Practical 12

Aim:

Design a distributed application using MapReduce which processes a log file of a system

---

Theory:

Introduction to Hadoop and MapReduce:

Hadoop is an open-source framework developed by the Apache Software Foundation for processing and storing large datasets in a distributed computing environment. It is designed to handle big data and provides scalable and fault-tolerant storage and processing capabilities.

Hadoop consists of four main modules:

1. **HDFS (Hadoop Distributed File System):** A distributed file system that provides high throughput access to application data.
2. **YARN (Yet Another Resource Negotiator):** Manages cluster resources and job scheduling.
3. **MapReduce:** A programming model for processing large datasets using parallel computing.
4. **Hadoop Common:** Provides common utilities that support the other Hadoop modules.

Understanding MapReduce

MapReduce is a programming model used for processing large-scale data across a Hadoop cluster. It divides the work into two main phases:

1. **Mapper Phase:** The input dataset is processed, and key-value pairs are generated.
2. **Reducer Phase:** The intermediate key-value pairs from the Mapper are aggregated to generate the final output.

Each MapReduce program follows a **divide and conquer** strategy, distributing tasks across multiple nodes in a Hadoop cluster to improve efficiency and speed.

Working of MapReduce

The execution of a MapReduce program involves the following steps:

1. **Splitting:** The input data is divided into smaller parts called input splits.
2. **Mapping:** Each input split is processed by the Mapper function to generate key-value pairs.
3. **Shuffling and Sorting:** The intermediate key-value pairs from the Mapper are shuffled and sorted based on the keys.
4. **Reducing:** The sorted key-value pairs are aggregated to produce the final result.

This framework enables parallel processing and is highly efficient for large-scale data processing tasks.

System Requirements:

To execute this practical, the following system requirements must be met:

- **Software:**
  - Java JDK 8 or later
  - Apache Hadoop (Standalone Mode)
  - Eclipse/IntelliJ (optional for development)
- **Hardware:**

- Minimum 4GB RAM
- Processor with multi-core capability
- Adequate disk space for Hadoop installation

- **Operating System:**
  - Windows/Linux/MacOS

Algorithm for Word Count using MapReduce:

Mapper Phase:

- Read the input text file line by line.
- Tokenize each line into individual words.
- Emit each word as a key with a count of 1.

Reducer Phase:

- Accept key-value pairs from the Mapper.
- Aggregate the count of occurrences of each word.
- Write the final output as (word, count).

Implementation Details:

Step 1: Setting up Hadoop in Standalone Mode

Before executing the program, Hadoop must be installed and configured in standalone mode.

1. Download and install Hadoop.
2. Set environment variables such as HADOOP_HOME and JAVA_HOME.
3. Edit the Hadoop configuration files:
   - core-site.xml
   - hdfs-site.xml
   - mapred-site.xml
4. Format the Hadoop filesystem using the command:

hdfs namenode -format

5. Start Hadoop services:
6. start-dfs.sh

start-yarn.sh

Step 2: Writing Java Code for Word Count

1. **Create a Java class named WordCount.java.**
2. **Implement the Mapper class to process input text:**
   - Use StringTokenizer to split words.
   - Emit each word as key with a count of 1.
3. **Implement the Reducer class to aggregate counts:**
   - Sum up all occurrences of each word.
   - Write the final result.
4. **Define the driver class to configure and execute the MapReduce job.**

Step 3: Compiling and Running the Program

1. **Compile the Java code using javac:**

javac -classpath `hadoop classpath` -d . WordCount.java

2. **Create a JAR file:**

jar -cvf WordCount.jar -C . .

3. **Run the program using the Hadoop command:**

hadoop jar WordCount.jar WordCount /input /output

Code Explanation:

- **Mapper Class:** Extracts words from input text and assigns each a count of 1.
- **Reducer Class:** Aggregates the word occurrences and sums them up.
- **Driver Class:** Configures the Hadoop job, sets input and output paths, and starts execution.

Expected Output:

For an input file containing:

hello world
hello Hadoop
hello MapReduce

The output will be:

Hadoop 1
MapReduce 1
hello 3
world 1

Advantages of Using Hadoop for Word Count:

1. **Scalability:** Can process petabytes of data across multiple nodes.
2. **Fault Tolerance:** Data is replicated, ensuring reliability.
3. **Cost-Effective:** Uses commodity hardware to handle big data.
4. **Parallel Processing:** Improves efficiency by executing tasks concurrently.
5. **Data Locality:** Moves computation closer to the data, reducing network latency.

Conclusion:

This experiment demonstrated the implementation of a Word Count application using the Hadoop MapReduce framework. By utilizing parallel computing, Hadoop efficiently processes large datasets, making it an essential tool for big data analytics. The experiment successfully tokenized words, mapped them to key-value pairs, and reduced them to generate the final count.