COL 733: CLOUD COMPUTING TECHNOLOGY FUNDAMENTALS

Assignment 5
Map Reduce

Group 11

Members:

- Ankit Shubham
  2014CS10158
- Ayush Gupta
  2014CS50281
- Kapil Kumar
  2014CS50736
- Deepak Bansal
  2014CS50435

Setting up MapReduce Framework and running a task

1. **Configure mapred-site.xml** → This controls the configuration settings for MapReduce daemons. Here we need to ensure that we will be using YARN framework. Also we will configure the MapReduce Job History server.

2. **Setup HADOOP_CLASSPATH** → Set this environment variable as ${JAVA_HOME}/lib/tools.jar for compiling the map reduce programs.

3. **Create Jar file**→ Set the current working directory as the hadoop installation folder and compile the map reduce program and make a jar file using following commands:

   ```
   $ bin/hadoop com.sun.tools.javac.Main WordCount.java
   $ jar cf wc.jar WordCount*.class
   ```

4. **Run the task using following command**→

   ```
   $ bin/hadoop jar wc.jar WordCount input_path output_path
   ```

MapReduce Program for Word Count

- We have made ‘TokenizerMapper’ class as the Mapper class. Its structure is as follows:
  Data members:
  - *one* (IntWritable) - stores an integer for the frequency of a word.
  - *word* (Text) - stores a string value.

  Member functions:
  - *map(Object key, Text value, Context context)* - It accepts the document name as ‘key’ and the contents of the document as ‘value’.

    It splits the ‘value’ string by whitespace and produces an iterator to get the split words one by one. Then it writes (word, 1) as the key value pair to be used by the reducer class.

- We have made ‘IntSumReducer’ class as the Reducer class. Its structure is as follows:
  Data members:
  - *result* (IntWritable) - stores an integer for the total frequency of a word

  Member functions:
  - *reduce(Text key, Iterable<IntWritable> values, Context context)* - It accepts a word as ‘key’ and a list of values which are the frequencies of this word.

    It iterates over ‘values’ and add all the frequencies to get the total frequency of the word ‘key’ and stores it in *result* variable. Then it writes (key, result) as the final output.

We ran the WordCount task on a text file of 90MB as input.
After shutting down a VM, while the MapReduce task is running, the task throws 'java.net.ConnectException' exception and tries to reconnect to the failed node again Then, after another failure, it connects with another replica of the same block and continues the task.
In our case, we switched off the node at ip **10.17.6.73**.

Following is the log after node shutdown:

```
17/10/21 18:30:52 WARN impl.BlockReaderFactory: I/O error constructing remote
block reader.
17/10/21 18:30:52 WARN hdfs.DFSClient: Failed to connect to /10.17.6.73:50010
for block, add to deadNodes and continue. java.net.ConnectException:
Connection refused
17/10/21 18:30:52 INFO hdfs.DFSClient: Successfully connected to
/10.17.6.18:50010 for BP-1035421894-127.0.1.1-1504946808916:
blk_1073741845_1021
```

It is clear from the logs that after shutdown of **10.17.6.73,** it tries to reconnect and after another failure, it successfully connects to **10.17.6.18,** which has the other replica of the block which was being processed on **10.17.6.73.**

So, the task completed successfully even after shutting down one node.

It took 1 minute to finish the task for a 90MB input file without interruption whereas 2 minutes when a node was shut down.

MapReduce Program for Average Grade

- We have made ‘TokenizerMapper’ class as the Mapper class. Its structure is as follows:
  Data members:
  - *score* (IntWritable) - stores an integer for the score of a student in a course.
  - *course_code* (Text) - stores a string value of course code of a course.

  Member functions:
  - *map(Object key, Text value, Context context)* - It accepts the document name as ‘key’ and the contents of the document as ‘value’.

    It splits the ‘value’ string by whitespace and produces an iterator to get the split words one by one. Then it extracts the roll number, course code and the score for each record one by one and writes (course_code, score) as the key value pair to be used by the reducer class.

- We have made ‘AverageReducer’ class as the Reducer class. Its structure is as follows:
  Data members:
  - *result* (FloatWritable) - stores a float value for the average score.

  Member functions:
  - *reduce(Text key, Iterable<IntWritable> values, Context context)* - It accepts a course code as ‘key’ and a list of values which the scores of students in the course with course code ‘key’.

    It iterates over ‘values’ and adds all the scores to get the total score and also keeps a counter for number of students/scores. Then calculates average score by dividing the total score by number of students and stores it in *result* variable. Then it writes (key, result) as the final output. Hence, it outputs average scores for each course.

Record generation:

- We have made a python script to generate records. It takes course code list and number of students as input and generates a file with the records.
- For each student and each course, it picks a random number between 0 and 100, to get the grade for that record.
- It also calculates the average grade for each course for the generated records.

Results:

Input: 8MB file with 5,00,000 records.
Output:

```
COL703  50.02765
COL733  49.96571
COL768  50.12852
COL783  49.95149
COL100  49.87263
```

Results were found to be consistent with the expected output.

It took 6 seconds to finish the task.

References

1. https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html