

Lab 5: Mini-MapReduce on Amazon EMR

Student: Aruzhan Saparkhankyzy

Group: IT-2306

1 Introduction

Big data processing requires scalable and fault-tolerant distributed systems. MapReduce is a programming model designed to process large datasets in parallel across multiple nodes. Amazon Elastic MapReduce (EMR) is a managed cloud service that simplifies running Hadoop-based workloads. The goal of this laboratory work is to deploy a Hadoop cluster on Amazon EMR, run a MapReduce job using Hadoop Streaming with Python, store data in HDFS, and analyze performance through experimentation.

2 Objectives

The main objective of this laboratory work is to study the MapReduce programming model and understand how it enables distributed processing of large-scale datasets. Another important goal is to deploy and configure a Hadoop cluster using Amazon Elastic MapReduce (EMR) in a cloud environment. The laboratory work also aims to implement and execute a WordCount MapReduce job using Python through Hadoop Streaming. In addition, the use of the Hadoop Distributed File System (HDFS) as a distributed storage system for both input and output data is examined. Finally, the performance of the MapReduce job is analyzed by conducting an experiment that evaluates how execution time changes with different input sizes.

3 Technology Stack

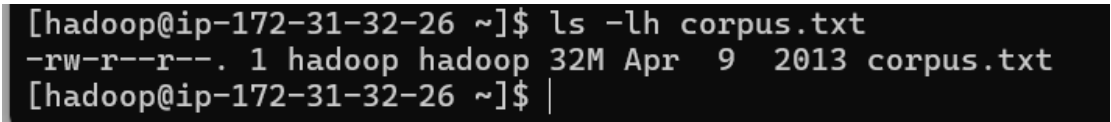
This laboratory work was implemented using Amazon Elastic MapReduce (EMR) as a managed cloud platform for running Hadoop. The MapReduce programming model was applied using Hadoop MapReduce, while the Hadoop Distributed File System (HDFS) was used for storing input and output data. Python was utilized for implementing the mapper and reducer through Hadoop Streaming. Cluster management and job execution were performed using Secure Shell (SSH) and the AWS Management Console.

4 Dataset Description

The dataset used in this laboratory is a small Wikipedia text dump from the Simple English Wikipedia. It contains textual articles suitable for word frequency analysis. The dataset was downloaded directly on the EMR master node and then uploaded to HDFS for distributed processing.

4.1 Dataset Preparation

```
wget https://github.com/LGDoor/Dump-of-Simple-English-Wiki/raw/refs/heads/master/corpus.tgz
tar -xvzf corpus.tgz
ls -lh corpus.txt
```



```
[hadoop@ip-172-31-32-26 ~]$ ls -lh corpus.txt
-rw-r--r--. 1 hadoop hadoop 32M Apr  9 2013 corpus.txt
[hadoop@ip-172-31-32-26 ~]$
```

Figure 1: Dataset download and extraction on the EMR master node.

5 Cluster Setup

An Amazon EMR cluster was created using the AWS Academy Learner Lab environment.

5.1 Cluster Configuration

The EMR cluster was configured using EMR version 7.12.0 with Hadoop version 3.4.1. The cluster consisted of one primary (master) node and one core (worker) node, which enabled distributed execution of MapReduce tasks. All instances were launched using the m4.large instance type supported in the AWS Academy environment. The cluster was created with the IAM roles EMR_DefaultRole and EMR_EC2_DefaultRole to ensure correct permissions and access.

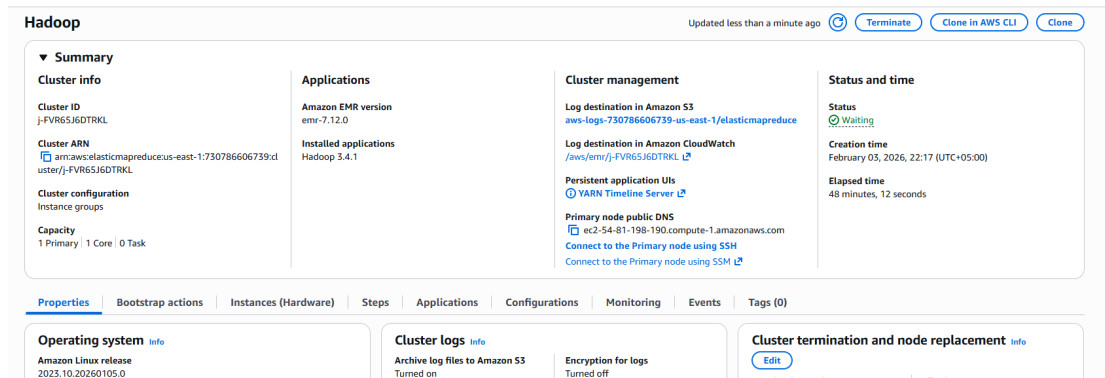


Figure 2: EMR cluster summary and configuration (version, nodes, roles).

5.2 Cluster Verification

After connecting to the master node via SSH, the cluster status was verified using YARN and HDFS administrative commands. The output confirmed that the cluster services were running correctly and that at least one node manager and datanode were available.

```
yarn node -list
hdfs dfsadmin -report | head -n 30
```

```
[hadoop@ip-172-31-32-26 ~]$ yarn node -list
hdfs dfsadmin -report | head -n 30
2026-02-03 18:06:31.957 INFO client.DefaultNoHARMFaloverProxyProvider: Connecting to ResourceManager at ip-172-31-32-26.ec2.internal/172.31.32.26:8032
2026-02-03 18:06:32.226 INFO client.AHSProxy: Connecting to Application History server at ip-172-31-32-26.ec2.internal/172.31.32.26:10200
Total Nodes:1
Node-Id Node-State Node-Http-Address Number-of-Running-Containers
ip-172-31-36-84.ec2.internal:8041 RUNNING ip-172-31-36-84.ec2.internal:8042 0
Configured Capacity: 28385980416 (26.44 GB)
Present Capacity: 28381491200 (26.43 GB)
DFS Remaining: 28342292480 (26.40 GB)
DFS Used: 39198720 (37.38 MB)
DFS Used%: 0.10%
Replicated Blocks:
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0
Low redundancy blocks (with replication factor 1): 0
Pending deletion blocks: 0
Erasure Coded Block Groups:
Low redundancy block groups: 0
Block groups with corrupt internal blocks: 0
Missing block groups: 0
Low redundancy blocks with highest priority to recover: 0
Pending deletion blocks: 0
-----
Live datanodes (1):
Name: 172.31.36.84:9866 (ip-172-31-36-84.ec2.internal)
Hostname: ip-172-31-36-84.ec2.internal
Decommission Status : Normal
Configured Capacity: 28385980416 (26.44 GB)
DFS Used: 39198720 (37.38 MB)
Non DFS Used: 4489216 (4.28 MB)
DFS Remaining: 28342292480 (26.40 GB)
DFS Used%: 0.10%
```

Figure 3: Cluster verification: YARN node list and HDFS report.

6 HDFS Data Storage

To ensure distributed processing, the input dataset was uploaded into HDFS. An input directory was created and the extracted `corpus.txt` file was copied to HDFS.

```
hdfs dfs -mkdir -p /user/hadoop/input
hdfs dfs -put -f corpus.txt /user/hadoop/input/
hdfs dfs -ls /user/hadoop/input
```

```
DFS Used%: 0.14%
[hadoop@ip-172-31-32-26 ~]$ hdfs dfs -ls /user/hadoop/input
Found 1 items
-rw-r--r-- 1 hadoop hdfsadmin group 32821626 2026-02-03 17:38 /user/hadoop/input/corpus.txt
[hadoop@ip-172-31-32-26 ~]$ |
```

Figure 4: The dataset stored in HDFS under `/user/hadoop/input`.

7 MapReduce Programming Model

The MapReduce programming model consists of three main stages. During the Map phase, input data is processed and transformed into intermediate key-value pairs. The Shuffle phase is automatically handled by the Hadoop framework and is responsible for grouping all intermediate values associated with the same key. Finally, during the Reduce phase, these grouped values are aggregated to produce the final output. This model enables parallel and scalable processing of large datasets across multiple nodes.

8 WordCount Implementation

The WordCount task counts the frequency of each word in the dataset. It was implemented using Hadoop Streaming with Python.

8.1 Mapper

The mapper reads lines of text, extracts alphabetic words, normalizes them to lowercase, and emits `(word, 1)` pairs.

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

for line in sys.stdin:
    line = line.strip().lower()
    for w in re.findall(r"[a-z]+", line):
        print(f"{w}\t1")
```

8.2 Reducer

The reducer receives grouped values for each word and sums them to produce total counts per word.

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

cur = None
```

```

cnt = 0

for line in sys.stdin:
    line = line.strip()
    if not line:
        continue
    w, c = line.split("\t", 1)
    c = int(c)

    if w == cur:
        cnt += c
    else:
        if cur is not None:
            print(f"{cur}\t{cnt}")
        cur = w
        cnt = c

if cur is not None:
    print(f"{cur}\t{cnt}")

```

9 Running MapReduce on EMR

The Hadoop Streaming job was executed on the EMR cluster. Before running the job, the output directory was removed to avoid conflicts.

```

hdfs dfs -rm -r -f /user/hadoop/output/wordcount

hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
  -D mapreduce.job.name="wordcount-streaming" \
  -input /user/hadoop/input/corpus.txt \
  -output /user/hadoop/output/wordcount \
  -mapper "python3_mapper.py" \
  -reducer "python3_reducer.py" \
  -file mapper.py \
  -file reducer.py

```

```

/usr/local/hadoop/output/wordcount
[hadoop@ip-172-31-32-26 ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming-jar \
-input /user/hadoop/input/corpus.txt \
-output /user/hadoop/output/wordcount \
-mapper "python3 mapper.py" \
-reducer "python3 reducer.py" \
-file mapper.py \
-file reducer.py
2026-02-03 18:07:42,795 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [mapper.py, reducer.py] [/usr/lib/hadoop/hadoop-streaming-3.4.1-anzn-4.jar] /tmp/streamjob3062634549799936648.jar tmpDir=null
2026-02-03 18:07:44,207 INFO client.DefaultHARMPFailoverProxyProvider: Connecting to ResourceManager at ip-172-31-32-26.ec2.internal/172.31.32.26:8032
2026-02-03 18:07:44,365 INFO client.AHSProxy: Connecting to Application History server at ip-172-31-32-26.ec2.internal/172.31.32.26:10200
2026-02-03 18:07:44,417 INFO client.DefaultHARMPFailoverProxyProvider: Connecting to ResourceManager at ip-172-31-32-26.ec2.internal/172.31.32.26:8032
2026-02-03 18:07:44,418 INFO client.AHSProxy: Connecting to Application History server at ip-172-31-32-26.ec2.internal/172.31.32.26:10200
2026-02-03 18:07:44,776 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/hadoop/.staging/job_1770139239401_0004
2026-02-03 18:07:45,167 INFO lzo.GPLNativeCodeLoader: Loaded native gpl library
2026-02-03 18:07:45,169 INFO lzo.LzoCodec: Successfully loaded & initialized native-lzo library [hadoop-lzo rev 049362b7cfc5ff5f739d6b1532457f2c6cd495e8]
2026-02-03 18:07:45,232 INFO mapred.FileInputFormat: Total input files to process : 1
2026-02-03 18:07:45,304 INFO mapreduce.JobSubmitter: number of splits:4
2026-02-03 18:07:45,586 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1770139239401_0004
2026-02-03 18:07:45,586 INFO mapreduce.JobSubmitter: Executing with tokens: []
2026-02-03 18:07:45,800 INFO conf.Configuration: resource-types.xml not found
2026-02-03 18:07:45,801 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2026-02-03 18:07:45,873 INFO impl.YarnClientImpl: Submitted application application_1770139239401_0004
2026-02-03 18:07:45,912 INFO mapreduce.Job: The url to track the job: http://ip-172-31-32-26.ec2.internal:20888/proxy/application_1770139239401_0004/
2026-02-03 18:07:45,913 INFO mapreduce.Job: Running job: job_1770139239401_0004
2026-02-03 18:07:54,118 INFO mapreduce.Job: Job job_1770139239401_0004 running in uber mode : false
2026-02-03 18:07:54,119 INFO mapreduce.Job: map 0% reduce 0%
2026-02-03 18:08:14,345 INFO mapreduce.Job: map 25% reduce 0%
2026-02-03 18:08:15,356 INFO mapreduce.Job: map 50% reduce 0%
2026-02-03 18:08:33,536 INFO mapreduce.Job: map 75% reduce 0%
2026-02-03 18:08:34,542 INFO mapreduce.Job: map 100% reduce 0%
2026-02-03 18:08:48,650 INFO mapreduce.Job: map 100% reduce 100%
2026-02-03 18:08:49,665 INFO mapreduce.Job: Job job_1770139239401_0004 completed successfully
2026-02-03 18:08:49,754 INFO mapreduce.Job: Counters: 55
File System Counters
  FILE: Number of bytes read=3516047
  FILE: Number of bytes written=9433858
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=32978218
  HDFS: Number of bytes written=1339515
  HDFS: Number of read operations=17
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=2
Job Counters
  Milled map tasks=1
  Launched map tasks=4
  Launched reduce tasks=1
  Data-local map tasks=4
  Total time spent by all maps in occupied slots (ms)=112111104
  Total time spent by all reduces in occupied slots (ms)=43726848
  Total time spent by all map tasks (ms)=72989
  Total time spent by all reduce tasks (ms)=14234
  Total vcore-milliseconds taken by all map tasks=72989
  Total vcore-milliseconds taken by all reduce tasks=14234
  Total megabyte-milliseconds taken by all map tasks=112111104
  Total megabyte-milliseconds taken by all reduce tasks=43726848
Map-Reduce Framework
  Map input records=254110
  Map output records=5502582
  Map output bytes=42846986
  Map output materialized bytes=4279406
  Input split bytes=484
  Combine input records=0
  Combine output records=0
  Reduce input groups=124584
  Reduce shuffle bytes=4279406
  Reduce input records=5502582
  Reduce output records=124584
  Spilled Records=11005164
  Shuffled Maps =4
  Failed Shuffles=0
  Merged Map outputs=4
  GC time elapsed (ms)=279
  CPU time spent (ms)=38890
  Physical memory (bytes) snapshot=2436632576
  Virtual memory (bytes) snapshot=17306226688
  Total committed heap usage (bytes)=2472542208
  Peak Map Physical memory (bytes)=529793024
  Peak Map Virtual memory (bytes)=3190763520
  Peak Reduce Physical memory (bytes)=369577984
  Peak Reduce Virtual memory (bytes)=4544094208
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=32977734
File Output Format Counters
  Bytes Written=1339515
2026-02-03 18:08:49,754 INFO streaming.StreamJob: Output directory: /user/hadoop/output/wordcount
[hadoop@ip-172-31-32-26 ~]$

```

Figure 5: Hadoop Streaming job execution and successful completion.

```

  FILE: Number of write operations=0
  HDFS: Number of bytes read=32978218
  HDFS: Number of bytes written=1339515
  HDFS: Number of read operations=17
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=2
  HDFS: Number of bytes read erasure-coded=0
Job Counters
  Milled map tasks=1
  Launched map tasks=4
  Launched reduce tasks=1
  Data-local map tasks=4
  Total time spent by all maps in occupied slots (ms)=112111104
  Total time spent by all reduces in occupied slots (ms)=43726848
  Total time spent by all map tasks (ms)=72989
  Total time spent by all reduce tasks (ms)=14234
  Total vcore-milliseconds taken by all map tasks=72989
  Total vcore-milliseconds taken by all reduce tasks=14234
  Total megabyte-milliseconds taken by all map tasks=112111104
  Total megabyte-milliseconds taken by all reduce tasks=43726848
Map-Reduce Framework
  Map input records=254110
  Map output records=5502582
  Map output bytes=42846986
  Map output materialized bytes=4279406
  Input split bytes=484
  Combine input records=0
  Combine output records=0
  Reduce input groups=124584
  Reduce shuffle bytes=4279406
  Reduce input records=5502582
  Reduce output records=124584
  Spilled Records=11005164
  Shuffled Maps =4
  Failed Shuffles=0
  Merged Map outputs=4
  GC time elapsed (ms)=279
  CPU time spent (ms)=38890
  Physical memory (bytes) snapshot=2436632576
  Virtual memory (bytes) snapshot=17306226688
  Total committed heap usage (bytes)=2472542208
  Peak Map Physical memory (bytes)=529793024
  Peak Map Virtual memory (bytes)=3190763520
  Peak Reduce Physical memory (bytes)=369577984
  Peak Reduce Virtual memory (bytes)=4544094208
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=32977734
File Output Format Counters
  Bytes Written=1339515
2026-02-03 18:08:49,754 INFO streaming.StreamJob: Output directory: /user/hadoop/output/wordcount
[hadoop@ip-172-31-32-26 ~]$

```

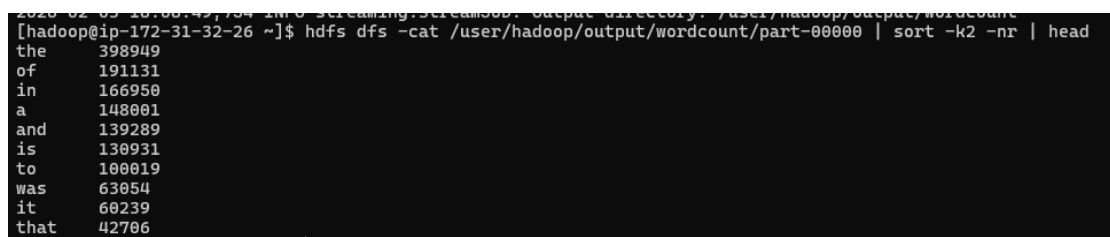
Figure 6: Hadoop Streaming job execution and successful completion.

10 Output Validation

After execution, the output was verified in HDFS. The output directory contained the `_SUCCESS` marker and the reducer output file `part-00000`. To validate correctness, the first lines were displayed and the top-10 most frequent words were extracted.

```
hdfs dfs -ls /user/hadoop/output/wordcount
hdfs dfs -head /user/hadoop/output/wordcount/part-00000

hdfs dfs -cat /user/hadoop/output/wordcount/part-00000 | sort -k2 -nr | head -
n 10
```



```
2020-02-05 10:00:49,784 INFO Streaming.StreamJob: Output directory: /user/hadoop/output/wordcount
[hadoop@ip-172-31-32-26 ~]$ hdfs dfs -cat /user/hadoop/output/wordcount/part-00000 | sort -k2 -nr | head
the      398949
of       191131
in       166950
a        148001
and      139289
is       130931
to       100019
was      63054
it       60239
that     42706
```

Figure 7: Output validation: HDFS output listing and top-10 most frequent words.

11 Experiment: Input Size Analysis

To analyze performance, an input-size experiment was conducted. The WordCount job was executed on a small input (approximately 1 MB) and on the full dataset (approximately 32 MB), and runtimes were measured using the `time` command.

11.1 Small Dataset

A smaller dataset was created by taking the first one million bytes from the original file and uploading it to HDFS. The WordCount job executed on the small dataset completed in **50.184 seconds**.

```
head -c 1000000 corpus.txt > corpus_small.txt
hdfs dfs -mkdir -p /user/hadoop/input_small
hdfs dfs -put -f corpus_small.txt /user/hadoop/input_small/

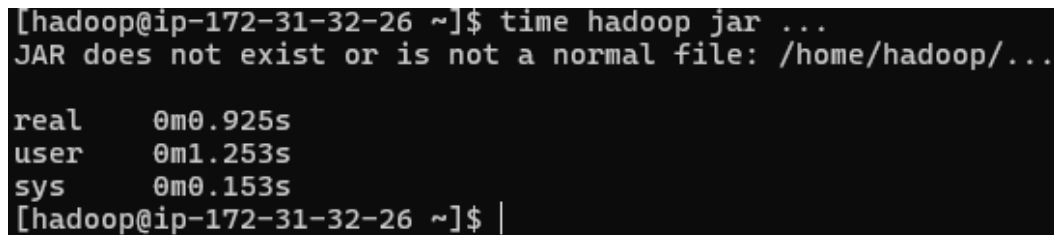
hdfs dfs -rm -r -f /user/hadoop/output/wordcount_small
time hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
  -input /user/hadoop/input_small/corpus_small.txt \
  -output /user/hadoop/output/wordcount_small \
  -mapper "python3_mapper.py" \
  -reducer "python3_reducer.py" \
```

```
-file mapper.py \  
-file reducer.py
```

11.2 Large Dataset

The same job was executed on the full dataset stored in HDFS. The runtime for the large dataset was **69.150 seconds**.

```
hdfs dfs -rm -r -f /user/hadoop/output/wordcount_big  
time hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \  
-input /user/hadoop/input/corpus.txt \  
-output /user/hadoop/output/wordcount_big \  
-mapper "python3_mapper.py" \  
-reducer "python3_reducer.py" \  
-file mapper.py \  
-file reducer.py
```



```
[hadoop@ip-172-31-32-26 ~]$ time hadoop jar ...  
JAR does not exist or is not a normal file: /home/hadoop/...  
  
real    0m0.925s  
user    0m1.253s  
sys     0m0.153s  
[hadoop@ip-172-31-32-26 ~]$ |
```

Figure 8: Experiment results: runtime comparison for small vs large input size.

11.3 Experiment Analysis

The experiment results show that increasing input size leads to increased execution time. This happens because a larger dataset produces more intermediate key-value pairs during the Map stage, which increases shuffle and sort overhead before the Reduce stage. Therefore, MapReduce performance scales with input volume, and runtime generally grows as the amount of processed data increases.

12 Conclusion

In this laboratory work, a Hadoop cluster was successfully deployed using Amazon EMR. A WordCount MapReduce job was implemented in Python using Hadoop Streaming and executed on the EMR cluster with input and output stored in HDFS. The output was validated by inspecting reducer results and extracting the most frequent words. Finally, an input-size experiment confirmed that MapReduce execution time increases

when processing a larger dataset due to higher map output and shuffle overhead. Overall, this laboratory demonstrates practical application of distributed computing concepts and performance analysis using MapReduce in a cloud environment.