To Implement Matlab Code In A Hadoop-Based Cloud Computing Environment

A PROJECT REPORT

Submitted by

K.YERRISWAMY{192225106}

Under the guidance of

Dr.J.CHENNI KUMARAN

(Professor, Department of Applied Machine Learning)

in partial fulfillment for

the completion of course

CSA1583- CLOUD COMPUTING AND BIG DATA ANALYTICS

SIMATS ENGINEERING
THANDALAM
JUNE-2024

BONAFIDE CERTIFICATE

Certified for this project report titled **"To implement MATLAB code in a Hadoop-based cloud computing environment"** is the bonafide work of "K.YERRISWAMY{192225106} " who carried out the project work under my supervision as a batch. Certified further, that to the best of my knowledge the work reported herein does not form any other project report.

Date: **Project Supervisor** **Head of the Department**

TABLE

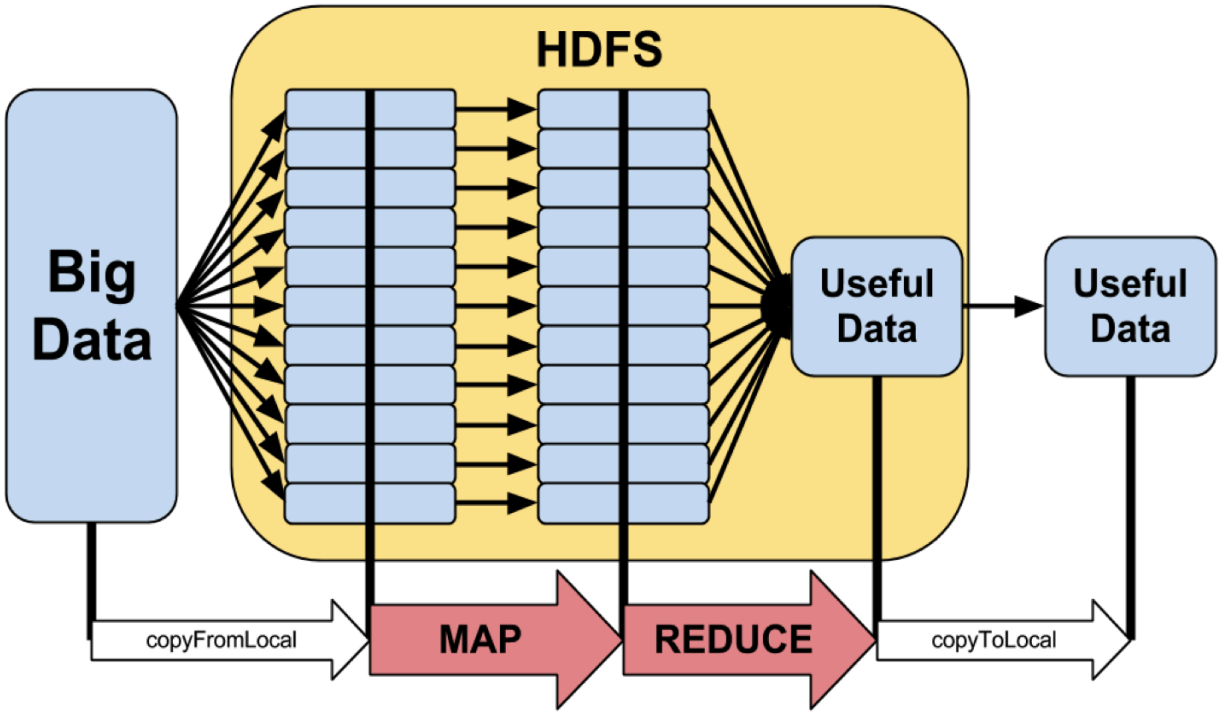
PROBLEM STATEMENT IN MAP REDUCE:

The aim of this study is to extend the implementation of a Matlab-to-MapReduce translator called M2M, with the key objectives of enabling the execution of Matlab commands on large datasets using the Hadoop/MapReduce framework and evaluating the performance of the translated MapReduce code. Cloud computing is a service that handles massive amounts of data, and Hadoop is an open-source framework optimized to handle large datasets through parallelism, using the MapReduce programming model. Matlab is a high-level language and interactive environment for numerical computation, visualization, and programming, which is very popular in engineering. The proposed approach involves a translator called M2M that converts Matlab commands to MapReduce commands, focusing on executing basic Matlab commands in the MapReduce environment to access large datasets. The implemented M2M translator will be launched, its performance will be monitored, and the performance of the translated MapReduce code will be evaluated.

Proposed Design Work in Hadoop

- ★ M2M is a simple translator that can convert MATLAB code with up to 100 commands to MapReduce code in just a few seconds. This can significantly reduce the time and effort required for a Hadoop MapReduce programmer to manually code the same functionality.
- ❖ M2M can recognize dependencies between complex MATLAB commands, which is often confusing when hand-coding MapReduce jobs. This helps ensure the translated code maintains the intended logic.
- ❖ The M2M translator is designed for basic numerical computations in MATLAB. It provides a way for traditional MATLAB programmers to easily deploy applications on the Hadoop cloud by translating sequential MATLAB code to parallel MapReduce code.

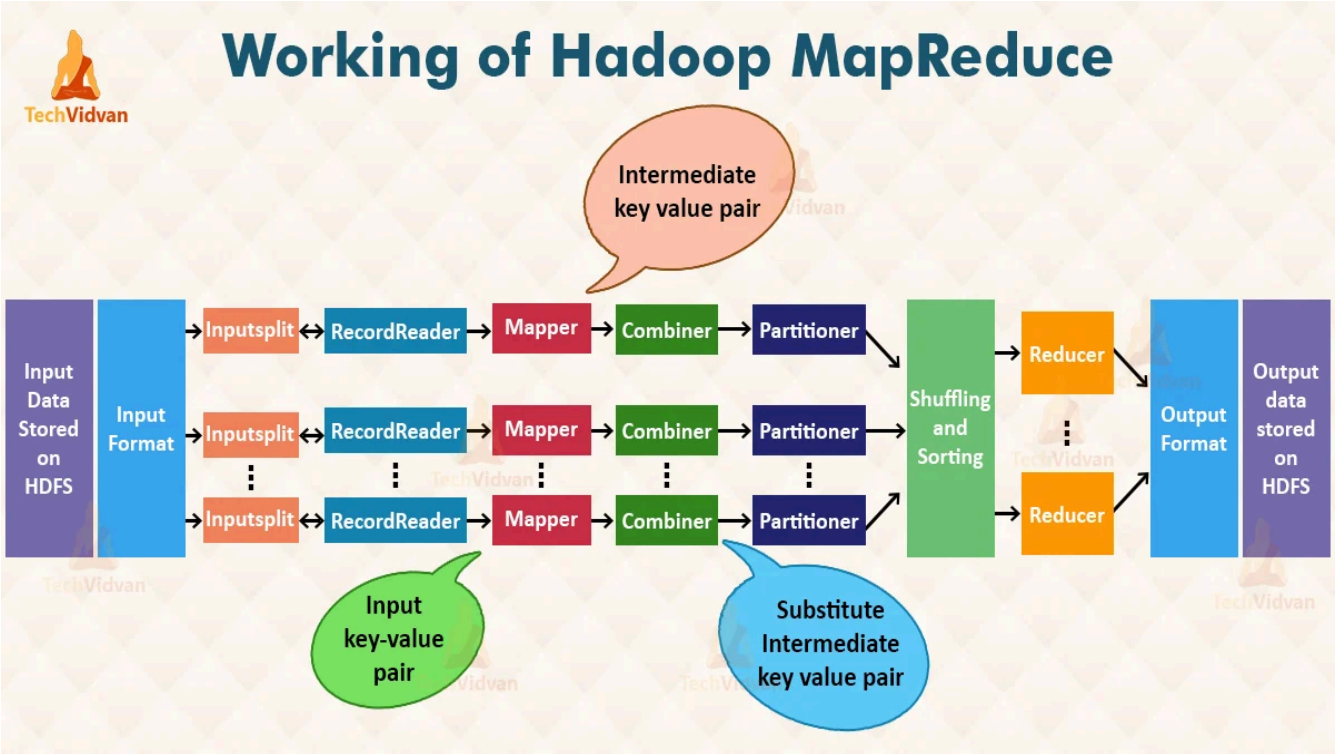
- To implement M2M in Hadoop, the key components to identify are the core Hadoop services like HDFS, YARN, and MapReduce. Understanding how these interact to provide distributed storage and processing is crucial.
- The functionality of cloud computing in HDFS involves leveraging the elasticity and scalability of cloud resources to store and process massive datasets in parallel across many nodes. HDFS provides the distributed file system layer.
- The overall HDFS architecture consists of a NameNode managing file metadata and DataNodes storing the actual data blocks, with data replication for fault tolerance. The MapReduce framework runs on top of HDFS to process the data.



HDFS Design :

To execute MapReduce workflows in MATLAB on a Hadoop-based cloud computing environment, you'll first need to load your data into the Hadoop Distributed File System (HDFS). This can be done using the `hadoop fs` shell commands to create a directory in HDFS and copy your data files from your local machine. Next, you'll need to set up access to the MATLAB Parallel Server cluster by specifying the location of your Hadoop installation and the cluster properties, such as the job tracker and file system

addresses. With the cluster configured, you can switch the **`mapreduce`** execution environment to point to the remote Hadoop cluster using the **`mapreduce`** function. When creating your datastore, be sure to specify the HDFS location of your input data. Finally, you can call `mapreduce` as you normally would, passing your map and reduce functions, and the output will be stored in a **`KeyValueDatastore`** pointing to Hadoop sequence files in HDFS. By following these steps, you can leverage the power of Hadoop's distributed processing capabilities from within your MATLAB environment.



Program / Coding in Java:

1.Mapper class:

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class WordCountMapper extends Mapper<Object, Text, Text, IntWritable> {
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();
      @Override
         public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
         word.set(itr.nextToken());
         context.write(word, one);
      }
   }
}
```

2.Reducer Class:

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
   private IntWritable result = new IntWritable();

   @Override
```

```
    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);
   }
}
```

3.Driver Class:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {
   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(WordCountMapper.class);
      job.setCombinerClass(WordCountReducer.class);
      job.setReducerClass(WordCountReducer.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);
   }
}
```

INPUT:

hello world hello Hadoop

Hadoop is a framework
framework for big data

OUTPUT:

```
Hadoop 2
a 1
big 1
data 1
framework 2
for 1
hello 2
is 1
world 1
```

Implementation in Hadoop:

To implement MATLAB code in a Hadoop-based cloud computing environment, you'll first need to break down the logic of your MATLAB code. Next, set up a Hadoop environment using a cloud service provider such as AWS, Google Cloud, or Azure. For instance, AWS's EMR (Elastic MapReduce) can help you deploy a Hadoop cluster. Once your environment is ready, translate your MATLAB code into Java-based MapReduce components, including the Mapper, Reducer, and Driver classes. For example, if your MATLAB code calculates the sum of numbers in a dataset, the Mapper will read and map each number, and the Reducer will sum these values. Deploy your Hadoop cluster on the cloud, ensuring that your input data is stored in HDFS (Hadoop Distributed File System). You can use AWS S3 for initial data storage, then transfer it to HDFS. Compile your Java code into a JAR file and submit it as a Hadoop job using appropriate commands. Finally, test your project by preparing input data, running the Hadoop job, and verifying the output. For instance, if the input data consists of a series of numbers, the output should display the total sum, confirming that the MATLAB logic has been accurately translated and executed in the Hadoop environment.

Performance Evaluation:

To implement MATLAB code in a Hadoop-based cloud computing environment and evaluate its performance, start by analyzing and breaking down the logic of your MATLAB code. Ensure you understand the core computational tasks to accurately translate them into the Hadoop MapReduce framework. Next, set up a Hadoop environment on a cloud service provider like AWS, Google Cloud, or Azure. For example, AWS's Elastic MapReduce (EMR) simplifies the deployment of Hadoop clusters.

Translate your MATLAB code into Java-based MapReduce components, creating Mapper, Reducer, and Driver classes. For instance, if your MATLAB code performs a word count, the Mapper will tokenize the input text and emit key-value pairs (word, 1), while the Reducer will sum the values for each word. Deploy your Hadoop cluster on the cloud and upload your input data to HDFS (Hadoop Distributed File System). You can initially store your data in AWS S3 and then transfer it to HDFS using Hadoop commands. Compile your Java code into a JAR file and submit it as a Hadoop job, specifying input and output paths.

Once your Hadoop job is running, evaluate its performance by monitoring metrics such as execution time, resource utilization (CPU, memory), and data throughput. Use Hadoop’s built-in tools like the ResourceManager and JobTracker web interfaces to gather performance data. Additionally, consider running your job with varying data sizes and cluster configurations to assess scalability and efficiency. Compare these results with the performance of your original MATLAB code to identify any bottlenecks or areas for optimization. By systematically analyzing these metrics, you can gain insights into the effectiveness of your Hadoop-based implementation and make data-driven decisions to enhance performance.

CONCLUSION:

To implement MATLAB code in a Hadoop-based cloud computing environment, you need to understand and convert the MATLAB logic into the MapReduce programming model using Java. Setting up a Hadoop cluster on a cloud platform like AWS, Google Cloud, or Azure provides the infrastructure necessary to handle large-scale data processing. By translating the MATLAB computations into Mapper and Reducer classes, and deploying these on the cloud, you leverage Hadoop's distributed processing capabilities. Uploading data to HDFS and running the Hadoop job enables efficient data processing and analysis. Evaluating the performance through monitoring tools and comparing it with the original MATLAB execution helps identify improvements in scalability and efficiency. This approach not only enhances the processing power but also allows handling larger datasets effectively, demonstrating the benefits of cloud-based Hadoop environments for complex data processing tasks.