Apache Hadoop Lab – 3

Lab Objective

To demonstrate how a Combiner optimizes network transfer and how a Partitioner controls data distribution among reducers using a custom MapReduce Python program.

Problem Statement

We have a text dataset where each line starts with a category (e.g., “sports”, “tech”, “politics”), followed by a sentence. Our goal is to:

- Count the frequency of each word within its respective category.
- Use a Combiner to reduce network traffic.
- Use a custom Partitioner to group all words of the same category in one Reducer.

Sample Input (social_category.txt):

```
sports Messi scored twice in the game
tech AI is changing the future
sports Ronaldo missed a penalty
tech Python is widely used
politics The new law was passed today
sports Game day is exciting
```

What is a Combiner?

A Combiner is a mini-reducer that performs local aggregation on the Mapper output before it is sent to the Reducer. It’s used to reduce the volume of data shuffled across the network.

- Example: If multiple lines on a node contain "sports the" → the Combiner will locally sum up "sports the" → ("sports the", 3) instead of sending 3 separate pairs to Reducer.

How it fits in Hadoop?

- Hadoop runs the Combiner function between the Mapper and the shuffle/sort phase.
- It’s optional and not guaranteed to run, but can significantly improve efficiency.

What is a Partitioner?

A Partitioner decides which Reducer receives each (key, value) pair based on the key.

- Custom partitioners help us group similar data to the same reducer, like all "sports" records go to Reducer 0, "tech" to Reducer 1, etc.

How it fits in Hadoop:

- Partitioning happens after the Combiner, before the data is sent to Reducers.
- The default partitioner hashes the key and uses key.hashCode() % numReducers.
- You can write a custom partitioner to route data based on a specific logic (e.g., category name).

The Code

Mapper.py:

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

# Read each line from input
for line in sys.stdin:
    line = line.strip()
    if not line:
        continue

    # Split into category and message
    parts = line.split(maxsplit=1)
    if len(parts) < 2:
        continue
    category, message = parts
    words = re.findall(r'\w+', message.lower())

    # Emit composite key: category_word
    for word in words:
        print(f"{category}_{word}\t1")
```

Combiner.py:

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

current_key = None
current_count = 0

for line in sys.stdin:
    key, count = line.strip().split("\t", 1)
    count = int(count)

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

if current_key == key:
    print(f"{current_key}\t{current_count}")
```

Partitioner.java:

```
public class CategoryPartitioner extends Partitioner<Text, IntWritable> {
    public int getPartition(Text key, IntWritable value, int numPartitions) {
        String category = key.toString().split("_")[0];
        switch(category) {
            case "sports": return 0;
            case "tech": return 1;
            case "politics": return 2;
            default: return 3;
        }
    }
}
```

Note:
In Hadoop Streaming, you can simulate this by using multiple reducers and splitting by prefix, but not via Python-partitioner logic.

Reducer.py:

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

current_key = None
current_count = 0

for line in sys.stdin:
    key, count = line.strip().split("\t", 1)
    count = int(count)

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

if current_key == key:
    print(f"{current_key}\t{current_count}")
```

Commands to Run

```
# Upload the file to HDFS
hdfs dfs -put social_category.txt /input/social_category.txt

# Run the job with combiner and multiple reducers (simulate partitioning)
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \
  -input /input/social_category.txt \
  -output /output/wordcount_combined \
  -mapper mapper.py \
  -combiner combiner.py \
  -reducer reducer.py \
  -file mapper.py \
  -file combiner.py \
  -file reducer.py \
  -numReduceTasks 3

# Check the Output
hdfs dfs -ls /output/wordcount_combined
```

Output Example:

```
sports_game    1
sports_day     1
sports_messi   1
tech_ai 1
tech_python    1
politics_law   1
...
```

End of The Lab