to use the development data while testing the code. You are expected to write a simple Hadoop job. You can just tweak this example if you'd like, but make sure you understand it first. For this,

a. To create nano editor:

   sudo nano <filename>

b. Write code and save the file.

2. If you want to test the code first, you can create a Java (Maven) project in eclipse and write down the code and manage the dependencies just like in assignment 2. Once that is done, you will have to upload the single **.java** file from that project on GCP. To do this,
   a. Go to dataproc cluster.
   b. Click on the cluster name.
   c. Go to the VM Instance tab.
   d. On the right side of your master node, you can see the SSH. Click on it. If it is disabled, then please make sure that your billing is on.
   e. Inside the terminal, on the top right corner, you will see an icon (settings icon), click on it. There you will find an option of upload a file. With that, you should submit your **.java file**. (Please don't create **.jar** from eclipse as it causes class Def not found exception).

   Note: Before uploading **.java** file, please remove the first line defining package name, which is created by default by eclipse. Otherwise, it causes the class not found exception.

3. You can create .java code locally and follow all the steps in option 2 to upload it.

Creating a jar for your code

Now that your code for the job is ready, we'll need to run it. The Google Cloud console requires us to upload a Map-Reduce job as a jar file. In the following example the Mapper and Reducer are in the same file called `InvertedIndexJob.java`.To create a jar for the Java class implemented please follow the instructions below. The following instructions were executed on the cluster's master node on the Google Cloud.

1. Say your Java Job file is called `InvertedIndexJob.java`. Create a JAR as follows:
   - `hadoop com.sun.tools.javac.Main InvertedIndexJob.java`
     If you get the following Notes you can ignore them
     ```
     Note: InvertedIndexJob.java uses or overrides a deprecated API.
     Note: Recompile with -Xlint:deprecation for details.
     ```
   - `jar cf invertedindex.jar InvertedIndex*.class`

Now you have a jar file for your job. You need to place this jar file in the default cloud bucket of your cluster. Just create a folder called JAR on your bucket and upload it to that folder. If you created your jar file on the cluster's master node itself use the following commands to copy it to the `JAR` folder.

- `hadoop fs -copyFromLocal ./invertedindex.jar`
- `hadoop fs -cp ./invertedindex.jar gs://dataproc-69070.../JAR`

The highlighted part is the default bucket of your cluster. It needs to be prepended by the `gs://` to tell the Hadoop environment that it is a bucket and not a regular location on the filesystem.
Note: This is not the only way to package your code into a jar file. You can follow any method that will create a single jar file that can be uploaded to the Google cloud.

Submitting the Hadoop job to your cluster

As mentioned before, a job can be submitted in two ways.

1. From the console's UI.
2. From the command line on the master node.

If you'd like to submit the job via the command line follow the instructions here although doing it from the UI is simpler.

https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Follow the instructions below to submit a job to the cluster via the console's UI.

1. Go to the "Jobs" section in the left navigation bar of the Dataproc page and click on "**Submit job**".

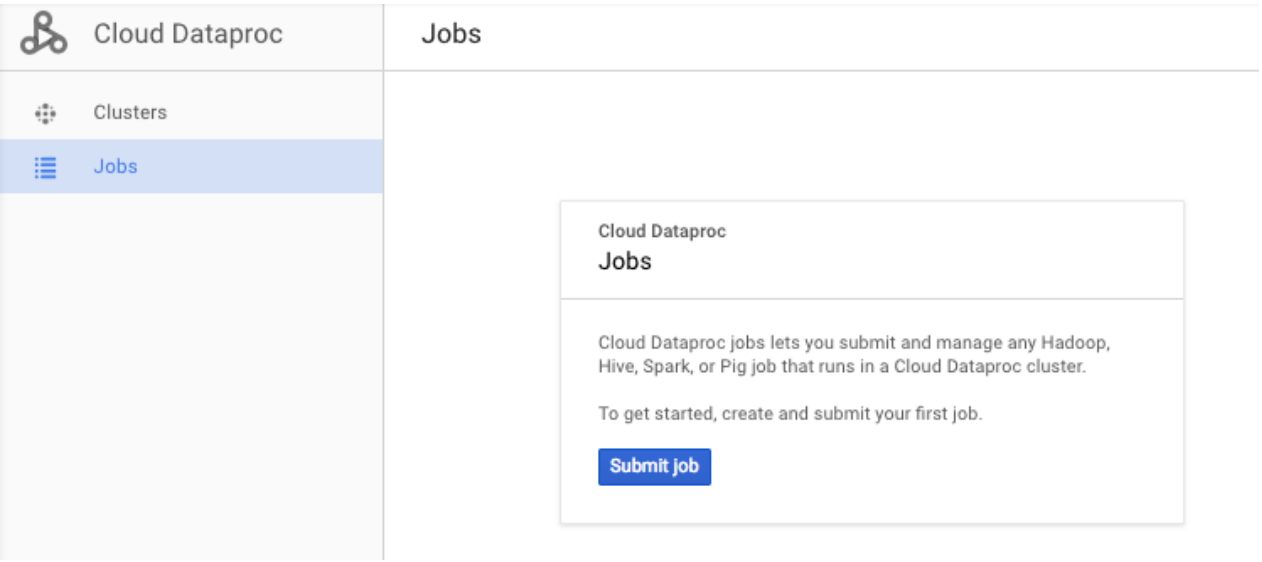

Figure 4: Dataproc jobs section

2. Fill the job parameters as follows (see Figure 13 for reference):
   - **Cluster**: Select the cluster you created
   - **Region:** Choose the same region you used to create your cluster – us-west1
   - **Job Type**: Hadoop
   - **Jar File**: Full path to the jar file you uploaded earlier to the Google storage bucket. Don't forget the `gs://`
   - **Main Class or jar**: The name of the Java class you wrote the mapper and reducer in.
   - **Arguments**: This takes two arguments
     i. **Input**: Path to the input data you uploaded. Don't forget the `gs://`
     For part 1 (unigrams) the input folder will be fulldata and for part 2 (bigrams) it will be devdata.
     ii. **Output**: Path to the storage bucket followed by a **new** folder name. The folder you specify here is created during execution. You will get an error if you give the name of an existing folder. Don't forget the `gs://`.
   - Leave the rest at their default settings

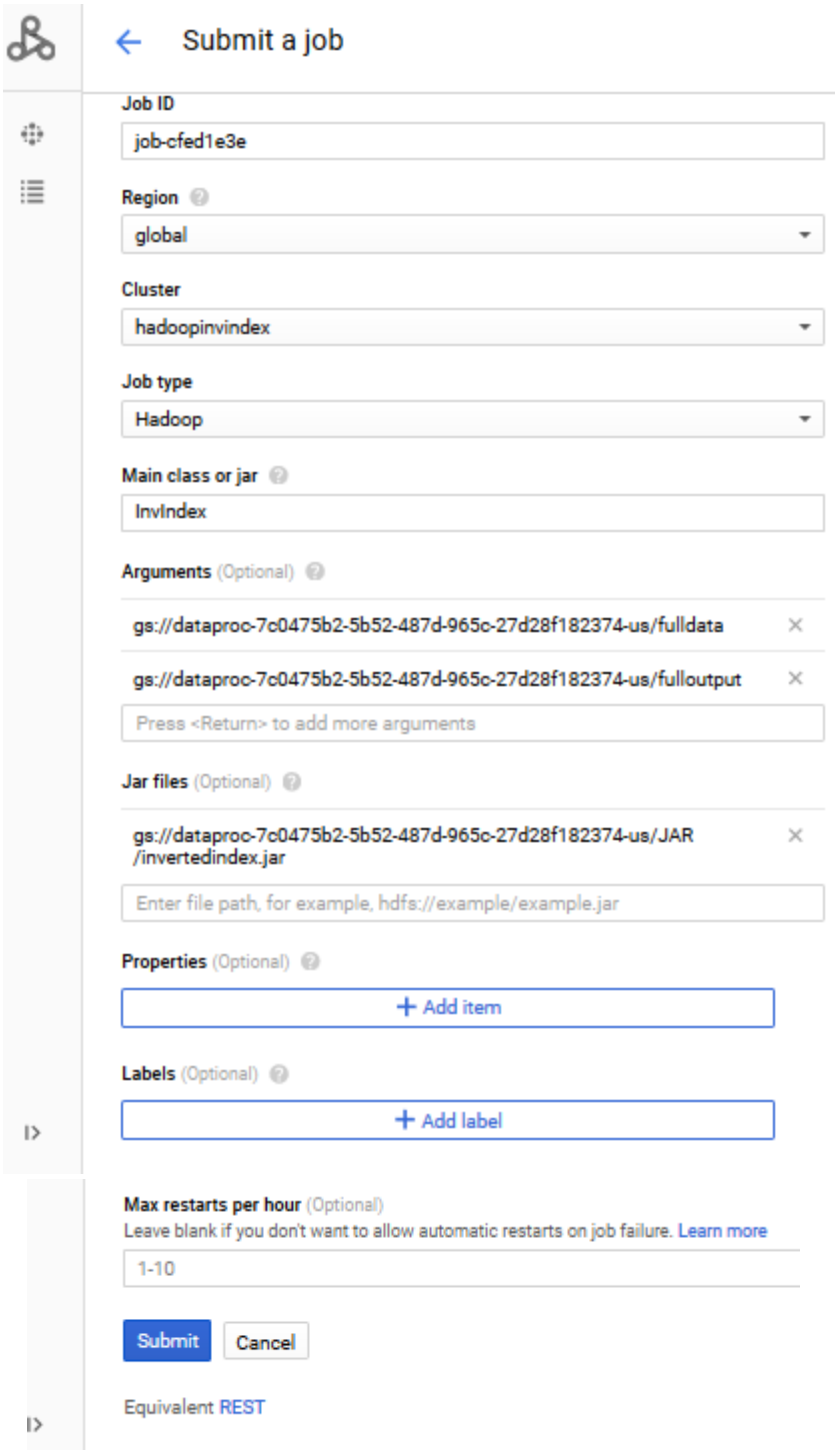

Figure 5: Job submission details

3. Submit Job. It will take quite a while. Please be patient. You can see the progress on the job's status section.

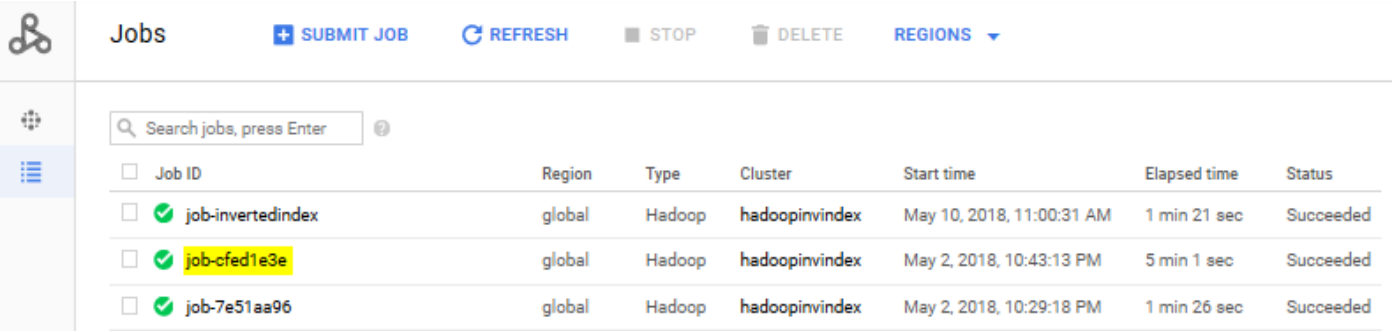

Figure 6: Job ID generated. Click it to view the status of the job.

<u>NOTE</u>: If you encounter a **Java.lang.Interrupted exception** you can safely ignore it. Your submission will still execute.

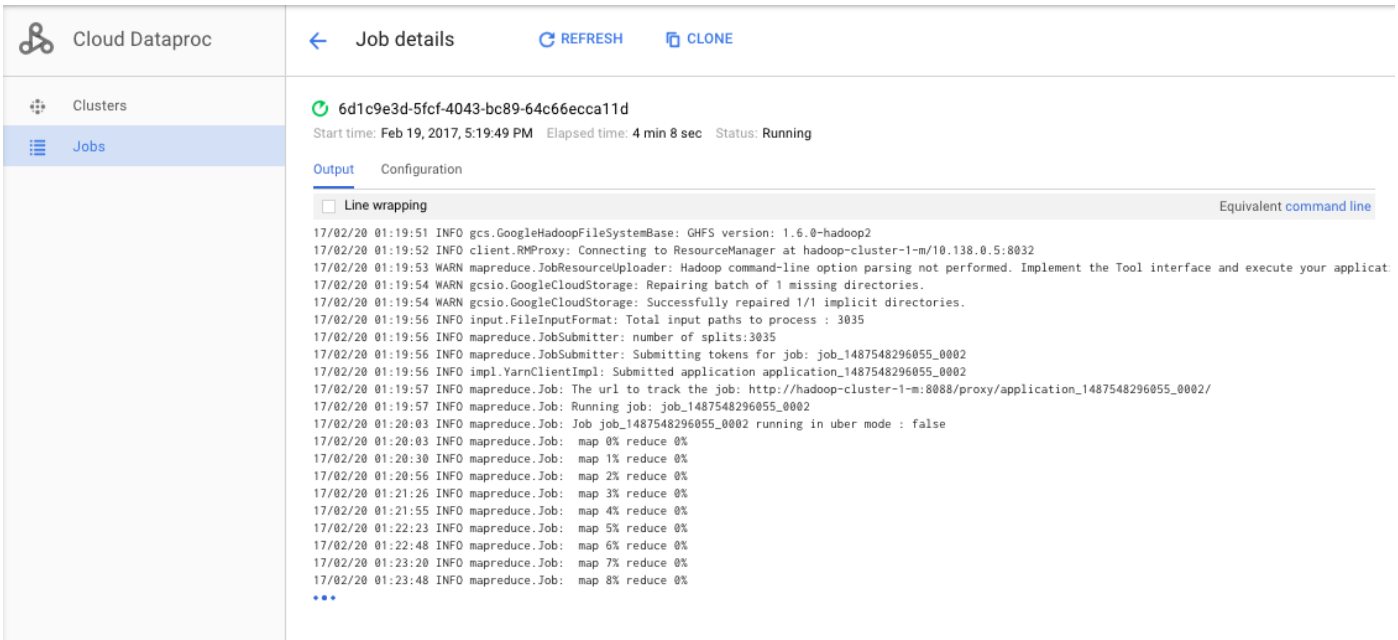

Figure 7: Job progress

4. Once the job executes copy all the log entries that were generated to a text file called `log.txt`. You need to submit this log along with the Java code. You need to do this only for the job you run on the full data. No need to submit the logs for the dev_data.

5. The output files will be stored in the `output` folder on the bucket. If you open this folder, you'll notice that the inverted index is in several segments. (Delete the **_SUCCESS** file in the folder before merging all the output files)

To merge the output files, run the following command in the master nodes command line (SSH)

- `hadoop fs -getmerge gs://dataproc-69070458-bbe2-.../output ./output.txt`
- `hadoop fs -copyFromLocal ./output.txt`
- `hadoop fs -cp ./output.txt gs://dataproc-69070458-bbe2-.../output.txt`

The output.txt file in the bucket contains the full Inverted Index for all the files.

1.Sort your output.txt file using the command

```
sort -o output_sorted.txt output.txt
```

2. Use `grep` to search for the words mentioned in the submissions section. Using grep is the fastest way to get the entries associated with the words. For example, to search for "`string`" use

```
grep -w '^string      ' output_sorted.txt
or
grep -w '^string' output_sorted.txt
```

Note:-> In the above grep command, the word to be searched should be followed by a tab character.

Part II: Inverted Index of Bigrams using Map-Reduce

Now that you are familiar with setting up and running Hadoop jobs on GCP, you will now modify your `InvertedIndexJob.java` script to generate an inverted index of bigrams (instead of unigrams).

Your existing Mapper class emits (word, docID) pairs which are then aggregated in the Reducer class. You will have to modify your Mapper class to emit ("word1 word2", docID) pairs instead. The reducer remains unchanged.

Once you modify your class(es), create the jar `invertedindex_bigrams.jar` and dispatch a Hadoop job on the `devdata/` in the same manner as before.

The output should look something like this: **Single `'\t'` separates the word from the docID and its count. For each docID:count, a single space separates them.**

```
word['\t']docID:count[space]docID:count[space]docID:count...
```

```
a buff  5722018235:1 5722018508:1
a buffer    5722018235:12 5722018301:34 5722018101:2 5722018496:1 5722018508:21
a bugs  5722018301:1 5722018235:2 5722018508:1
a bulging   5722018235:2 5722018101:1
a bulldozer 5722018235:2 5722018301:2 5722018101:1 5722018496:1 5722018508:2
a bullet    5722018301:9 5722018235:7 5722018101:1 5722018496:4 5722018508:7
```

To get credit for this task, create another text file `index_bigrams.txt` with the index entries for the following bigram phrases (generated from `devdata/`):

1. computer science
2. information retrieval
3. power politics
4. los angeles
5. bruce willis

You can apply grep on the output file in the same way you did for the previous exercises.

Submission Instructions:

Part I

1. Include all the code that you have written (Java) and the log file created for the full data job submission.
2. Also include the inverted index file for the document "5722018484.txt"
3. Create a text file named `index.txt` and include the index entries for the following words
    a. architecture
    b. technology
    c. temperature
    d. academics
    e. concurrent
    f. experiment
    g. catalogue
    h. hierarchy

    Add the full line from the index including the word itself

4. Also submit a screenshot of the output folder for the fulldata run in GCP**.**
5. Also submit log file generated from running the job on the fulldata.
6. **Do NOT submit your full index.**
7. Compress your code and the text file into a single zip archive and name it `index.zip`. Use a standard zip format and not zipx, rar, ace, etc.

Part II

1. Create a folder named bigram.
2. Submit this file `index_bigrams.txt` along with your modified Java code (rename it as `InvertedIndexBigrams.java`) as part of the `index.zip` archive.
3. Screenshot of the output folder for the devdata run in GCP.
4. Submit log file generated from running the job on the devdata.

To summarize

For Part 1 submit: (To be done on fulldata)

1. InvertedIndex.java
2. Logs of unigram job on full data (named as "log.txt")
3. index.txt for given words in full data folder (named as "index.txt")
4. Index.txt for 5722018484.txt (named as "index_5722018484.txt")
5. Screenshot of output folder (named as "fulloutput.png")

For Part 2 submit: (To be done on devdata)

6. InvertedIndexBigrams.java
7. index_bigrams.txt for given words in devdata folder (named as "index_bigrams.txt")
8. Logs for bigram job on devdata (named as "log_bigram.txt")
9. Screenshot of output folder (named as "devoutput.png")

Compress these 9 files into index.zip (Create a separate folder for the files in part 2. Name the folder as "bigram"). This should be the architecture of the index.zip:

index.zip:

- **InvertedIndexJob.java**
- **log.txt**
- **index.txt**
- **index_5722018484.txt**
- **fulloutput.png**
- **bigram (folder)**
  - **InvertedIndexBigrams.java**
  - **log_bigram.txt**
  - **index_bigrams.txt**
  - **devoutput.png**

FAQ:

Q) Can't seem to select a cluster for submitting a job?
A) Changing the region will do the trick

Q) How many files were there in full_data while uploading?
A) You need to upload .txt files only.
Number of .txt files is 74.

Q) Chrome suffers, in uploading 74 files.
A) Consider opening the storage in another tab and checking the number of files. This way you will be able to know when the upload is complete.

Q) Do we have to use Java as the programming language?
A) Please go ahead and use any language binding of your choice.
Note: You may be on your own with language other than Java. TA's may not be able to help with other languages.

Q) How to Import and Export Java Projects as JAR Files in Eclipse?
A) **http://www.albany.edu/faculty/jmower/geog/gog692/ImportExportJARFiles.htm**

Q) Is it fine to submit only one .java file, which has the all the (Mapper and Reducer Classes) inside it ?
A) One .java file containing your entire program should be good enough.

Q) Approximately how long does it take for a submitted job to finish in GCloud Dataproc?
A) It takes approximately 10 minutes

Q) Should the postings list be in the sorted order of docIDs?
A) No need to sort the listings.

Q) Google cloud is not allowing to ssh?
A) You need to start VMs manually.

Q) Where can I find log files?
A) Cloud Dataproc -> Under Jobs

Click on one of the jobs you ran. The details of execution you see can be copied to a text file since there is NO single log.t*xt* that gets generated. The logs are visible there and it just needs to be copied.

Q) Failure in creating the JAR file (Hadoop not found error)
A) Check if environment variables for JAVA and HADOOP_CLASSPATH are set up. Please note that this step has to be done each time you open a new SSH terminal.

Q) How to check number of files in full_data on storage bucket?
A) Go to your bucket, select the full_data folder and click on delete. It'll list out the total files present. DO NOT PRESS DELETE in the dialog box that appears. Or run the following command from the hadoop cluster terminal:
hadoop fs -find gs://...//full_data/*.txt | wc -l
You can perform certain sanity checks.
1) Check if your code run properly for the dev_data?
2) Check if you used correct space / tab specifications as mentioned in the assignment description, sometimes it might be the problem with the storage space related to that.
3) You can debug with a single custom file to see, if everything is properly indexed or not.

Q) Different index order. Should we take the same index order (sorted) or can it be different (unsorted)?
A) Order does not matter. The accuracy of results is important.

Q) Code runs fine on development but strange file size with full data.
A) Check if the results produced by running on dev_data produces huge file sizes as well. If so, that means you have to check your code. If not, check if your full_data is uploaded correctly.

Q) I'm getting this error repeatedly, but I've already created the output directory and have set the argument path to that directory. Can someone help me with it?
A) You need to delete the output folder because the driver will attempt to create the output folder based on the argument provided.

Q) Am able to run the dev_data and it is generating results. But if I ran the same code on the full data I am getting an error. The job is running for till map 25% and then it throws an error?
A) Please check that you have all the files uploaded just fine, and you should have 74 files in full_data.

Q) Starting VM instance failed

When I try to start the VM instances, for some of them it shows the message:

Error: Quota "CPUS" exceeded: Limit 8.0?

A) If you get an error saying that you've exceeded your quota, reduce the number of worker nodes or choose a Machine Type (for master and worker) with fewer vCPUs.

Q) Did anyone run into a situation where if you go under Dataproc > Clusters > (name of cluster instance) > VM instances > SSH, the only available option is to use another SSH client?
A) You probably didn't start the VM instances. Every time you disable billing and enable billing, you need to start VMs manually.

Q) Error enabling DataProc API
A) shut down project and create new one

Q) No space between DocID:count pairs in the output file after merge?
A) Happens due to copy-pasting the grep output from console to a text file. Pipe the grep output into a file and then download that file from gcloud

Q) "message" : "982699654446-compute@developer.gserviceaccount.com does not have storage.objects.get access to dataproc-60450493-bff5-4160-8156-fcb96702ebf0-us/full_data_new/32229287.txt.",
"reason" : "forbidden"
A) If you're using a custom service account, you still have to give reader access to the Default service account <your-project-number>-compute@developer.gserviceaccount.com

Q) Shall we add everything in one folder or different folders?
A) Anything is fine, just make sure you give proper file names so that graders can understand it.

Q) Can we use a different name for file index.txt generated from 5722018484.txt in part 1 ?
A)You can use index_5722018484.txt.

Q) Can bigram cross two sentences? For example "Today is a sunny day. Tomorrow will be windy.". After we remove all special characters and numbers, we get: "today is a sunny day tomorrow will be windy". In this case is "day tomorrow" a valid bigram?
A) Yes, Once you remove all the special characters and convert everything to lower case, you have to consider the whole thing as one sentence and create the bigrams.

Important Points:

#P1) Output folder Number of parts generated - can be any number
#P2) Manually inspect output.txt and copy lines for the words from it and create a new txt file named index.txt. - for the 8 words
#P3) start worker nodes before submitting job to cluster
#P4) No Sysout - write in logs from reduce function
#P5) jar tvf jar_file_name - to list class files archived for a jar
#P6) space in your folder name which is treated as illegal character - throws error
#P7) Every time you disable billing and enable billing, you need to start VMs manually.
#P8) If you are not able to upload full data then add the data in parts. The main aim is to get all the files there.
#P9) Size of output folder for full data can vary.
#P10) When using "grep" command to search for keywords, look for exact match.
#P11) You just disable the billing account (when you're not using it) as per mentioned in the HW description. There's no need to disable the cluster, they'll not charge you. Also, when you'll disable the billing account; your master and worker threads will be disabled automatically.
#P12) You have to write a program that outputs the entire inverted index file and then using the grep command filter out the indices of the eight words given