Cloud Assignment 5
MapReduce

Group 20
Kabir Chhabra (2013CS50287)
Ankush Phulia (2014CS50279)
Rachit Arora (2014CS50292)
Vaibhav Bhagee (2014CS50297)

PART I - Checking Status of HDFS

1. First one needs to check whether the HDFS is correctly installed and up and running on all cluster machines.
2. Run the command 'jps' on the Namenode and the Datanode machines.
3. On the Namenode, the output should contain
    a. Namenode
    b. Secondary Namenode
    c. JobTracker
    d. Jps
4. On the datanode, the output should contain
    a. Datanode
    b. TaskTracker
    c. Jps

Possible Issues

- One possible problem here could be that the Namenode is not running and only the Secondary namenode is running. If this is the case, then trying to run a mapreduce job gives errors like the '*Java:NetConnect Exception'*.
- This can arise due to reasons like incorrect/broken configuration of network or temporary file directories. To rectify this, follow the following steps[1] :
    a. Temporarily suspend the running of HDFS by running the script 'stop-all.sh', located in the the 'bin/' folder of the hadoop installation directory.
    b. Check the processes running that are using the port which is used by master to connect to the slaves (9000 in our case).
        ■ This can be done by running ''sudo netstat -tulpn | grep :9000'
    c. Kill all such processes, if necessary, using 'sudo kill -9 <process-id>'
    d. Now, create a new directory to keep the temporary files for hadoop. For example 'sudo mkdir ~/hadoop_tmp'
        ■ Ensure the access permissions of this directory are appropriately set, or simply make it public, running 'sudo chmod 777 -R ~/hadoop_tmp'
    e. Ensure that hadoop is configured to use this new directory, by editing the 'conf/core-site.xml' present in the hadoop installation folder

        - Add the following to the file :
          <property>
          *<name>*hadoop.tmp.dir*</name>*
          <value>~/hadoop_tmp*</value>*
          </property>
    f. Now format the Namenode, use 'bin/hadoop namenode -format'
    g. Finally start up the cluster, via the script 'bin/start-all.sh'
- Another possible problem at this stage can be that on the datanode(s), the 'Tasktracker' process is running, but the 'Datanode' process is not. If this is the case, then trying to manually start up the datanode with the command 'hadoop datanode', yields an error like '*Java:NamespaceID Mismatch Exception*'.
- The reason for this error is due to the differing versions of metadata on the namenodes and datanodes. Formatting the namenode updates the version stored on the namenode, but might not change that on the datanodes. It can be rectified as follows :
    a. First open the file 'dfs/name/data/current/VERSION' on the Namenode, and record the version mentioned in the first line of the file
    b. In each of the Datanodes, change the version in 'dfs/name/data/current/VERSION' to that recorded in the previous step
    c. Test if the Datanode process launches using the command 'hadoop datanode
    d. Run 'stop-all.sh' followed by 'start-all.sh' from the Namenode
    e. Verify, using 'jps', whether each Datanode has 3 processes running - 'jps', 'Tasktracker' and 'Datanode'

PART II - Map-Reduce for Word Count and Average Grade

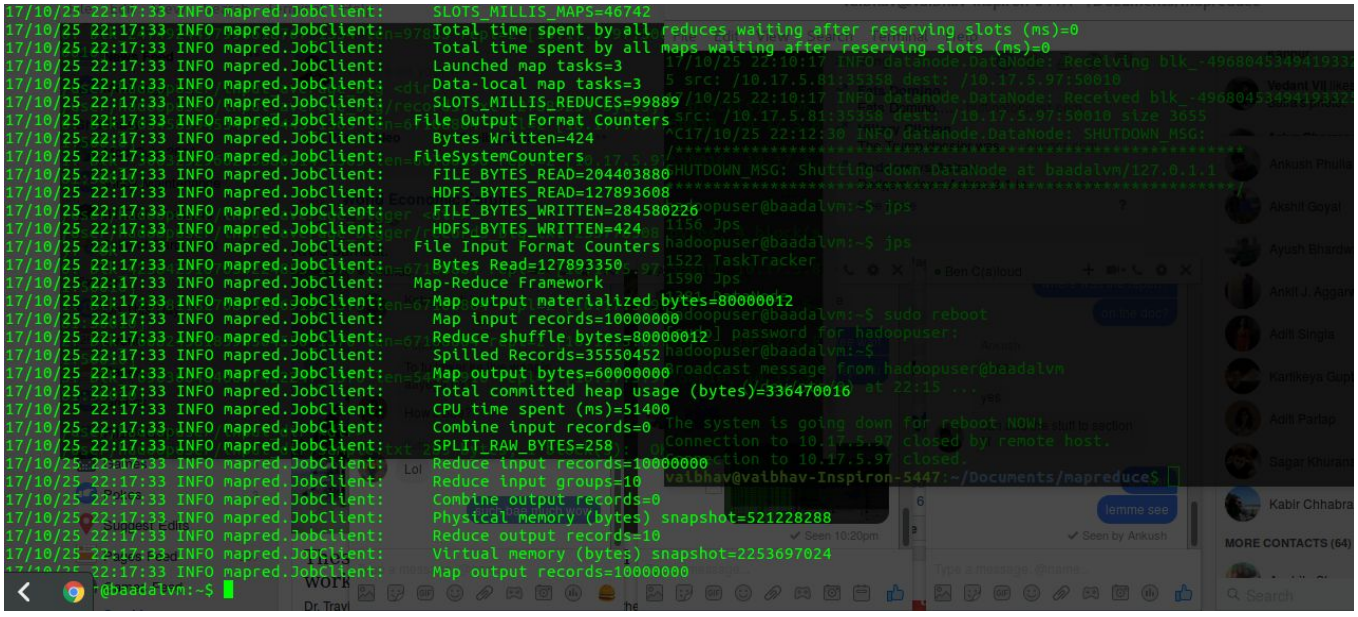
1. MapReduce in older hadoop versions does not require a particular framework to be installed. While installing hadoop on the VMs, map-reduce was configured by editing the file 'conf/mapred-site.xml' to specify the port. Additional properties may be added as necessary.
2. In order to run the mapreduce job, we follow the steps from the tutorialspoint link[2] (see the sources).
    a. Create a directory to stored compiled java classes. Eg. 'mkdir wordcount'
    b. Download Hadoop-core-1.2.0.jar, which is used to compile and execute the MapReduce program.
    c. Create input and output directories in hdfs -
        i. 'bin/hadoop fs -mkdir input_wordcount'
        ii. 'bin/hadoop fs -mkdir output_wordcount'
    d. Compile the the java file containing the actual methods for map and reduce - 'javac -classpath hadoop-core-1.2.0.jar -d wordcount WordCount.java'
    'jar -cvf wordcount.jar -C wordcount/'
    e. Put the input file (containing words) into hdfs - in the input directory - 'bin/hadoop fs -put words.txt input_wordcount'
    f. To verify the contents of the directory, the command 'bin/hadoop fs -ls wordcount_input/' can be used

g. Finally to run, use
‘bin/hadoop jar wordcount.jar WordCount wordcount_input wordcount_output’
h. To verify the files in the output folder, run ‘‘bin/hadoop fs -ls wordcount_output’
i. To see the output in ‘part-r-00000’ file, use the command ‘bin/hadoop fs -cat wordcount_output/part-r-00000’
j. To get the output in the local file system, we can copy it by executing ‘bin/hadoop fs -cat wordcount_output/part-r-00000/bin/hadoop dfs -get wordcount_output ~/’
k. For average grade, in the steps mentioned above, we just change names as
    i. ‘wordcount’ gets replaced by ‘average’
    ii. ‘input_wordcount’ -> ‘input_average’
    iii. ‘output_wordcount’ -> ‘output_average_1’
    iv. ‘WordCount.java’ -> ‘AverageGradeCompute.java’
    v. ‘WordCount’ -> ‘AverageGradeCompute’

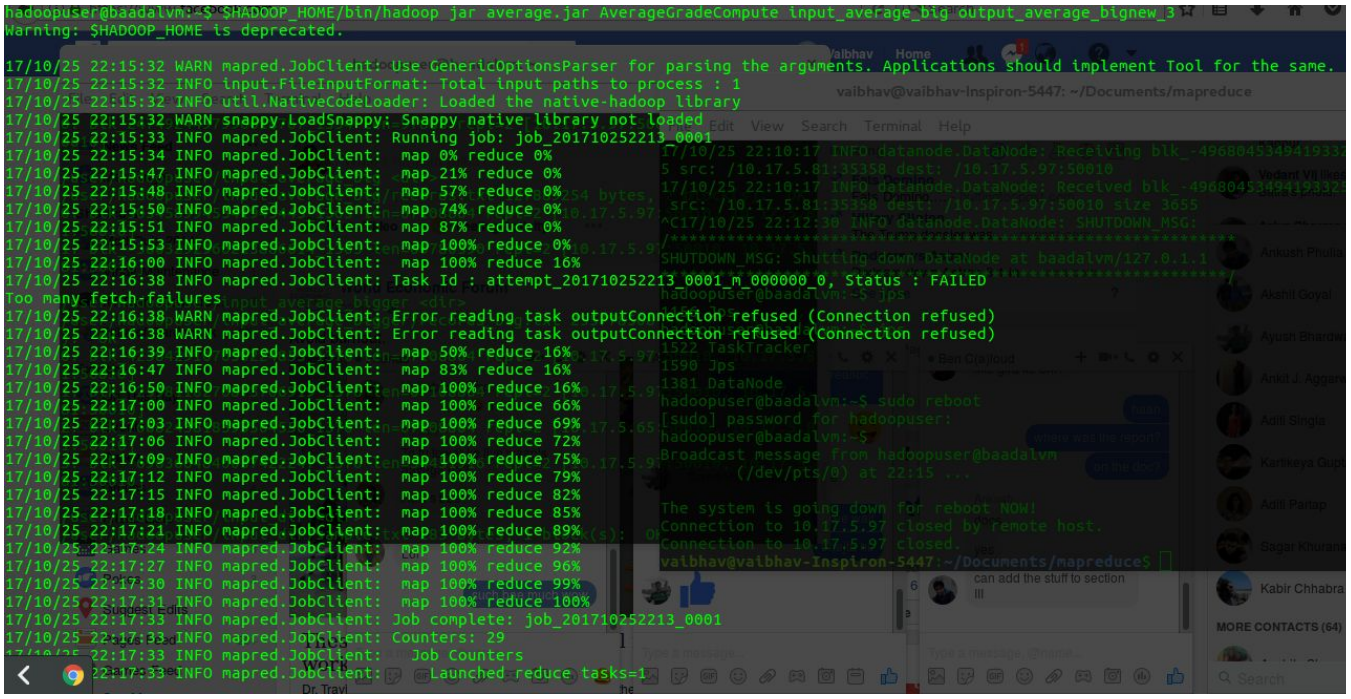
3. The java code for the word count has been borrowed from the apache documentation website - given as an example[3]
4. A python script was used to generate a table of 10,000,000 records of about ~127 MB. Each record has 3 fields - roll number, course code (0-10) and grade (0-100). The code for computing average grade for every course has been borrowed[4] and modified suitable for our use.

PART III - Fault Tolerance of MapReduce

1. On small sized files (< 64 MB), since only one block is required for storage, only one MapReduce job is required to be run, more than one VM may be needed to put down to observe fault tolerance mechanism in action.
2. In case of large files, spanning more than one block, multiple parallel map and reduce tasks were instantiated - keeping multiple nodes busy.
3. In our case, the file was ~127MB and need 3 map tasks, of which all were data-local, i.e. they were executed on machines which had the input data available locally. All this took about ~51 seconds



4. Repeats occurred due to mostly connection errors and the like, as reduce workers could not read the map output, causing reduce tasks, and some of the map tasks as well, to be repeated.
5. When the job was in progress, the command ‘sudo reboot’ was used on one of the VMs, to restart it. This caused the pending tasks to stall for a bit, as some progress was lost, but soon enough these were repeated.



6. Eventually, after many repeats, reduce operation completed successfully, and the data was stored on hdfs.

Sources

1. https://stackoverflow.com/questions/8076439/namenode-not-getting-started
2. https://www.tutorialspoint.com/map_reduce/implementation_in_hadoop.htm
3. https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
4. http://www.devinline.com/2015/12/find-total-and-average-salary.html