Modern Education Society's Wadia College of Engineering Pune-01

Department of Computer Engineering

Name of Student:	**Class:**
Semester/Year:	**Roll No:**
Date of Performance:	**Date of Submission:**
Examined By:	**Experiment No:**

ASSIGNMENT No: 05

TITLE: Write a map-reduce program to count the number of occurrences of each alphabetic character in the given dataset. The count for each letter should be case-insensitive (i.e., include both upper-case and lower-case versions of the letter; Ignore non-alphabetic characters).

AIM: To design and implement a map-reduce program to count the number of occurrences of each alphabetic character.

OBJECTIVES: To help students understand and implement Map-reduce.

TOOLS REQUIRED:

- **Hardware:**
- **Software:** Open source operating system

THEORY:

MapReduce is a programming model and processing technique introduced by Google to handle large-scale data processing in a distributed manner across clusters of computers. It was designed to process massive datasets in parallel, enabling efficient handling of tasks such as indexing web pages, analyzing logs, and more. The model abstracts the complexities of parallelization, fault-tolerance, and data distribution, making it accessible to developers without requiring in-depth knowledge of distributed systems.

Key Concepts of MapReduce

1. **Distributed Computing**: In MapReduce, tasks are distributed across multiple machines (nodes). These machines work on chunks of data in parallel, which enhances the scalability and efficiency of the system.
2. **Fault Tolerance**: MapReduce frameworks (e.g., Hadoop) are fault-tolerant, meaning that if a node fails during execution, the task is reassigned to another node, ensuring data

reliability and consistency.

3. **Parallelism**: The entire process is parallelized, allowing multiple map and reduce tasks to run concurrently across different nodes, which is key to its efficiency and performance.

Architecture of MapReduce

The MapReduce process is composed of **two primary functions**:

1. **Map Function**: The *map* function processes input data and produces key-value pairs as intermediate outputs.
2. **Reduce Function**: The *reduce* function aggregates the intermediate outputs (key-value pairs) from the map function, generating the final result.

Step-by-Step Execution of MapReduce

1. Input Splitting

Before processing begins, the dataset is divided into smaller, manageable chunks called *input splits*. Each split is typically assigned to a separate machine for processing in the Map phase. This splitting is crucial for parallelization.

- For example, a large text file might be split into chunks of 64 MB or 128 MB depending on the system configuration.
- Input splits can range across multiple machines in the cluster, allowing distributed processing.

2. Map Phase

The **Map function** is applied to each input split. Each map task works on a portion of the input data and produces key-value pairs. These intermediate key-value pairs are generated by analyzing and processing the raw input.

Map Function Anatomy:

- **Input**: Takes a chunk of the input dataset.
- **Processing**: Processes this input and extracts relevant information to generate key-value pairs.
- **Output**: Emits intermediate key-value pairs, which are further passed on to the shuffle phase.

Example: Consider an input file containing a collection of sentences, and the task is to count the number of occurrences of each word. The Map function will:

- Read each line of the text.

- Tokenize the words in the line.
- Emit a key-value pair for each word, where the key is the word itself, and the value is 1.

Example output of the Map function for the sentence "Hello World Hello":
("hello", 1), ("world", 1), ("hello", 1)

3. Shuffle and Sort Phase

After the Map phase, the intermediate key-value pairs are passed through the **shuffle and sort phase**. This phase is crucial for organizing the data in preparation for the Reduce phase.

- **Shuffling**: During this stage, all values associated with the same key (across all mappers) are collected and grouped together. This step involves sending data across different nodes to bring the same keys together, even if they originated from different map tasks.
- **Sorting**: After shuffling, the framework sorts the data by the key. Sorting ensures that the data for each key is provided in an ordered fashion to the Reduce tasks.

Example: If multiple Map tasks emit pairs like:
("hello", 1), ("world", 1), ("hello", 1), ("data", 1)
The Shuffle and Sort phase will group the values for the same key together:
("hello", [1, 1]), ("world", [1]), ("data", [1])

4. Reduce Phase

The **Reduce function** processes the grouped key-value pairs. It aggregates the values for each key (from the Shuffle and Sort phase) and performs the final computation to generate the output.

Reduce Function Anatomy:

- **Input**: Takes a key and a list of values associated with that key.
- **Processing**: Aggregates the values for each key to produce a single result.
- **Output**: Emits the final result as key-value pairs.

In the word count example, the Reduce function will sum the list of values for each word:

- For the key "hello" with values [1, 1], it will produce the output ("hello", 2).
- For the key "world" with the value [1], it will produce the output ("world", 1).

Example output of the Reduce function:
("hello", 2), ("world", 1), ("data", 1)

5. Final Output

The output from all the Reduce tasks is written to a distributed storage system, such as the

Hadoop Distributed File System (HDFS). This output represents the final result of the computation, and depending on the nature of the task, it can be written to a single file or multiple files across different nodes.

Conclusion:

MapReduce is a powerful model for distributed data processing. It simplifies large-scale data analysis by abstracting the complexities of parallelization, distribution, and fault tolerance. While it has certain limitations, especially for real-time or iterative workloads, it remains a foundational technique for batch processing of vast datasets.

QUESTIONS:

Q1. What is MapReduce? Explain the overall workflow of MapReduce with an example.

Q2. What is the role of the Map function in a MapReduce program? How does it contribute to parallel data processing?

Q3. What is the purpose of the Reduce function in MapReduce? How does it aggregate the intermediate data?