CSOE18 – Big Data Analytics

Name:-	Kshitij Kanade
Roll No:-	107121045
Date:-	02/12/2023
Batch:-	2025
Dept:-	EEE Sec A

Assignment - 2

Hadoop MapReduce for
Climate Data Analytics

Task_1 and 2 Answers

Aim/Objective:-

The objective of this project is to familiarize with Hadoop Installation and MapReduce and to apply them in real time data analytics.

Task #1:-

a. **What will the output pairs look like?**

The output pairs will be in the form of key-value pairs, where the key is a word and the value is the count of that word, which is always 1. This is because each occurrence of a word in the input text will be emitted with a count of 1.

b. **What will be the types of keys and values of the input and output pairs in the Map phase?**

The types of keys and values for the input and output pairs in the Map phase are as follows:

Input Key:	LongWritable (offset of the line in the input file)
Input Value:	Text (a line of text from the input file)
Output Key:	Text (a word from the line)
Output Value:	IntWritable (always 1)

c. **What will the input pairs look like?**

The input pairs for the Reduce phase will be grouped by key, and each key will have a list of values representing the counts of that key (word). For example, if the word "apple" appeared three times in the map output, the input to the reduce function would be ("apple", [1, 1, 1]).

d. **What will be the types of keys and values of the input and output pairs in the Reduce phase?**

The types of keys and values for the input and output pairs in the Reduce phase are as follows:

Input Key:	Text (a word)
Input Value:	Iterable<IntWritable> (an iterable collection of counts for that word)
Output Key:	Text (the same word)
Output Value:	IntWritable (the sum of counts for that word)

e. Write map () function for Questions a and b.

```
public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    String line = value.toString().replaceAll("[^a-zA-Z]+", " ");
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
    }
}
```

f. Write reduce () function for Questions c and d.

```
public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
        sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
}
```

These functions perform the necessary operations for word counting: in the map function, each word is emitted with a count of 1, and in the reduce function, the counts for each word are summed up to get the total count for that word. The combiner also performs a local aggregation of counts before sending them to the reducer.

Task #2:-

b. How many Map and Reduce tasks did running Word Count on Gberg-100M.txt produce? Run it again on Gberg-200M.txt and Gberg-500M.txt and write your observations. Additionally, run the following command on the cluster: $ hdfs getconf -confKey dfs.blocksize?

For Gberg-100M.txt;
Killed Map Tasks = 0
Launched Map Tasks = 1
Launched Reduce Tasks = 1

For Gberg-200M.txt;
Killed Map Tasks = 1
Launched Map Tasks = 3
Launched Reduce Tasks = 1

For Gberg-500M.txt;
Killed Map Tasks = 0
Launched Map Tasks = 4
Launched Reduce Tasks = 1

Observations:

- As the input size increases, the number of map tasks also increases. This is because larger files are split into more blocks, and each block is processed by a separate map task.
- The number of reduce tasks remains constant across different input sizes. This indicates that the data is not substantial enough to require additional reduce tasks.
- The overall execution time increases with the input size, which is expected due to the larger volume of data being processed.

HDFS Block Size (on running the above-mentioned command)
The HDFS block size is reported as **134,217,728 bytes (128 MB)**. This is a standard Hadoop block size, and it remains constant across different executions.

In summary, the number of map tasks increases with the input size, while the number of reduce tasks remains constant. This reflects the scalability of the MapReduce paradigm, allowing it to efficiently handle larger datasets by parallelizing the map tasks. The HDFS block size is a configuration parameter that influences how data is stored and distributed across the Hadoop cluster.

c. What is the link between the input size, the number of Map tasks, and the size of a block on HDFS?

The link between the input size, the number of Map tasks, and the size of a block on HDFS is influenced by the Hadoop framework's default block processing mechanism.

Input Size and Number of Map Tasks:

- The input size determines the number of Map tasks in a Hadoop job.
- Hadoop divides the input data into fixed-size blocks. Each block is typically 128 MB (134,217,728 bytes) by default, but this can be configured.
- Each Map task processes one block of data. Therefore, the number of Map tasks is proportional to the number of blocks created from the input data.

Size of a Block on HDFS:

- The size of a block on HDFS is a configurable parameter, and the default is 128 MB.
- When a file is stored in HDFS, it is split into blocks, and each block is stored on a separate DataNode in the Hadoop cluster.
- The block size affects the distribution of data across the cluster. Smaller blocks can lead to more parallelism but may increase metadata overhead, while larger blocks can improve data locality but reduce parallelism.

In summary, the relationship can be described as follows:

- The input size determines the number of blocks, and consequently, the number of Map tasks.

- The size of a block on HDFS influences how the input data is distributed across the cluster and can impact data locality and parallelism in the MapReduce job.

Further possible modifications:-

The mapper function can be modified to such that it keeps the words like “non-existent” without removing the hyphen but removing hyphen from all other places like “but-“, etc.

References:-

1. The Word Count program on MapReduce demonstrated in class.
2. Regular expressions application in Java.
3. Verbal Discussion with Classmates Rohit Meena, Harshit Malik, Atharv Bhavey, Akanksh Muvva, Vinay.
4. StackOverflow, GeeksForGeeks, etc online platforms.