B649 Cloud Computing Project 3 Report

Team Members: Yash Ketkar (yketkar@indiana.edu) | Neelam Tikone (ntikone@indiana.edu)

- **What is Hadoop Distributed Cache and how is it used in this program?**

  DistributedCache is a facility provided by the Map-Reduce framework to cache files (text, archives, jars etc.) needed by applications. The framework will copy the necessary files on to the slave node before any tasks for the job are executed on that node. Its efficiency stems from the fact that the files are only copied once per job and the ability to cache archives which are un-archived on the slaves. We first add the database and program archive to the distributed cache and generate the two absolute location filepaths. We then copy the database and the binary program executable from the HDFS to the local file system along with all the required dependencies to execute each map task.

- **Write the two lines that put and get values from Distributed cache. Also include the method and class information.**

To put the values in the Distributed cache we have the following line:

DataAnalysis.java

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

//Add the archive to be localized to the conf.
```

To get the values from the Distributed Cache we have the following line:

RunnerMap.java

```
Path[] local = DistributedCache.getLocalCacheArchives(conf);

//Here we get the path array of the localized caches containing the Blast binary and the db
```

- **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?**

Here we use a provided customized Hadoop MapReduce InputFormat DataFileInputFormat.java to generate key-value pairs of <filename, filepath on HDFS>. It uses the custom FileRecordReader which has the getCurrentKey() and getCurrentValue() method which produce these pairs.

- **Do you think this particular implementation will work if the input files are larger than the default HDFS block size? Briefly 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]**

We increased the input file size to greater than 64mb which is the default HDFS block size by concatenating the same input file to itself multiple times. We then executed it using the modified input file. We would have to modify the timeout property to be greater than the default 600 seconds as specified in the mapred-site.xml.

```
<property>
  <name>mapred.task.timeout</name>
  <value>1800000</value> <!-- 30 minutes -->
</property>
```

If we increase the timeout until we get the buffer where the file is entirely executed. We will get output result for the modified input file.

- **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? Also, how would you use these in the reduce that you would need to add?**

We can write the output using a fixed text value as the key and the outFile as the value. We can then send this output to the reducer. The reducer will have all the outFiles as the value which will then be concatenated and written to the output file.