Apache Hadoop Lab – 2

Objective

To understand how Hadoop's MapReduce programming model works by executing a word frequency analysis on social media posts. This lab will help students learn how to write and execute a Python-based MapReduce program using Hadoop Streaming, explore concepts such as Mapper, Reducer, Combiner, Partitioner, and how Hadoop utilizes HDFS and parallel computing in real-world applications like social media analytics.

2.1 Introduction to MapReduce

MapReduce is a distributed computing model used in Hadoop to process vast amounts of data by splitting tasks across many machines. It's divided into two core phases:

- Mapper Phase: Each mapper receives a split of data and processes it to generate intermediate key-value pairs.
- Reducer Phase: All intermediate data is grouped by key and sent to the reducer, which processes it to produce final output.

This model supports parallel processing, where many machines work together to finish a task faster similar to how multiple workers assemble different parts of a car on a production line.

2.2 Application Domain: Social Media Analysis

Imagine you're working at a social media company like Facebook or Twitter. Your job is to analyze millions of posts to find trending words. Each post is stored as a line in a file. You want to count how many times each word appears—ignoring common stop words like "the", "is", "and".

This is a common big data problem. Manual analysis would take forever. But with Hadoop's MapReduce framework, you can distribute this task across many machines and complete it in minutes.

2.3 Key components Involved

Mapper.py

This file takes a line of text (a post), cleans it by removing stop words, and outputs a list of word-count pairs like:

Movie 1
Amazing 1
Lunch 1

How Hadoop Uses Mapper:

Hadoop splits the input file into smaller blocks and sends each block to a Mapper. The Mapper reads the text line-by-line and emits (word, 1) for every meaningful word. These are intermediate results stored in local storage until all mapping is done.

Reducer.py

This file receives key-value pairs grouped by word and sums up the values:

Amazing 3
Lunch 5
Movie 6

How Hadoop Uses Reducer:

Once mapping is finished, Hadoop sorts all the keys and sends related key-value pairs to the same Reducer. Reducer then adds the counts and produces the final result.

2.4 Supporting Components for Efficiency

Combiner

A Combiner is an optional mini reducer that runs after the Mapper but before sending data to the Reducer. It summarizes data locally, reducing network traffic and speeding up processing. In our case, it could locally count "movie" as 3 before sending it to Reducer.
In Python streaming, you can set the reducer.py file as a Combiner too.

Partitioner

A Partitioner decides which intermediate key goes to which Reducer. This allows load balancing—if you have multiple Reducers, the Partitioner ensures no single Reducer is overwhelmed.
By default, Hadoop hashes the key and assigns it to Reducers. Advanced users can customize this behavior.

2.5 Step-by-Step Execution in Hadoop

Input File: social_media.txt:

A text file with Facebook-style status updates, such as:
"Just finished watching a great movie!
Lunch with friends hits different.
This movie changed my mood completely."

Sample text file has been attached to this lab.

Mapper Code – mapper.py:

```
#!/usr/bin/env python3
import sys
import re

# Set of common stop words
STOP_WORDS = set(["the", "is", "and", "a", "to", "in", "of", "on", "for", "with", "at", "an"])

# Read each line from Hadoop input stream
for line in sys.stdin:
    line = line.strip()
    words = re.findall(r'\w+', line.lower())  # Tokenize words
    for word in words:
        if word not in STOP_WORDS:
            print(f"{word}\t1")  # Emit (key, value) pair for Reducer
```

Explanation: Hadoop feeds each line to the Mapper via sys.stdin. The script tokenizes the line, removes common words, and prints each valid word with a count of 1. This file is also attached with the lab.

Reducer Code – reducer.py:

```
#!/usr/bin/env python3
import sys
```

```
current_word = None
current_count = 0

# Read each (word, count) pair
for line in sys.stdin:
    word, count = line.strip().split("\t", 1)
    count = int(count)

    if current_word == word:
        current_count += count
    else:
        if current_word:
            print(f"{current_word}\t{current_count}")
        current_word = word
        current_count = count

if current_word == word:
    print(f"{current_word}\t{current_count}")
```

Explanation: Hadoop groups the data by word and streams them line-by-line into the Reducer. The Reducer aggregates the counts for each word and prints the final total. This file is also attached with the lab.

2.6 Running the Lab

Step 1: Make the scripts executable.

```
chmod +x mapper.py
chmod +x reducer.py
```

Step 2: Upload file to HDFS

```
hdfs dfs -mkdir -p /user/social
hdfs dfs -put social_media.txt /user/social
```

Step 3: Run the MapReduce job using Hadoop Streaming API

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \
-files mapper.py,reducer.py \
-mapper mapper.py \
-reducer reducer.py \
-input /user/social/social_data.txt \
-output /user/social/output
```

Step 4: View the Output

```
hdfs dfs -cat /user/social/output/part-00000
```

2.7 Summary of Learning

Concept	Explanation
MapReduce	Allows parallel processing of massive datasets by dividing tasks across many nodes.
HDFS	Provides a distributed storage system for input/output files.
Mapper	Breaks data into (key, value) format.
Reducer	Aggregates and finalizes output.
Combiner	Optional mini reducer to reduce data before shuffle.
Partitioner	Balances load by sending keys to specific reducers.
Streaming API	Let you write MapReduce jobs in Python or other languages.