Nicolas Escobar (escobarn)  
Harsh Reddy (hagandav)

Cloud Computing - Project 3 Report

**Deliverables**

You are required to turn in the following items in a zip file (username HadoopBlast.zip)

1. The source code of Hadoop Blast you implemented.

**The source code is located in the folder Hadoop\_BLAST\_Source**

1. Technical report (username HadoopBlast report.docx) that answers the following questions.
   1. What is Hadoop Distributed Cache and how is it used in this program?

**Hadoop Distributed Cache is a mechanism to distribute files and/or archives to the nodes so the tasks can use this data during the MapReduce job before they run.**

**In this program, we used the Distributed Cache to distribute the BLAST archive (which contains the DB and the binary BLAST program) to the different nodes. This is done in the driver program (DataAnalysis). Later in the Map setup (RunnerMap), the path to the archive is extracted from the Distributed Cache just before the Mapper runs. The local paths to the binary and the DB are later used in the Mapper to execute the BLAST command.**

* 1. Write the two lines that put and get values from Distributed cache. Also include the method and class information.  
       
     Putting values into the Distributed Cache  
     **Class:** public class DataAnalysis extends Configured implements Tool **Method:** void launch(int numReduceTasks, String programDir, String execName, String workingDir, String databaseArchive, String databaseName, String dataDir, String outputDir, String cmdArgs)

***DistributedCache.addCacheArchive(new URI(programDir), jc);***

Getting values from the Distributed Cache **Class:** public class RunnerMap extends Mapper<String, String, IntWritable, Text> **Method:** public void setup(Context context) ***Path[] local = DistributedCache.getLocalCacheArchives(conf);***

* 1. In previous projects we used Hadoop's TextInputFormat to feed in the file splits line by line to map tasks. In this program, however, we want to feed in a whole file to a single map task. What is the technique used to achieve this? Also, briefly explain what are the key and value pairs you receive as input to a map task and what methods are responsible for producing these pairs?  
     **The technique used to achieve this is to have a RecordReader that can return the complete file in the record’s value (instead of a file split). To do this, the *DataFileInputFormat* class extends FileInputFormat and overrides the createRecordReader method which creates an instance of the *FileRecordReader*. This class is a custom implementation of the RecordReader. During initialization of the *FileRecordReader*, the full path of the file is stored in the path class variable which is later used in the *getCurrentKey* method to extract and return the filename, and in the *getCurrentValue* method to return full path. These two methods, generate the key and value of the customized *DataFileInputFormat* which are submitted to the Map task as the filename and the filepath respectively.**
  2. Do you think this particular implementation will work if the input files are larger than the default HDFS block size? Briey explain why. [Hint: you can test what will happen by concatenating the same input file multiple times to create a larger input file in the resources/blast input folder]

**If the input files are larger than the default HDFS block size, then the file will be split into multiple blocks depending on the size.**

**We performed a test by creating a larger input file using the existing celllines file:**

for i in {1..1000}; do cat celllines\_1.fa >> celllines\_big.fa; done

**Then, we copied this file to HDFS and ran the fsck command to look at the details of the file. As we can see below, the file has two blocks because the total file size exceeds the default HDFS block size by 2 MB. The size of the second block is only the difference, not the full block.**/user/summer/HDFS\_blast\_input/celllines\_big.fa 69704000 bytes, 2 block(s): OK

0. blk\_-5053682609392578122\_1133 len=67108864 repl=1

1. blk\_2715290028991850017\_1133 len=2595136 repl=1

**This can help us answer the question above. When the files are larger than the default HDFS block size, then the storage subsystem deals with the blocks and the locations but the view of the file does not change. That is, the implementation should work with larger input files as well.**

**However, we need to consider also that for this particular project, working with larger input files have an impact on performance since each Map task is in charge of processing one file.**

**When the input files are too large (even if they are smaller than the HDFS default block size), the application with the current configuration fails because the map task hangs waiting for the BLAST binary to finish executing. The default timeout of the mapreduce task is 600 seconds. The error is shown below:**

Task Id : attempt\_201702162003\_0009\_m\_000000\_0, Status : FAILED

Task attempt\_201702162003\_0009\_m\_000000\_0 failed to report status for 600 seconds. Killing!

**As mentioned, this issue appeared even with files smaller than 64MB but large enough to take a long time to be processed by BLAST. To avoid this, we have to modify the value of the *mapred.task.timeout* to a large number or zero if we don’t want it to fail no matter the time it takes to run.**

* 1. If you wanted to extend this program such that all output files will be concatenated into a single file, what key and value pairs would you need to emit from the map task?   
     **The key of the Map task could be the path to the output files and the values could be the names of the output files (celllines\_1.fa, celllines\_2.fa, etc.). The value/pair then would be <path, filename>**  
       
     Also, how would you use these in the reduce that you would need to add? **The Reduce task would receive the key and values mentioned above and perform a concatenation of all the *celllines* files. Since the key (path) will be the same for all the key/value pairs, then only one Reduce task will run which will perform the concatenation of all files and store the result of the single concatenated file in HDFS.**

1. The 4 output FASTA files: celllines 1.fa to celllines 4.fa.

**The output files are located in the folder Hadoop\_BLAST\_Output**