
ECE4721J - Lab 2 Report

Methods and Tools for Big Data

Yiding Chang, Yifan Shen, Kexuan Huang, Qinhang Wu

May 30, 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

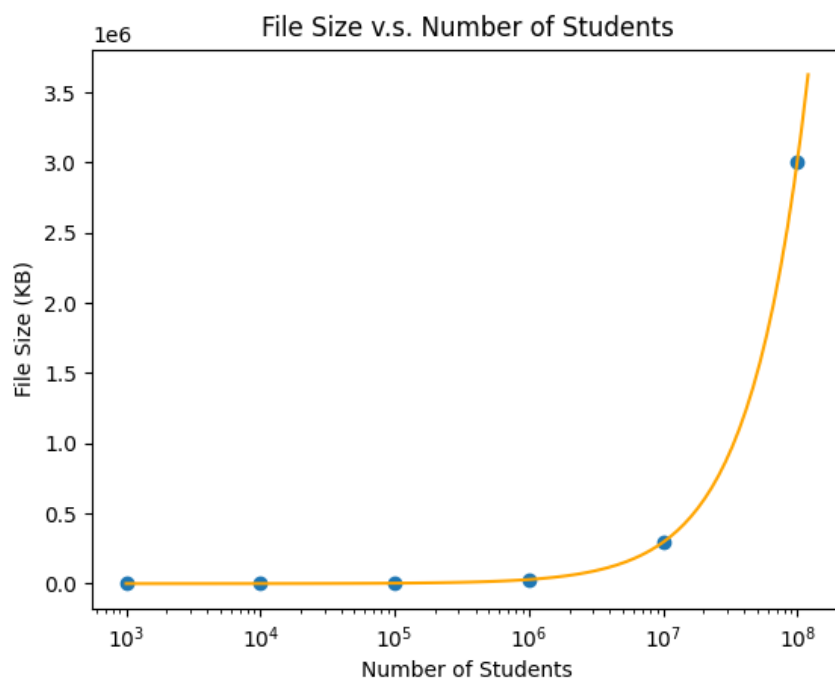


Figure 1: File Generation

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 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

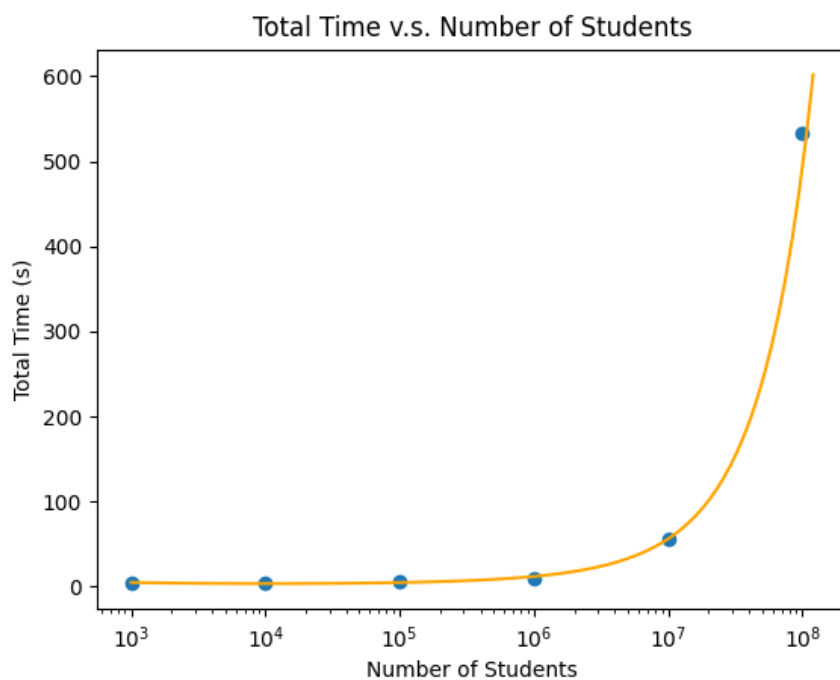


Figure 2: Single Performance

3. Performance on the group cluster

4. Code

a. Generate Raw Data

- Run: `python generate.py`
- Python module: `names, random`
- Input: None
- Output: 3 text files in `data/`
 - `firstnames.txt`
 - `lastnames.txt`
 - `id.txt`

Code for `generate.py`

```
1 import os
2 import random
3 import names
4
5 DATA_NUMBER = 1000
6 ENTRY_NUMBER = 10000
7
8 BASE_DIR = "data/"
9
10 id = set()
11 firstnames = set()
12 lastnames = set()
13
14
15 def generate_raw():
16
17     for _ in range(DATA_NUMBER):
18         id.add(random.randint(10000000000, 9999999999))
19
20     for _ in range(DATA_NUMBER):
21         firstnames.add(names.get_first_name())
22
23     for _ in range(DATA_NUMBER):
24         lastnames.add(names.get_last_name())
25
26     if (not os.path.exists(BASE_DIR)):
27         os.makedirs(BASE_DIR)
28
```

```
29     with open(os.path.join(BASE_DIR, 'id.txt'), 'w') as f:
30         for i in id:
31             f.write(str(i) + '\n')
32
33     with open(os.path.join(BASE_DIR, 'firstnames.txt'), 'w') as f:
34         for i in firstnames:
35             f.write(i + '\n')
36
37     with open(os.path.join(BASE_DIR, 'lastnames.txt'), 'w') as f:
38         for i in lastnames:
39             f.write(i + '\n')
40
41
42     def generate_csv():
43         first = list(firstnames)
44         last = list(lastnames)
45         ID = list(id)
46         with open("grades.csv", 'w') as f:
47             for i in range(ENTRY_NUMBER):
48                 rand = random.randint(0, min(len(first), len(last), len(ID))
49                     )-1)
50                 grade = random.randint(0, 100)
51                 f.write("{} {}{}\n".format(
52                     first[rand], last[rand], ID[rand], grade))
53
54     if "__main__" == __name__:
55         generate_raw()
56         generate_csv()
```

b. Generate grades.csv

- run: `./grading.sh`

c. Mapper

- run: `./mapper.sh < grades.csv`
- Usage: Reads `stdin` with name, studentID and grades separated by newline. Returns the tab-separated pair: `studentID<TAB>grade`
- Input: `stdin` (e.g `Michael Huang,0123456789,100`)
- Output: `stdout` (e.g `0123456789<TAB>10`)
- 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 # Output:
5 #     STDOUT: 0123456789<TAB>100
6
7 awk -F "," '{printf "%s\t%s\n", $2, $3}'
```

d. Reducer

- run: `./reducer.sh < data/reducer.in`
- Usage: Reads pairs from the standard input. Each tab-separated pair is composed of a studentID and a list of grades. Returns the max grade for each student on the standard output.
- Input: `stdin` (e.g. 0123456789<TAB>86 100 92)
- Output: A single number as the max grade (e.g. 100)
- Test: Use input redirection to read from file `data/reducer.in`

Code for reducer.py

```
1 #!/usr/bin/python
2 #coding:utf-8
3
4 import sys
5
6 def reduce():
7
8     roster = {}
9
10    # build roster
11    line = sys.stdin.readline()
12    while line:
13        entry = line.split()
14        ID = entry[0]
15        grade = int(entry[1])
16        roster.setdefault(ID, []).append(grade)
17        line = sys.stdin.readline()
18
19    # find max grade
20    for ID in sorted(roster.keys()):
21        print("{} {}".format(ID, max(roster[ID])))
22
23
24 if "__main__" == __name__:
25     reduce()
```

e. Hadoop Pseudo distribution

Run the following command for Streaming and Mapreduce,

```
1 time hadoop jar share/hadoop/tools/lib/hadoop-streaming-3.3.2.jar -  
  input input/lab2/grades.csv -output output -mapper mapper.sh -  
  reducer reducer.sh -file ~/Downloads/lab2/mapper.sh -file ~/  
  Downloads/lab2/reducer.sh
```

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 need to empty the output directory  
  everytime you want to rerun the code, you will see a message if rm  
  is successful
```

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

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 `dir_in_hdfs` for `inputdir` and `outputdir`.

```
1 hadoop jar share/hadoop/tools/lib/hadoop-streaming-3.3.2.jar -input  
  inputdir -output outputdir -mapper mapper.sh -reducer reducer.sh -  
  file localdirectorymapper.sh -file localdirectoryreducer.sh
```

Notes:

- `reducer.sh` and `mapper.sh` can be local file, while inputfile is in the HDFS
- Error handling
 - `WARN org.apache.hadoop.streaming.PipeMapRed: java.io.IOException`
 - * Check if `#!/usr/bin/env python` is included if you are using Python
 - * Check if exception handling is correct in the shell scripts