**PROJECT ONE : HADOOP BLAST REPORT**

In this project, we want to use bioinformatics tool Blast. In the mapper, we want to code the Blast program to execute one input file from our <key,value> pair that we downloaded from the HDFS. In the homework, we are asked to change the data input format that is the key, because we have the HDFS path that we can use this path to download the files from HDFS than pass it to the Blast program and execute it to generate an output file. After that, the output should be uploaded back to the HDFS. This is because of this Blast program is map only type MapReduce program.

The problem is getting the input files and returning the output files. We are provided to run 16 files with different names and each must be identical, but Blast can recognize local things, which means it can recognize only the local path. We need to generate a local path and the command to run the process inside the mapper program. Also output must be identical, we need to copy from local to copy file back to the HDFS in the code not using command line.

One of the important goals here is to customize the key value pairs as the file name of the key and the path of HDFS.

Initialize(): Is the first step when we generate the key value pairs. We will use input split in this step. Input split (by hadoop) contains the information of the input file but not as a path. We need to use this input split object to generate our path.

nextKeyValue(): We have 16 files, we use this step to go to the next file sequentially. The return value should return the current key on null if there is no current key.

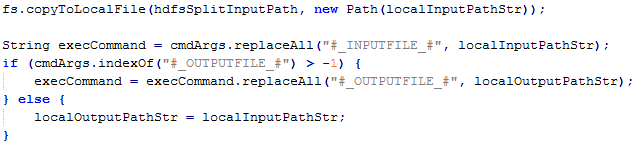
getCurrentKey(): We need to do substring to get just the file name and just pathName.substring(index + 1, pathName.length()). Then we return false or true if there is next key in the NextKeyValue().

In the RunnerMap, we upload the Blast to program HDFS, the we get the blast binary path from the distributed cache which is downloaded from local disk. We have the db ~/blast/db/nr, it should be done after we do the path analyze. Then we need to download the input files according to the key value pairs. After we have the input files and the database we run the jar to execute the binary blast to get the generated output. Then if there are some errors, we need to capture it from the process. At last we upload the output back to the HDFS. We need the do an application in the end to get outputs in one single file, we need to write a reducer or a script. Writing a reducer is better but we didn’t have time.

In the map code, we needed to use a setup function to run it after the mapper which is doing any job tasks. We needed to get a configuration parameter from our previous main program which is the hadoop blast java. Then we need to get a parameter from the distributed cache. The first parameter is the LocalDB and the other one is the binary blast path.



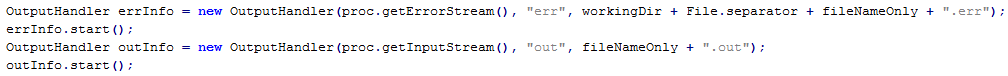
In the mapper, we needed the specify a function, when we have the database and the binary blast path we do some string argument to replace some parts of the command. This is how we handled with the input and output file names: We replace the files names with the #\_INPUTFILE\_# and #\_OUTPUTFILE\_# in the command line.



And we run the process by using process p and execCommand.



We need to write the outputs to a file. So, we are given an output handler class which we can provide a stream, file type and file name that we stored on the local disk.



Distributed Cache :

Hadoop provides a mechanism called the **distributed cache**. The distributed cache can contain small data files needed for initialization or libraries of code that may need to be accessed on all nodes in the cluster.

Here are our performance results and speedup charts :

For local machines in LH (molerat and polaris)

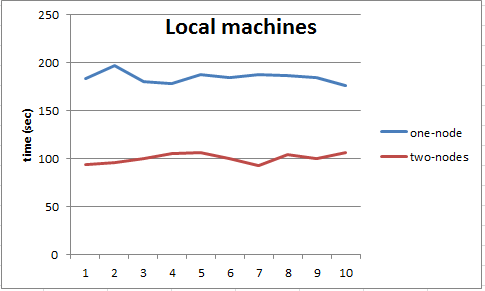
Configurations:

CPU: 2 cores (Intel(R) Core(TM)2 Duo CPU E8400 @ 3.00GHz)

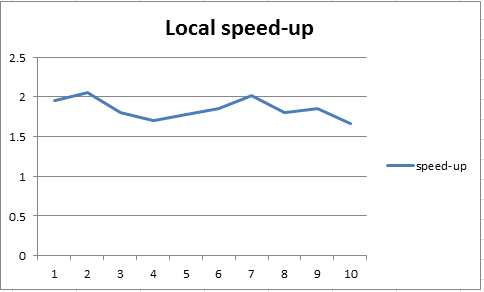
Mem:4G

Mapper: two for each

Reducer: two for each



Speed up chart:



For FutureGrid Eucalyptus (c1.xlarge)

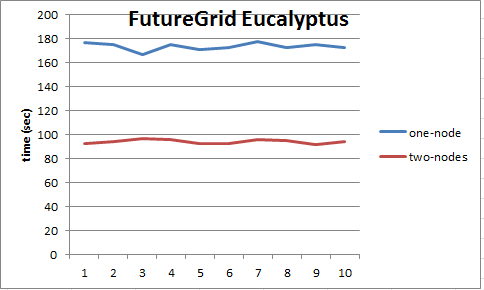
Configurations:

CPU: 8 \* 2.93G

Mem: 20G

Mapper: two for each

Reducer: two for each



Speed up chart:

