

Question 1 Report

Rohan Gore

~rmg9725

Files attached:

- mapper1.py, reducer1.py, job1_part-00000, job1_part-00001
- mapper2.py, reducer2.py, job2_output.txt,
- mapper3.py, reducer3.py, job3_part-00000, job3_part-00001
- mapper4.py, reducer4.py, job4_output

Employed a **multi-stage MapReduce approach**, with each job building upon the results of the previous one. For solving questions 1.1 and 1.2 the solution procedure has been broken down into **four MapReduce jobs**.

Overall Strategy

1. **Job 1: Calculate Station-Region Average Temperatures** - This job processes the raw temperature data to compute the average temperature for each station in each region.
2. **Job 2: Calculate Temperature Anomalies** - This job takes the original temperature readings and the average temperatures calculated in Job 1 to compute temperature anomalies for each reading.
3. **Job 3: Calculate Regional Volatility** - This job takes the output from Job 2 (temperature anomalies for each region) and calculates the standard deviation of these anomalies, which represents the temperature volatility for each region.
4. **Job 4: Find Top 3 Regions by Volatility** - This job processes the output from Job 3 to identify the three regions with the highest temperature volatility.

Detailed Approaches:

Job 1 Approach:

1. We create a composite key by combining the station ID and region code (StationID_RegionCode) which allows us to group temperature readings by both station and region simultaneously.
2. **Map Phase:** The mapper extracts the station ID, region code, and temperature from each record and emits key-value pairs with the composite key and temperature value.
3. **Reduce Phase:** The reducer receives all temperature readings for each station-region pair and calculates the average temperature by **maintaining running sums and counts**.
4. The final output consists of records with the format StationID_RegionCode\tAverageTemperature, **which will be used in Job 2**.

Job 2 Approach:

1. **Using the distributed cache mechanism to make the average temperatures calculated in Job 1 available to all mappers in Job 2.**
2. **Map Phase:** The mapper reads the original temperature data, looks up the corresponding station-region average temperature from cache, and calculates the anomaly.
3. **Reduce Phase:** The reducer collects all anomalies for each region, which will be used in subsequent jobs to calculate volatility.
4. The final output consists of records with the format RegionCode\tAnomaly, which will be used in Job 3 to calculate regional volatility.

Job 3 Approach:

1. We **process the temperature anomalies from Job 2** to calculate the standard deviation (volatility) for each region.
2. **Map Phase:** The mapper simply passes through the data without modification, preserving the region code and anomaly values.
3. **Reduce Phase:** The reducer collects all anomalies for each region, calculates the mean, and then computes the standard deviation using the formula $\sqrt{\text{sum}((x - \text{mean})^2) / n}$.
4. The final output consists of records with the format RegionCode\tVolatility, which will be used in Job 4 to identify the top regions.

Job 4 Approach:

1. We **process the region volatility data from Job 3** to identify the top 3 regions with the highest temperature volatility.
2. **Map Phase:** The mapper passes through the region code and volatility values without modification.
3. **Reduce Phase:** Using a **single reducer** (numReduceTasks=1), we collect all region-volatility pairs, sort them by volatility in descending order, and select the top 3 regions.
4. The final output consists of a ranked list of the top 3 regions with their volatility values, formatted as "Rank X: Region YYY - Volatility: Z.ZZZ".

Question 1.1: Documentation with Commands

Job 1: Calculate Station-Region Average Temperatures

1. Uploaded mapper1.py and reducer1.py code files

```
cat > mapper3.py << 'EOF'
--- code (attached mapper3.py) ---
EOF
```

```
cat > reducer3.py << 'EOF'
--- code (attached reducer3.py) ---
EOF
```

2. Upload temperature-data and shift it to HDFS

```
hdfs dfs -put temperature-data.txt hw1-rmg9725/data/
```

3. Make the scripts executable:

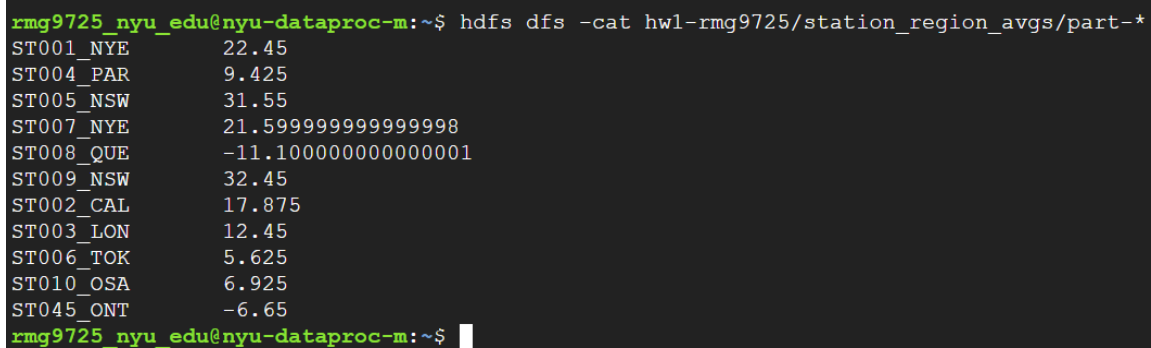
```
chmod +x mapper1.py reducer1.py
```

4. Run Job 1:

```
mapred streaming \
-files mapper1.py,reducer1.py \
-mapper mapper1.py \
-reducer reducer1.py \
-input hw1-rmg9725/temperature-data.txt \
-output hw1-rmg9725/station_region_avgs
```

5. Watch output:

```
hdfs dfs -cat hw1-rmg9725/station_region_avgs/part-*
```



A terminal window showing the output of the command `hdfs dfs -cat hw1-rmg9725/station_region_avgs/part-*`. The output lists 13 station-region pairs with their corresponding average temperatures. The prompt is `rmg9725_nyu_edu@nyu-dataproc-m:~$`.

Station-Region	Average Temperature
ST001_NYE	22.45
ST004_PAR	9.425
ST005_NSW	31.55
ST007_NYE	21.599999999999998
ST008_QUE	-11.100000000000001
ST009_NSW	32.45
ST002_CAL	17.875
ST003_LON	12.45
ST006_TOK	5.625
ST010_OSA	6.925
ST045_ONT	-6.65

6. Bringing file to local from hdfs and downloading for confirmation:

```
hdfs dfs -get hw1-rmg9725/station_region_avgs/part-00000
```

```
hdfs dfs -get hw1-rmg9725/station_region_avgs/part-00001
```

Output Files Attached: job1_part-00000.txt, job1_part-00001.txt,

Job 2: Calculate Temperature Anomalies

1. Combine all parts of output of job 1 on local

```
mkdir -p job1_output
cat job1_output/part-* > station_region_avgs.txt
```

2. Created and wrote mapper3.py and reducer3.py using following commands:

<pre>cat > mapper3.py << 'EOF' --- code (attached mapper3.py) --- EOF</pre>	<pre>cat > reducer3.py << 'EOF' --- code (attached reducer3.py) --- EOF</pre>
--	--

3. Make the scripts executable:

```
chmod +x mapper2.py reducer2.py
```

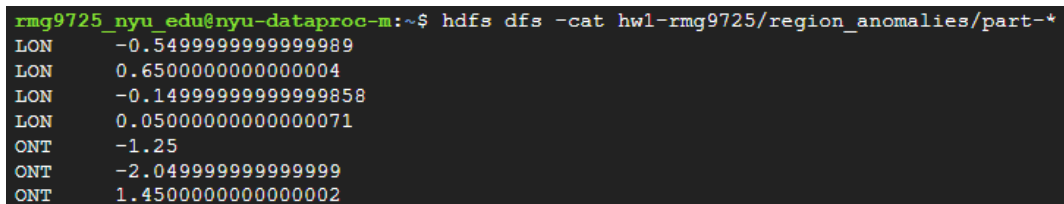
4. Run Job 2:

```
mapred streaming \
-files mapper2.py,reducer2.py,station_region_avgs.txt \
-mapper mapper2.py \
-reducer reducer2.py \
-input hw1-rmg9725/temperature-data.txt \
-output hw1-rmg9725/region_anomalies \
-cmdenv STATION_REGION_AVGS=station_region_avgs.txt
```

The `-cmdenv STATION_REGION_AVGS=station_region_avgs.txt` option sets an environment variable that tells the mapper where to find the station-region averages file.

5. Watch output:

```
hdfs dfs -cat hw1-rmg9725/region_anomalies/part-*
```



```
rmg9725_nyu_edu@nyu-dataproc-m:~$ hdfs dfs -cat hw1-rmg9725/region_anomalies/part-*
LON      -0.5499999999999989
LON      0.6500000000000004
LON      -0.14999999999999858
LON      0.05000000000000071
ONT      -1.25
ONT      -2.0499999999999999
ONT      1.4500000000000002
```

6. Downloading output files

```
rm part-00000
rm part-00001
hdfs dfs -getmerge hw1-rmg9725/region_anomalies region_anomalies_combined.txt
```

Output Files Attached: job2_output.txt

Question 1.2: Documentation with Commands

Job 3: Calculate Regional Volatility

1. Created and wrote mapper3.py and reducer3.py using following commands:

```
cat > mapper3.py << 'EOF'
--- code (attached mapper3.py) ---
EOF
```

```
cat > reducer3.py << 'EOF'
--- code (attached reducer3.py) ---
EOF
```

2. Make the scripts executable:

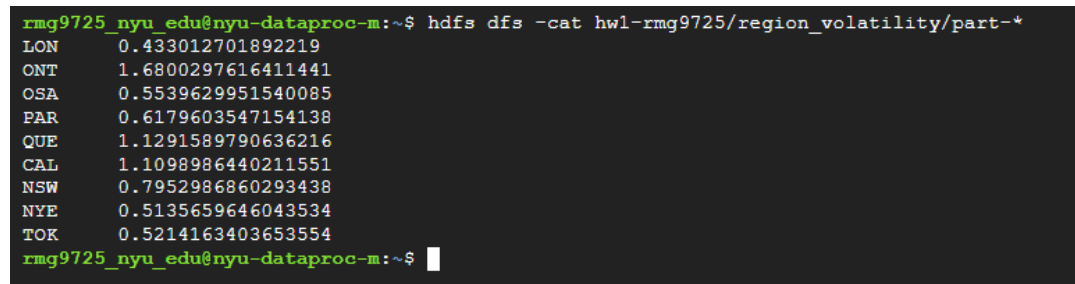
```
chmod +x mapper3.py reducer3.py
```

3. Run Job 3:

```
mapred streaming \
  -files mapper3.py,reducer3.py \
  -mapper mapper3.py \
  -reducer reducer3.py \
  -input hw1-rmg9725/region_anomalies \
  -output hw1-rmg9725/region_volatility
```

4. Watch output:

```
hdfs dfs -cat hw1-rmg9725/region_volatility/part-*
```



```
rmg9725_nyu_edu@nyu-dataproc-m:~$ hdfs dfs -cat hw1-rmg9725/region_volatility/part-*
LON      0.433012701892219
ONT      1.6800297616411441
OSA      0.5539629951540085
PAR      0.6179603547154138
QUE      1.1291589790636216
CAL      1.1098986440211551
NSW      0.7952986860293438
NYE      0.5135659646043534
TOK      0.5214163403653554
rmg9725_nyu_edu@nyu-dataproc-m:~$
```

5. Downloading output files for confirmation:

```
rm part-00000
```

```
rm part-00001
```

```
hdfs dfs -get hw1-rmg9725/region_volatility/part-00000
```

```
hdfs dfs -get hw1-rmg9725/region_volatility/part-00001
```

Output Files Attached: job3_part-00000.txt, job3_part-00001.txt,

Job 4: Find Top 3 Regions by Volatility

1. Created and wrote mapper4.py and reducer4.py using following commands:

```
cat > mapper4.py << 'EOF'
--- code (attached mapper4.py) ---
EOF
```

```
cat > reducer4.py << 'EOF'
--- code (attached reducer4.py) ---
EOF
```

2. Make the scripts executable:

```
chmod +x mapper4.py reducer4.py
```

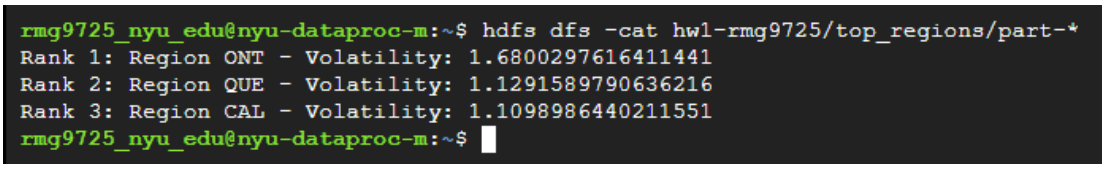
3. Run Job 4:

```
mapred streaming \
-files mapper4.py,reducer4.py \
-mapper mapper4.py \
-reducer reducer4.py \
-numReduceTasks 1 \
-input hw1-rmg9725/region_volatility \
-output hw1-rmg9725/top_regions
```

The `-numReduceTasks 1` parameter ensures all data is processed by a single reducer, allowing us to find the global top 3 regions.

4. Watch output:

```
hdfs dfs -cat hw1-rmg9725/top_regions/part-*
```



```
rmg9725_nyu_edu@nyu-dataproc-m:~$ hdfs dfs -cat hw1-rmg9725/top_regions/part-*
Rank 1: Region ONT - Volatility: 1.6800297616411441
Rank 2: Region QUE - Volatility: 1.1291589790636216
Rank 3: Region CAL - Volatility: 1.1098986440211551
rmg9725_nyu_edu@nyu-dataproc-m:~$
```

5. Download final output:

```
hdfs dfs -get hw1-rmg9725/top_regions/part-00000
```

Output File Attached: job4_output.txt

Addressed the Challenges

1. **Challenge: Calculate regional averages before determining anomalies, but you can't hold all data in memory.**
 - **Solution:** We split the calculation into two separate MapReduce jobs. Job 1 calculates the average temperature for each station in each region, and Job 2 uses these averages to calculate anomalies. This approach allows us to process datasets of any size without holding all data in memory at once.
2. **Challenge: There's no global state in MapReduce streaming, so computing standard deviations requires thought.**
 - **Solution:** We address this challenge by using a multi-stage approach. In Job 2, we collect all anomalies for each region, which will be used in Job 3 (in upcoming section) to calculate standard deviations. By breaking the computation into stages and passing intermediate results between jobs, we overcome the stateless limitation of MapReduce.
3. **Challenge: Computing top-3 regions requires aggregation across the entire dataset.**
 - **Solution:** This challenge is addressed in Job 4 (not covered in this report), where we use a single reducer to process all volatility values and find the global top 3 regions. By setting `-numReduceTasks 1`, we ensure that all data is processed by a single reducer, allowing us to find the global top 3 regions.

Meeting the Requirements

1. **Requirement: Implement this solution using MapReduce Streaming with any language of your choice (Python, Ruby, Perl, etc.)**
 - **Solution:** We implemented the solution using MapReduce Streaming with Python, which allows for rapid development and easy handling of text data.
2. **Requirement: Your solution must scale to handle datasets too large to fit in memory.**
 - **Solution:** Our solution scales to handle large datasets by:
 - Breaking the computation into multiple MapReduce jobs, each focusing on a specific task
 - Using the distributed nature of MapReduce to process data in parallel across multiple nodes
 - Minimizing memory usage in mappers and reducers by processing records one at a time
 - Using the distributed cache to efficiently share data between jobs
3. **Requirement: The solution can be solved using multiple MapReduce jobs that communicate through intermediate files, environment variables and/or command line arguments.**

- **Solution:** Our solution uses multiple MapReduce jobs that communicate through:
 - Intermediate files: The output of Job 1 (station-region averages) is used as input for Job 2
 - Environment variables: We use the `-cmdenv` parameter to pass the location of the station-region averages file to the mapper in Job 2
 - Command line arguments: We use the `-files` parameter to include the necessary scripts and data files in the job
- 4. **Requirement: Explain the data flow between your MapReduce jobs and how each overcomes the stateless limitation.**
 - **Solution:**
 - **Data Flow:** Job 1 processes the raw temperature data and outputs station-region average temperatures. Job 2 uses these averages along with the original data to calculate temperature anomalies for each region.
 - **Overcoming Stateless Limitation:**
 - ♣ We use the distributed cache to share the output of Job 1 with the mappers in Job 2
 - ♣ We use environment variables to tell the mappers where to find the shared data
 - ♣ We design our reducers to maintain state for the current key being processed, allowing us to perform aggregations like calculating averages
 - ♣ We break complex calculations (like standard deviation) into multiple jobs, with each job building on the results of the previous one

By carefully designing our MapReduce jobs and the data flow between them, we've created a solution that effectively addresses the challenges and meets all the requirements specified in the problem statement.