
ECE4721J - Lab 2 Report

Methods and Tools for Big Data

Yiding Chang, Yifan Shen, Kexuan Huang, Qinhang Wu

June 4, 2022



1. Input File Generation

We first randomly generate 1000 students with `generate.py`. Then we use `grading.sh` and `grading.awk` to randomly assign student ID and grades for these 1000 students, with each student randomly appearing a number of times depending on the input data size. The input files we used are as follows,

Number of students	File Size
1000	29 KB
10000	287 KB
100000	2.8 MB
1000000	28.7 MB
10000000	286.9 MB
100000000	2.87 GB

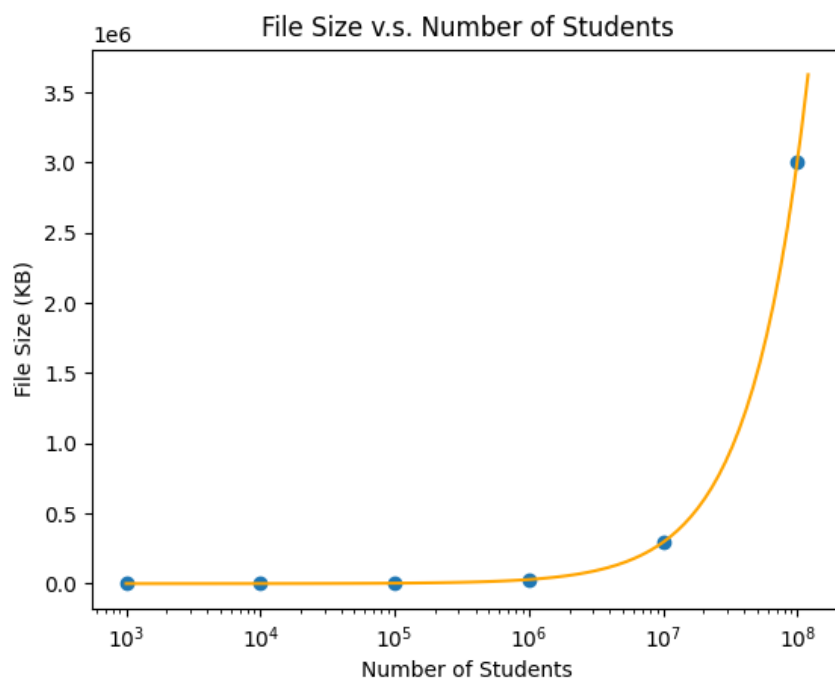


Figure 1: File Generation

2. Performance on a single computer

The CPU of the computer used in this session is 2.3 GHz Dual-Core Intel Core i5 and the RAM is 8 GB.

The command used is listed in the last section of this report. A sample output is attached as well.

The speed (total time in the unit of seconds) versus the number of student and the size of the file is as follows,

Number of students	File Size	Total Time (s)
1000	29 KB	4.088
10000	287 KB	4.826
100000	2.8 MB	5.189
1000000	28.7 MB	8.904
10000000	286.9 MB	55.917
100000000	2.87 GB	534

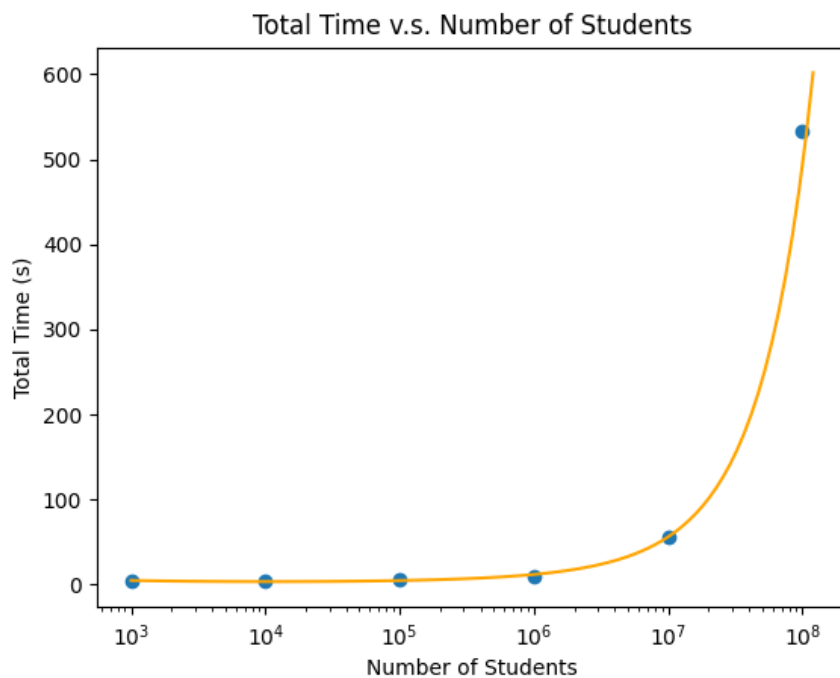


Figure 2: Single Performance

3. Performance on the group cluster

The Apache Hadoop is a framework supporting the distributed processing of large data sets across clusters of computers, which takes advantage of the MapReduce programming model that processes and generates big data sets with distributed algorithm on a cluster. MapReduce mainly consists of:

- Mapper: takes splitted input from the disk as `<key,value>` pairs, processes them, and produces another intermediate `<key,value>` pairs as output.
- Reducer: takes `<key,value>` pairs with the same key, aggregates the values, and produces new useful `<key,value>` pairs as output.

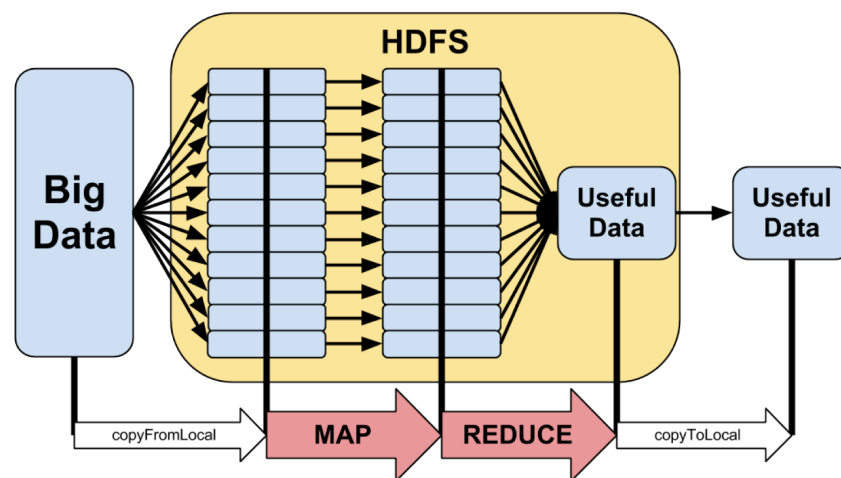
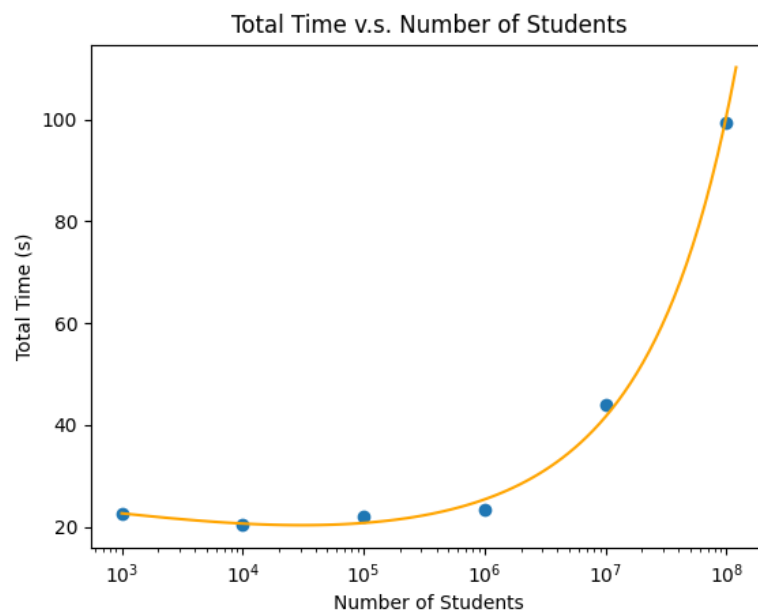
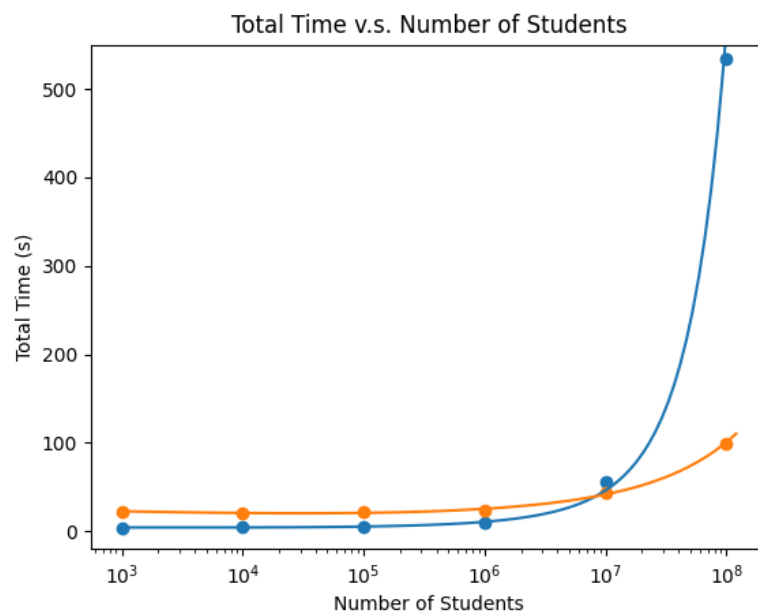


Figure 3: Hadoop MapReduce

With Hadoop cluster set up, we have 1 master and 2 slave for MapReduce tasks. The speed (total time in the unit of seconds) versus the number of student and the size of the file is as follows,

Number of students	File Size	Total Time (s)
1000	29 KB	22.6
10000	287 KB	20.3
100000	2.8 MB	22.0
1000000	28.7 MB	23.3
10000000	286.9 MB	44.0
100000000	2.87 GB	99.3

**Figure 4:** Cluster Performance**Figure 5:** Benchmark

We can find that Hadoop MapReduce is much more efficient for big data.

4. Hadoop MapReduce Configuration

a. Generate Data

Script `generate.py` generates `csv` file contains random names, studentsID and grades.

- Run: `$ python3 generate.py <number of lines>`
 - e.g. `python3 generate.py 100`
- Python module: `names`, `random`
- Input: None
- Output: create directory `data/` and generate `grades_100.csv`
 - Format: `<name>,<studentID>,<grade>`
 - e.g. `Michael Huang,0123456789,90`

Code for `generate.py`

```
1  #!/usr/bin/python3
2
3  import os
4  import sys
5  import random
6  import names
7
8  DATA_NUMBER = 300
9  if (len(sys.argv) < 2):
10     print("Usage: generate.py <number of lines>")
11     exit(1)
12  LINE_NUMBER = int(sys.argv[1])
13  BASE_DIR = "data/"
14
15  id = set()
16  firstnames = set()
17  lastnames = set()
18
19
20  def generate_raw():
21
22     for _ in range(DATA_NUMBER):
23         id.add(random.randint(1000000000, 9999999999))
24
25     for _ in range(DATA_NUMBER):
26         firstnames.add(names.get_first_name())
27
28     for _ in range(DATA_NUMBER):
```

```
29     lastnames.add(names.get_last_name())
30
31     if (not os.path.exists(BASE_DIR)):
32         os.makedirs(BASE_DIR)
33
34
35 def generate_csv():
36     first = list(firstnames)
37     last = list(lastnames)
38     ID = list(id)
39     with open(os.path.join(BASE_DIR, "grades_{}.csv".format(LINE_NUMBER
40         )), 'w') as f:
41         for i in range(LINE_NUMBER):
42             rand = random.randint(0, min(len(first), len(last), len(ID)
43                 )-1)
44             grade = random.randint(0, 100)
45             f.write("{} {},{}\n".format(
46                 first[rand], last[rand], ID[rand], grade))
47
48 if "__main__" == __name__:
49     generate_raw()
50     generate_csv()
```

b. Mapper

Mapper reads `stdin` with name, studentID & grade separated by newline, and returns the tab-separated pair: `studentID<TAB>grade`

- Run: `$./mapper.sh`
- Input: `stdin` (e.g `Michael Huang,0123456789,100`)
- Output: `stdout` (e.g `0123456789<TAB>100`)
- Test: Use input redirection to read from file `grades.csv`

Code for mapper.sh

```
1 #!/bin/bash
2 # Reads STDIN with name, studentID and grades separated by newline
3 # Returns the tab-separated pair: studentID<TAB>grade
4 # Input:
5 #     STDIN: Michael Huang,0123456789,100
6 # Output:
7 #     STDOUT: 0123456789<TAB>100
8
9 awk -F, '{print $2"\t"$3}'
```

c. Reducer

Reducer reads tab-separated pairs from the standard input, each of which is composed of a studentID and a grade, and returns the max grade for each student on the standard output.

- Run: `$./reducer.sh`
- Input: `stdin` (e.g. `0123456789<TAB>80 ... 0123456789<TAB>100`)
- Output: `stdout` (e.g. `0123456789 100`)

Code for `reducer.py`

```
1  #!/usr/bin/python3
2  # coding:utf-8
3
4  # Reads pairs from the standard input. Each tab-separated pair is
   composed of a studentID and a list of grades
5  # Returns the max grade for each student on the standard output.
6  # Input:
7  #     StudentID<TAB>Grade1
8  #     StudentID<TAB>Grade2
9  #     ...
10 # Output:
11 #     StudentID Max(Grade1, Grade2, ...)
12
13
14 import sys
15
16
17 def reduce():
18
19     roster = {}
20
21     # build roster
22     line = sys.stdin.readline()
23     while line:
24         entry = line.split()
25         ID = entry[0]
26         grade = int(entry[1])
27         roster.setdefault(ID, []).append(grade)
28         line = sys.stdin.readline()
29
30     # find max grade
31     for ID in sorted(roster.keys()):
32         print("{} {}".format(ID, max(roster[ID])))
33
34
35 if "__main__" == __name__:
36     reduce()
```


d. Single Task

Single task cascades mapper and reducer with pipe.

- Run: `$./mapper.sh < grades_100.csv | ./reducer`
- Benchmark: use `time` command to calculate time elapsed

e. Hadoop Cluster

1) HDFS

```
1 hdfs dfs -ls <dir_in_hdfs>
2 hdfs dfs -mkdir <dir_in_hdfs>
3 hdfs dfs -put <file_in_your_system> <dir_in_hdfs>
4 hdfs dfs -get <file_in_hdfs>
5 hdfs dfs -rm -r -f output
```

You can check via Utilities->Browse file system in `localhost:9870` to check the directory and files on the hdfs.

2) Streaming

In your hadoop home directory, run streaming with package: `share/hadoop/tools/lib/hadoop-streaming-3.3.2.jar` via the following command after you have created directory in hdfs for `<DFS_INPUT_DIR>` and `<DFS_OUTPUT_DIR>`

```
1 hadoop jar <HADOOP_HOME>/share/hadoop/tools/lib/hadoop-streaming-3.3.2.jar
  -input <DFS_INPUT_DIR> -output <DFS_OUTPUT_DIR> -mapper <MAPPER>
  -reducer <REDUCER> -file <LOCAL_MAPPER_DIR> -file <LOCAL_REDUCER_DIR>
```

Notes:

- and can be local files, while and is in hdfs
- `<DFS_OUTPUT_DIR>` needs to be emptied everytime re-running the MapReduce task

Error Handling:

- Check `<HADOOP_HOME>/logs` for detailed logs
- `WARN org.apache.hadoop.streaming.PipeMapRed: java.io.IOException`
 - Check if `#!/usr/bin/env python` is included if you are using python
 - Check if shell scripts are granted permission to execute
 - Check if shell scripts handle the exception correctly

f. Benchmark

1) Run Hadoop MapReduce Tasks

- Path: `root@hadoop-master:/home/s/lab2_benchmark`
- Command

```
1 $ chmod +x ./benchmark.sh
2 $ ./benchmark.sh
```

- Input: put `grades_#.csv` into `input/`
- Output: `/log/time.log` containing task time in seconds
- Effect:
 1. copy the `csv` files in `input/` to `hdfs`
 2. For each file in `input/`:
 - Run MapReduce tasks
 - Calculate time used in seconds
 3. Generate a log file `time.log` in `log/`

2) Scatter and fitting Plot

- Run: `$ python3 plot.py`
- Output: plots saved to `img/`

Code for `benchmark.sh`

```
1 #!/bin/bash
2 # run this script on root@hadoop-master:/home/s/lab2_benchmark
3
4 mkdir -p output
5 mkdir -p log
6
7 # delete hdfs input/lab2
8 hdfs dfs -rm -r -f input/lab2
9
10 # copy input to hdfs
11 mv input/ lab2/
12 hdfs dfs -put lab2/ input/
13 mv lab2/ input/
14
15 rm -f log/time.log
```

```
16
17 for ((NUM=1000;NUM<=1000000000;))
18 do
19     hdfs dfs -rm -r -f output/
20     cd /home/s/hadoop
21
22     start=$SECONDS
23
24     hadoop jar share/hadoop/tools/lib/hadoop-streaming-3.3.2.jar -input
        input/lab2/grades_$NUM.csv -output output -mapper /src/mapper.
        sh -reducer "python reducer.py" -file /src/mapper.sh -file /src
        /reducer.py
25
26     end=$SECONDS
27
28     cd /home/s/lab2_benchmark
29     hdfs dfs -get output/part-00000
30     mv part-00000 output/$NUM.out
31
32     duration=$((end - $start))
33     echo $NUM: $duration >> log/time.log
34
35     ((NUM=$NUM*10))
36 done
```

Code for plot.py

```
1 import os
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 BASE_DIR = "img/"
6
7 START_EXPONENT = 3
8 STOP_EXPONENT = 8
9
10 ##### Single #####
11 # Total Time v.s. Number of Students
12 student_num = [10**i for i in range(START_EXPONENT, STOP_EXPONENT+1)]
13 total_time = [4.088, 4.826, 5.189, 8.904, 55.917, 534]
14
15 coeffs = np.polyfit(np.log(student_num), np.log(total_time), deg=2)
16 poly = np.poly1d(coeffs)
17
18 x = np.logspace(START_EXPONENT, 1.01*STOP_EXPONENT, 100)
19 y = np.exp(poly(np.log(x)))
20
21 plt.scatter(student_num, total_time)
22 plt.plot(x, y, 'orange')
```

```
23
24 plt.xscale('log')
25 plt.xlabel('Number of Students')
26 plt.ylabel('Total Time (s)')
27 plt.title('Total Time v.s. Number of Students')
28
29 plt.savefig(os.path.join(BASE_DIR, 'single.png'))
30 plt.clf()
31
32 # File Size v.s. Number of Students
33 file_size = [29, 287, 2.8*1024, 28.7*1024, 286.9*1024, 2.87*1024*1024]
34
35 coeffs = np.polyfit(np.log(student_num), np.log(file_size), deg=2)
36 poly = np.poly1d(coeffs)
37 y = np.exp(poly(np.log(x)))
38
39 plt.scatter(student_num, file_size)
40 plt.plot(x, y, 'orange')
41
42 plt.xscale('log')
43 plt.xlabel('Number of Students')
44 plt.ylabel('File Size (KB)')
45 plt.title('File Size v.s. Number of Students')
46
47 plt.savefig(os.path.join(BASE_DIR, 'file.png'))
48 plt.clf()
49
50 ##### Cluster #####
51 student_num = [10**i for i in range(START_EXPONENT, STOP_EXPONENT+1)]
52
53 total_time_1 = [25, 21, 21, 24, 47, 107]
54 total_time_2 = [21, 20, 22, 24, 40, 96]
55 total_time_3 = [22, 20, 23, 22, 45, 95]
56
57 total_time = []
58 for i in range(len(total_time_1)):
59     total_time.append((total_time_1[i] + total_time_2[i] + total_time_3
60                        [i])/3)
61
62 coeffs = np.polyfit(np.log(student_num), np.log(total_time), deg=3)
63 poly = np.poly1d(coeffs)
64
65 x = np.logspace(START_EXPONENT, 1.01*STOP_EXPONENT, 100)
66 y = np.exp(poly(np.log(x)))
67
68 # Total Time v.s. Number of Students
69 plt.scatter(student_num, total_time)
70 plt.plot(x, y, 'orange')
71
72 plt.xscale('log')
73 plt.xlabel('Number of Students')
```

```
73 plt.ylabel('Total Time (s)')
74 plt.title('Total Time v.s. Number of Students')
75
76 plt.savefig(os.path.join(BASE_DIR, 'cluster.png'))
77 plt.clf()
78
79 ##### Both #####
80 student_num = [10**i for i in range(START_EXPONENT, STOP_EXPONENT+1)]
81
82 total_time_1 = [25, 21, 21, 24, 47, 107]
83 total_time_2 = [21, 20, 22, 24, 40, 96]
84 total_time_3 = [22, 20, 23, 22, 45, 95]
85
86 total_time_single = [4.088, 4.826, 5.189, 8.904, 55.917, 534]
87
88 total_time_cluster = []
89 for i in range(len(total_time_1)):
90     total_time_cluster.append(
91         (total_time_1[i] + total_time_2[i] + total_time_3[i])/3)
92
93 coeffs_cluster = np.polyfit(
94     np.log(student_num), np.log(total_time_cluster), deg=3)
95 coeffs_single = np.polyfit(
96     np.log(student_num), np.log(total_time_single), deg=3)
97 poly_cluster = np.poly1d(coeffs_cluster)
98 poly_single = np.poly1d(coeffs_single)
99
100
101 x = np.logspace(START_EXPONENT, 1.01*STOP_EXPONENT, 100)
102 y_cluster = np.exp(poly_cluster(np.log(x)))
103 y_single = np.exp(poly_single(np.log(x)))
104
105 # Total Time v.s. Number of Students
106 plt.plot(x, y_single)
107 plt.scatter(student_num, total_time_single)
108 plt.plot(x, y_cluster)
109 plt.scatter(student_num, total_time_cluster)
110
111 plt.ylim(-20, 550)
112 plt.xscale('log')
113 plt.xlabel('Number of Students')
114 plt.ylabel('Total Time (s)')
115 plt.title('Total Time v.s. Number of Students')
116
117 plt.savefig(os.path.join(BASE_DIR, 'benchmark.png'))
118 plt.clf()
```