CLOUD COMPUTING

Project 3: Hadoop Blast

Abhishek Naik and Govind Mishra

25th February, 2017

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

Hadoop Distributed Cache:

DistributedCache is a facility provided by the MapReduce framework to cache the files needed by various applications. It is the responsibility of the applications to specify the files to be cached via URLs. It is assumed by DistributedCache that the file is already present at the location pointed to by the URLs and that every machine in the cluster has access to it.

All the necessary files are copied onto the system by the slave nodes before the tasks for any job are run. The files are copied only once per job. This makes the system efficient. It also helps in saving the network bandwidth.

Its usage in the program:

The input format consists of FASTA files which is used for denoting the peptide or nucleotide sequences. The individual amino acids are represented using single characters. The *FileRecordReader* and *DataFileInputFormat* classes get the input and create <key, value> pairs. The 4 input files make use of Hadoop’s Distributed Cache.

1. **Write the two lines that put and get values from the Distributed Cache. Also include the method and class information.**
2. The line that **puts** values into the Distributed Cache:

Code segment:

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

Method name:

*void launch(int numReduceTasks, String programDir, String execName, String workingDir, String databaseArchive, String databaseName, String dataDir, String outputDir, String cmdArgs)*

Class:

*DataAnalysis*

1. The line that **gets** values from the Distributed Cache:

Code segment:

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

Method name:

*void setup (Context context)*

Class:

*RunnerMap*

1. **In previous projects, we used Hadoop’s TextInputFormat to feed in the file splits line by line to the 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 are receive as input to a map task and what methods are responsible for producing these pairs?**

We use *DataFileInputFormat*, which invokes the *FileRecordReader* class, which in turn extends the *RecordReader* class. We then define a custom input format and set *isSplittable* to false.

The key is the filename and the value consists of the absolute HDFS path for each of the files. The methods used are *getCurrentKey()* and *getCurrentValue()*, respectively. Both of these have been defined in the *FileRecordReader* class.

1. **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].**

The default block size in HDFS is set to be 64 MB. If the file sizes are greater, then HDFS will split them up into blocks of 64 MB. Out of the 4 input files, when we increased the size of two of them to around 70 MBs while making the rest two around 128 kBs, we observed that 6 input files were processed, as expected. However, the code did not run successfully, as denoted by the log file. It included many time limit exceeded errors. Thus, the approach clearly failed.

1. **If you wanted to extend this program such that all the 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?**

The map task gets the input files from HDFS and sends them to the Blast program. The below code segment copies the output to HDFS:

*fs.copyFromLocalFile(new Path(outFile), outputFileName)*

Now, in order to extend the program such that we output key and value pairs, we need to stop sending the output to the *OutputHandler* and instead send it as <key, value> pairs to the reducer class. We would require a single reducer for this. Thus, we need to set the number of reducers equal to one, as per the below code segment:

*job.setNumberOfReducer(1)*

**References:**

1. Hadoop docs: <https://hadoop.apache.org/docs/r2.6.3/api/org/apache/hadoop/filecache/DistributedCache.html>
2. Blast doc:

<http://blast.ncbi.nlm.nih.gov/Blast.cgi>