TP08: Simulating MapReduce Jobs on Google Colab

kiram mohammed ali

Introduction

This practical session introduces the core logic of MapReduce by simulating its behavior directly in Python using Google Colab. No Hadoop or Spark installation is required. Instead, we implement the three conceptual phases manually:

- **Map** – extract key/value pairs from raw data
- **Shuffle/Group** – group values by key
- **Reduce** – aggregate grouped values

This TP covers:

1. Word Count (introductory MapReduce example)
2. Sales per Region (applied example)
3. Web Log Analysis (exercise)
4. Bonus tasks (URL count, response size sum, error filtering)

Dataset

The `weblogs.txt` file contains simulated HTTP request logs:

```
Date, Time, IP, Method, URL, Status, ResponseSize
2025-10-10,12:01:32,192.168.1.2,GET,/index.html,200,1024
2025-10-10,12:01:33,192.168.1.3,GET,/products.html,200,850
2025-10-10,12:01:35,192.168.1.4,GET,/contact.html,404,512
...
```

1. Mapper Implementation (Status Code Counter)

The goal is to output a pair `(status, 1)` for each log entry.

Code

```
def mapper(line):
    fields = line.strip().split(",")
    if len(fields) != 7 or fields[0].startswith('#'):
        return []
    status = fields[5]
    return [(status, 1)]
```

2. Shuffle Phase

```
from collections import defaultdict

def shuffle(mapped_data):
    grouped = defaultdict(list)
    for key, value in mapped_data:
        grouped[key].append(value)
    return grouped
```

3. Reduce Phase

```
def reducer(shuffled_data):
    reduced = {}
    for key, values in shuffled_data.items():
        reduced[key] = sum(values)
    return reduced
```

4. Combining Map → Shuffle → Reduce

```
mapped = []
with open("weblogs.txt", "r") as f:
    for line in f:
```

```
        mapped.extend(mapper(line))

grouped = shuffle(mapped)
reduced = reducer(grouped)

for code, count in sorted(reduced.items()):
    print(f"HTTP {code}: {count} requests")
```

Output

```
HTTP 200: 10 requests
HTTP 403: 2 requests
HTTP 404: 5 requests
HTTP 500: 3 requests
```

5. Bonus Task 1: Count Requests per URL

Mapper

```
def mapper2(line):
    fields = line.strip().split(",")
    if len(fields) != 7 or fields[0].startswith('#'):
        return []
    url = fields[4]
    return [(url, 1)]
```

Output

```
url /about.html: 2 requests
url /checkout: 3 requests
url /contact.html: 3 requests
url /images/logo.png: 2 requests
url /index.html: 5 requests
url /login: 2 requests
url /products.html: 3 requests
```

6. Bonus Task 2: Total Response Size per Status Code

Mapper

```
def mapper3(line):
    fields = line.strip().split(",")
    if len(fields) != 7 or fields[0].startswith('#'):
        return []
    status = fields[5]
    response_size = int(fields[6])
    return [(status, response_size)]
```

Reducer

```
def reducer3(shuffled_data):
    reduced = {}
    for key, values in shuffled_data.items():
        reduced[key] = sum(values)
    return reduced
```

Output

```
status 200: 8182 bytes
status 403: 128 bytes
status 404: 2560 bytes
status 500: 384 bytes
```

7. Bonus Task 3: Error-only Analysis (Filter out 200)

Mapper

```
def mapper4(line):
    fields = line.strip().split(",")
    if len(fields) != 7 or fields[0].startswith('#'):
        return []
```

```
    status = fields[5]
    if status == "200":
        return []
    return [(status, 1)]
```

Output

```
HTTP 403: 2 requests
HTTP 404: 5 requests
HTTP 500: 3 requests
```

Conclusion

Through these exercises, we simulated the internal operations of a MapReduce system using pure Python code. Each mapper extracts key–value pairs, the shuffle phase groups entries by key, and reducers aggregate the values. These patterns form the foundational logic of distributed processing frameworks such as Hadoop MapReduce and Apache Spark.