Big Data Engineering

EXP 1 (Hadoop Installation & Map Reduce)

Name: Kresha Shah

SAP ID: 60009220080

INTRODUCTION:

Hadoop is an open-source software framework that is used for storing and processing large amounts of data in a distributed computing environment. It is designed to handle big data and is based on the MapReduce programming model, which allows for the parallel processing of large datasets.

What is Hadoop?

Hadoop is an open source software programming framework for storing a large amount of data and performing the computation. Its framework is based on Java programming with some native code in C and shell scripts.

Hadoop is an open-source software framework that is used for storing and processing large amounts of data in a distributed computing environment. It is designed to handle big data and is based on the MapReduce programming model, which allows for the parallel processing of large datasets.

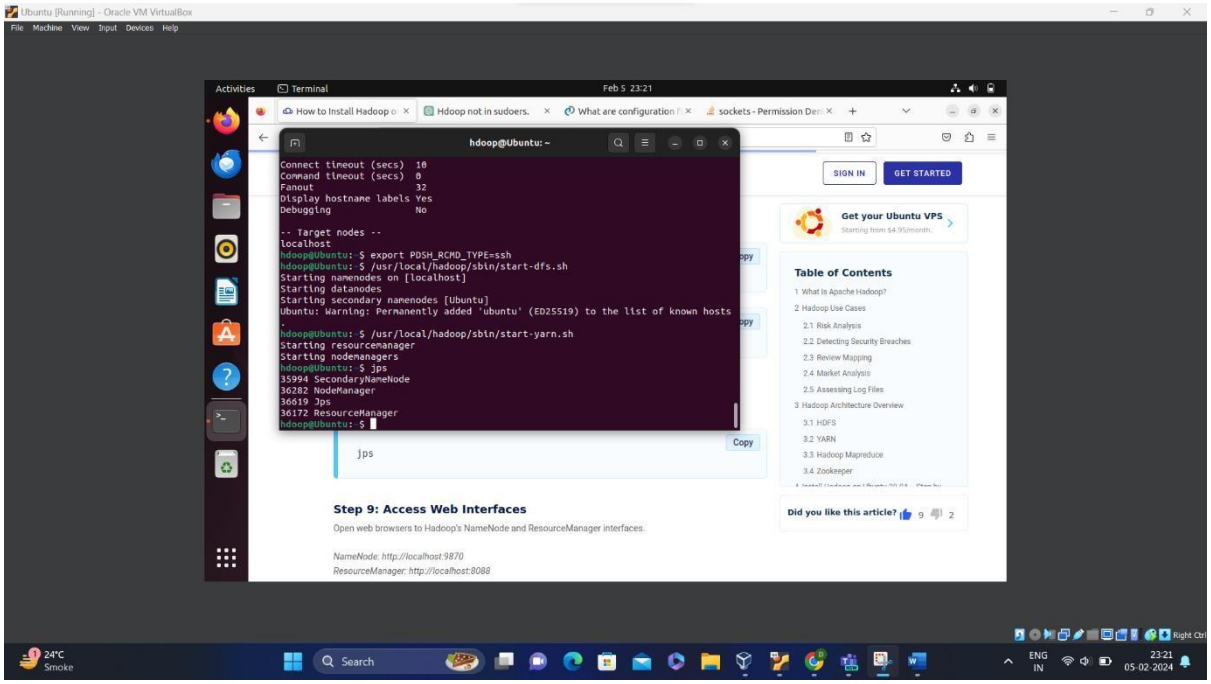
Hadoop has two main components:

- HDFS (Hadoop Distributed File System): This is the storage component of Hadoop, which allows for the storage of large amounts of data across multiple machines. It is designed to work with commodity hardware, which makes it cost-effective.
- YARN (Yet Another Resource Negotiator): This is the resource management component of Hadoop, which manages the allocation of resources (such as CPU and memory) for processing the data stored in HDFS.
- Hadoop also includes several additional modules that provide additional functionality, such as Hive (a SQL-like query language), Pig (a high-level platform for creating MapReduce programs), and HBase (a non-relational, distributed database).
- Hadoop is commonly used in big data scenarios such as data warehousing, business intelligence, and machine learning. It's also used for data processing, data analysis, and data mining. It enables the distributed processing of large data sets across clusters of computers using a simple programming model.

What is MapReduce?

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

Lab1:




```java
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

    public static class TokenizerMapper
            extends Mapper<LongWritable, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value, Context context
        ) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
```

```java
    public static class IntSumReducer
            extends Reducer<Text, IntWritable, Text, IntWritable> {

        private IntWritable result = new IntWritable();

        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);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
```

```java
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

FILE: Number of bytes read = 61
FILE: Number of bytes written = 279400
FILE: Number of read operations = 0
FILE: Number of large read operations = 0
FILE: Number of write operations = 0
HDFS: Number of bytes read = 546
HDFS: Number of bytes written = 40
HDFS: Number of read operations = 9
HDFS: Number of large read operations = 0
HDFS: Number of write operations = 2 Job Counters

Launched map tasks = 2
Launched reduce tasks = 1
Data-local map tasks = 2
Total time spent by all maps in occupied slots (ms) = 146137

Total time spent by all reduces in occupied slots (ms) = 441
Total time spent by all map tasks (ms) = 14613
Total time spent by all reduce tasks (ms) = 44120
Total vcore-seconds taken by all map tasks = 146137
Total vcore-seconds taken by all reduce tasks = 44120
Total megabyte-seconds taken by all map tasks = 149644288
Total megabyte-seconds taken by all reduce tasks = 45178880

Map-Reduce Framework

Map input records = 5
Map output records = 5
Map output bytes = 45
Map output materialized bytes = 67
Input split bytes = 208
Combine input records = 5
Combine output records = 5
Reduce input groups = 5
Reduce shuffle bytes = 6
Reduce input records = 5
Reduce output records = 5
Spilled Records = 10
Shuffled Maps = 2
Failed Shuffles = 0
Merged Map outputs = 2
GC time elapsed (ms) = 948
CPU time spent (ms) = 5160
Physical memory (bytes) snapshot = 47749120
Virtual memory (bytes) snapshot = 2899349504
Total committed heap usage (bytes) = 277684224

File Output Format Counters

Bytes Written = 40

Conclusion:

Implementing and deploying MapReduce code on a Hadoop cluster was a rewarding experience. We navigated through configuring Hadoop with attention to detail, ensuring compatibility and seamless execution of MapReduce jobs. Utilizing documentation and community resources proved invaluable

for troubleshooting and optimization. Witnessing Hadoop's scalability and fault tolerance affirmed its suitability for large-scale data processing. Integrating MapReduce with Hadoop highlighted the platform's effectiveness in leveraging distributed computing resources. Overall, this successful deployment underscores Hadoop's robustness and scalability for big data analytics.