



Programming Assignment 1: A simple MapReduce-like Compute Framework

CSCI 5105: Introduction to Distributed Systems

Spring 2019

Instructor: Abhishek Chandra

March 12, 2019

Team:

Harshit Jain (jain0149@umn.edu)

Vaybhav Shaw (shaw0162@umn.edu)

Table of Contents

1. Overview	3
2. Components Implemented	3
2.1 Client	
2.2 Server	
2.3 WorkerNode	
2.4 CalculateSentimentService	
2.5 ClientServerHandler	
2.6 SentimentService	
2.7 SentimentHandler	
2.8 MapTest	
2.9 SortTest	
3. Workflow	5
4. Running the code	6
5. Testing	7
6. Performance Evaluation	14
7. Limitations	15

1. Overview

The goal of this project is to implement a simple MapReduce-like Compute Framework using Thrift and Java. The project helped us to understand RPC communications, Map Reduce operation and other distributed systems fundamentals.

The stack used for analysis:

1. **Thrift** - To build distributed services.
2. **Java** - For coding
3. **CSE Lab Machines** - To host server and worker nodes

2. Components Implemented

In this section, we present a brief overview of the various components used.

2.1 Client (Client.java)

The client reads all the files given in the input directory and pass it to the server for map reduce tasks. It reads the fileNames as Strings into an arrayList which is then sent to Server.

2.2 Server (Server.java)

The server receives all the filenames from the Client as an ArrayList and reads it. It then sends it to the Compute Nodes for Map-Reduce tasks.

2.3 Worker Node (WorkerNode.java)

Worker Nodes does the Map and Sort tasks as specified. Each Worker node runs on a different machine.

2.4 Calculate Sentiment (CalculateSentiment.java)

Calculate Sentiment service is the code that is generated using Apache Thrift to build RPC between the client and server. This file is generated from the CalculateSentiment.thrift file.

2.5 Client Server Handler (ClientServerHandler.java)

The client server handler receives all the strings(FileNames) sent by the Client. From there, based on the hostname and ports provided in the config file, the handler establishes the connection between the server and the Worker Nodes. It uses all the strings to generate individual threads and divide them among worker nodes. It is done on the basis of scheduling policy as mentioned in the config file. The Handler also takes care to run Sort task after the map task is completed for all the threads or Strings(Filenames).

2.6 Sentiment Service (SentimentService.java)

Sentiment service is the code that is generated using Apache Thrift to build RPC between the Server and the Worker Node. This file is generated from the SentimentService.thrift file.

2.7 Sentiment Handler (SentimentHandler.java)

Sentiment Handler takes in the thread and based on scheduling policy mentioned in the config file is handled by either Random or Load Balancing policy. The Sentiment Handler calls MapTest and SortTest which does the Map task and sort task respectively.

2.8 MapTest (MapTest.java)

The MapTest class takes in the fileName as input and computes the sentiment score using the positives and negatives as mentioned. It uses Regular expression to segregate the words from the file and classify them according to the negatives and the positives. The resulting score along with

the file name is written into the intermediate directory which is in the same directory as input and output directory.

2.9 SortTest (SortTest.java)

The SortTest class uses all the files in the intermediate directory and created a single output file containing the file name and the sentiment score of each file name.

3. Workflow

We created two thrift files and generated two services for communication between client and server and server and worker nodes respectively. We then created the handlers for each server and worker node.

Detailed Workflow:

1. The client extracts all the filenames from the input directory and passes it to the Server.
2. The Server runs the main thread. Using ClientServerhandler we generate the threads corresponding to each string and randomly choose a worker node from the given worker nodes in config file and assign the thread to it for the map task.
3. Each thread calls the services method of SentimentHandler.java and based on the scheduling policy defined in the config file, the thread is serviced.
4. The services method calls the Map function which in turn calls the map method in MapTest class which calculates the sentiment score based on the positives and negatives.
5. Once the map is computed for each file, the sort function is called.
6. The Sort function does the sorting based on Sentiment Score using Priority Queue and writes the final output to the output directory.

4. Running the Code

We created a config file which specifies the following:

1. First line mentions the scheduling policy. (0-Random 1-Load Balancing)
2. Second line mentions the port numbers. (9106-Client-Server, 9107-Server-WorkerNode) which is separated by a single space
3. From third line on, each line has a description of the hostnames and load value of each WorkerNode, separated by a single space.
(e.g *cse1-kh4250-06.cselabs.umn.edu 0.2*)

Note: All the values are separated by a single space.

To run the code, we need to follow the following steps:

(Everything needs to be run from the directory in which the files are present)

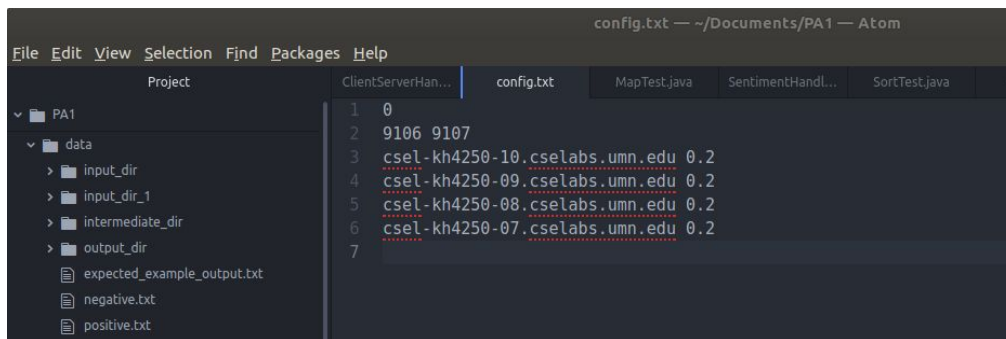
1. We first compile the project using the following syntax :
javac -cp ".:usr/local/Thrift/*" *.java -d .
2. Then we ssh into the WorkerNodes mention in the config file by ssh-ing into the CSE lab machines. For example, we ssh on 4 machines for running 4 individual worker nodes.
3. We then run the WorkerNodes individually using the following command
java -cp ".:usr/local/Thrift/*" WorkerNode
4. We then run the Server on the local node using:
java -cp ".:usr/local/Thrift/*" Server
5. Next, we then run the Client on the local node to run the entire Map-Reduce functionality:
java -cp ".:usr/local/Thrift/*" Client

5. Testing

Scenario 1 :

- Using the provided test files comedies, histories, poems, tragedies
- Using Random Scheduling
- Using 4 WorkerNodes each with load probability 0.2
 - csel-kh4250-10.cselabs.umn.edu 0.2
 - csel-kh4250-09.cselabs.umn.edu 0.2
 - csel-kh4250-08.cselabs.umn.edu 0.2
 - csel-kh4250-07.cselabs.umn.edu 0.2

1. This is how the config.txt looks



```
config.txt — ~/Documents/PA1 — Atom
File Edit View Selection Find Packages Help
Project
  PA1
  data
  input_dir
  input_dir_1
  intermediate_dir
  output_dir
  expected_example_output.txt
  negative.txt
  positive.txt
ClientServerHan...
config.txt
MapTest.java
SentimentHandl...
SortTest.java
C
1 0
2 9106 9107
3 csel-kh4250-10.cselabs.umn.edu 0.2
4 csel-kh4250-09.cselabs.umn.edu 0.2
5 csel-kh4250-08.cselabs.umn.edu 0.2
6 csel-kh4250-07.cselabs.umn.edu 0.2
7
```

2. Output on the Server Node

Runnable Servers:

csel-kh4250-10.cselabs.umn.edu 0.2

csel-kh4250-09.cselabs.umn.edu 0.2

csel-kh4250-08.cselabs.umn.edu 0.2

csel-kh4250-07.cselabs.umn.edu 0.2

Starting Map Task

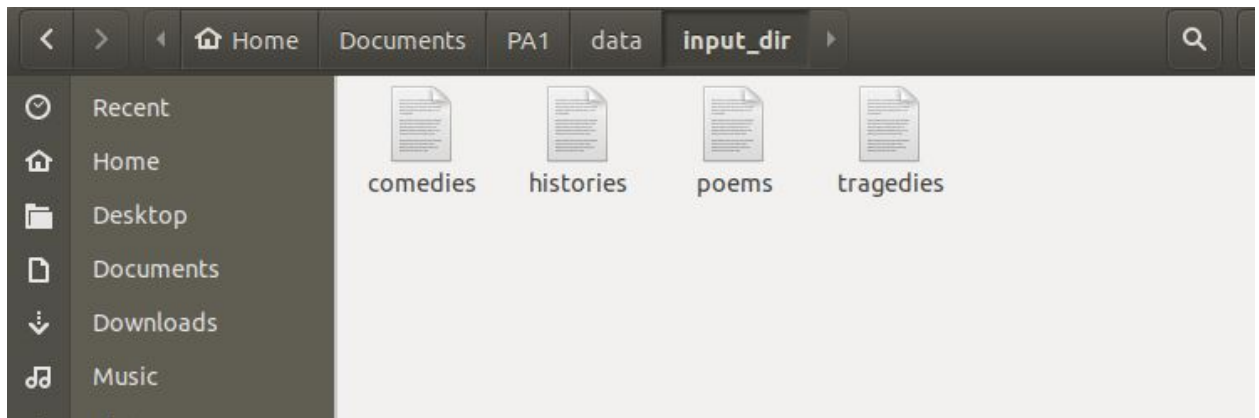
Total Run Time of Map Task = 3.177 secs

Starting Sort Task:

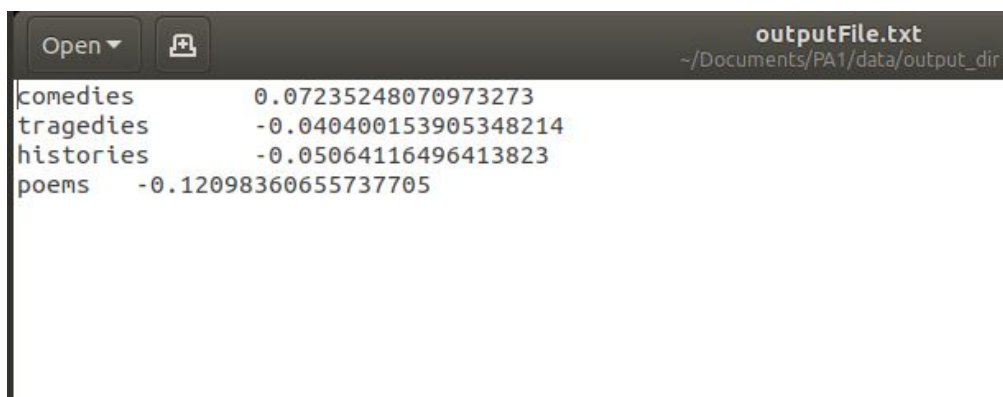
Sort Call End

Total Run Time of Sort Task = 0.068 secs

3. Intermediate Files Generated:



4. Final Output in data/output_dir/outputFile.txt



****Note:** *Output is accurate to two places of decimals*

Scenario 2 :

- Using the 500 test files provided to us
- Using Load Balancing Scheduling with Rejections
- Using 4 WorkerNodes each with load probability
 - *cse-kh4250-10.cselabs.umn.edu* 0.8
 - *cse-kh4250-09.cselabs.umn.edu* 0.6
 - *cse-kh4250-08.cselabs.umn.edu* 0.5
 - *cse-kh4250-07.cselabs.umn.edu* 0.2

1. This is our config.txt for this example

```
1
9106 9107
cse-kh4250-10.cselabs.umn.edu 0.8
cse-kh4250-09.cselabs.umn.edu 0.6
cse-kh4250-08.cselabs.umn.edu 0.5
cse-kh4250-07.cselabs.umn.edu 0.2
```

2. Output on the Server Node

```
Runnable Servers:
cse-kh4250-10.cselabs.umn.edu 0.8
cse-kh4250-09.cselabs.umn.edu 0.6
cse-kh4250-08.cselabs.umn.edu 0.5
cse-kh4250-07.cselabs.umn.edu 0.2
Starting Map Task
Total Run Time of Map Task = 5.722 secs
Starting Sort Task:
Sort Call End
Total Run Time of Sort Task = 0.977 secs
```

3. Output on one of the WorkerNode with Load Probability 0.2

WorkerNode Output shows the task number that has been executed along with the task execution time and the average task execution time until then.

```
Rejecting File : 12498.txt with load probability : 0.2
Rejecting File : 12434.txt with load probability : 0.2
Rejecting File : 12360.txt with load probability : 0.2
Rejecting File : 12747.txt with load probability : 0.2
Rejecting File : 12651.txt with load probability : 0.2
```

.

.
 .
 .
 File: 12457.txt ,task#:215, ran for: 329 ms, current average time = 1731 ms
 File: 12442.txt ,task#:216, ran for: 325 ms, current average time = 1725 ms
 File: 12883.txt ,task#:217, ran for: 189 ms, current average time = 1718 ms
 File: 12599.txt ,task#:218, ran for: 38 ms, current average time = 1710 ms
 File: 12737.txt ,task#:219, ran for: 54 ms, current average time = 1702 ms
 File: 12772.txt ,task#:220, ran for: 98 ms, current average time = 1695 ms
 File: 12639.txt ,task#:221, ran for: 260 ms, current average time = 1689 ms

4. Final Output in data/output_dir/outputFile.txt

12642.txt	0.5434530706836617
12626.txt	0.4893410852713178
12648.txt	0.4666666666666667
12549.txt	0.43205013428827216
12631.txt	0.414911781445646
12701.txt	0.4090909090909091
12657.txt	0.4088145896656535
.	
.	
.	
12660.txt	-0.5202821869488536
12756.txt	-0.5205992509363296
12613.txt	-0.529276693455798
12698.txt	-0.5884177869700103
12822.txt	-0.6073298429319371
12636.txt	-0.8584660985550204

Scenario 3 :

- Using the 500 test files provided to us
- Using Random Scheduling **without** Rejections
- Using 4 WorkerNodes each with load probability
 - *cse1-kh4250-10.cselabs.umn.edu* 0.8
 - *cse1-kh4250-09.cselabs.umn.edu* 0.6
 - *cse1-kh4250-08.cselabs.umn.edu* 0.5
 - *cse1-kh4250-07.cselabs.umn.edu* 0.2

1. This is our config.txt for this example

```
0
9106 9107
cse1-kh4250-10.cselabs.umn.edu 0.8
cse1-kh4250-09.cselabs.umn.edu 0.6
cse1-kh4250-08.cselabs.umn.edu 0.5
cse1-kh4250-07.cselabs.umn.edu 0.2
```

2. Output on the Server Node

```
cse1-kh4250-10.cselabs.umn.edu 0.8
cse1-kh4250-09.cselabs.umn.edu 0.6
cse1-kh4250-08.cselabs.umn.edu 0.5
cse1-kh4250-07.cselabs.umn.edu 0.2
Starting Map Task
Total Run Time of Map Task = 6.83 secs
Starting Sort Task:
Sort Call End
Total Run Time of Sort Task = 0.454 secs
```

3. Output on one of the WorkerNode with Load Probability 0.5

WorkerNode Output shows the task number that has been executed along with the task execution time and the average task execution time until then.

```
.
.
.
```

File: 12792.txt ,task#:69, ran for: 1068 ms, current average time = 1322 ms

File: 12729.txt ,task#:70, ran for: 1731 ms, current average time = 1328 ms

File: 12368.txt ,task#:71, ran for: 1216 ms, current average time = 1326 ms

File: 12607.txt ,task#:72, ran for: 102 ms, current average time = 1309 ms
File: 12401.txt ,task#:73, ran for: 1819 ms, current average time = 1316 ms
File: 12743.txt ,task#:74, ran for: 736 ms, current average time = 1308 ms
File: 12860.txt ,task#:75, ran for: 22 ms, current average time = 1291 ms
File: 12863.txt ,task#:76, ran for: 1790 ms, current average time = 1297 ms
File: 12807.txt ,task#:77, ran for: 1788 ms, current average time = 1304 ms
.
.
.

4. Final Output in data/output_dir/outputFile.txt

12642.txt	0.5434530706836617
12626.txt	0.4893410852713178
12648.txt	0.4666666666666667
12549.txt	0.43205013428827216
12631.txt	0.414911781445646
12701.txt	0.4090909090909091
12657.txt	0.4088145896656535
.	
.	
.	
12660.txt	-0.5202821869488536
12756.txt	-0.5205992509363296
12613.txt	-0.529276693455798
12698.txt	-0.5884177869700103
12822.txt	-0.6073298429319371
12636.txt	-0.8584660985550204

Scenario 4 (Negative Test Case) :

We use the following config file

```
4
9106 9107
cse-kh4250-10.cselabs.umn.edu 0.1
cse-kh4250-09.cselabs.umn.edu 0.5
cse-kh4250-08.cselabs.umn.edu 0.2
cse-kh4250-07.cselabs.umn.edu 0.9
```

Output on Server:

```
^Cshaw0162@cse-kh4250-03:/home/shaw0162/Documents/PA1 $ java -cp ".:usr/local/
rift/*" Server
SLF4J: The requested version 1.5.8 by your slf4j binding is not compatible with
[1.6, 1.7]
SLF4J: See http://www.slf4j.org/codes.html#version_mismatch for further details.
Runnable Servers:
Invalid Scheduling Policy Input in config.txt
```

First line of input is scheduling policy and should be either 0 (Random) or 1 (Load Balanced). Else, it will end with a message gracefully.

Scenario 5 (Negative Test Case) :

We use the following config file

```
1
9106 abc
cse-kh4250-10.cselabs.umn.edu 0.1
cse-kh4250-09.cselabs.umn.edu 0.5
cse-kh4250-08.cselabs.umn.edu 0.2
cse-kh4250-07.cselabs.umn.edu 0.9
```

Output on Server:

```
^Cshaw0162@cse-kh4250-03:/home/shaw0162/Documents/PA1 $ java -cp ".:usr/local/
rift/*" Server
SLF4J: The requested version 1.5.8 by your slf4j binding is not compatible with
[1.6, 1.7]
SLF4J: See http://www.slf4j.org/codes.html#version_mismatch for further details.
Runnable Servers:
Port Number should be a valid Integer
```

Second line of input is Port Numbers which should be valid integers. In case of incorrect input, it will end with a message gracefully.

Scenario 6 (Negative Test Case):

We use the following config file

1

9106 9107

cse1-kh4250-10.cselabs.umn.edu 4.1

cse1-kh4250-09.cselabs.umn.edu 0.5

cse1-kh4250-08.cselabs.umn.edu 0.2

cse1-kh4250-07.cselabs.umn.edu 0.9

Output on Server:

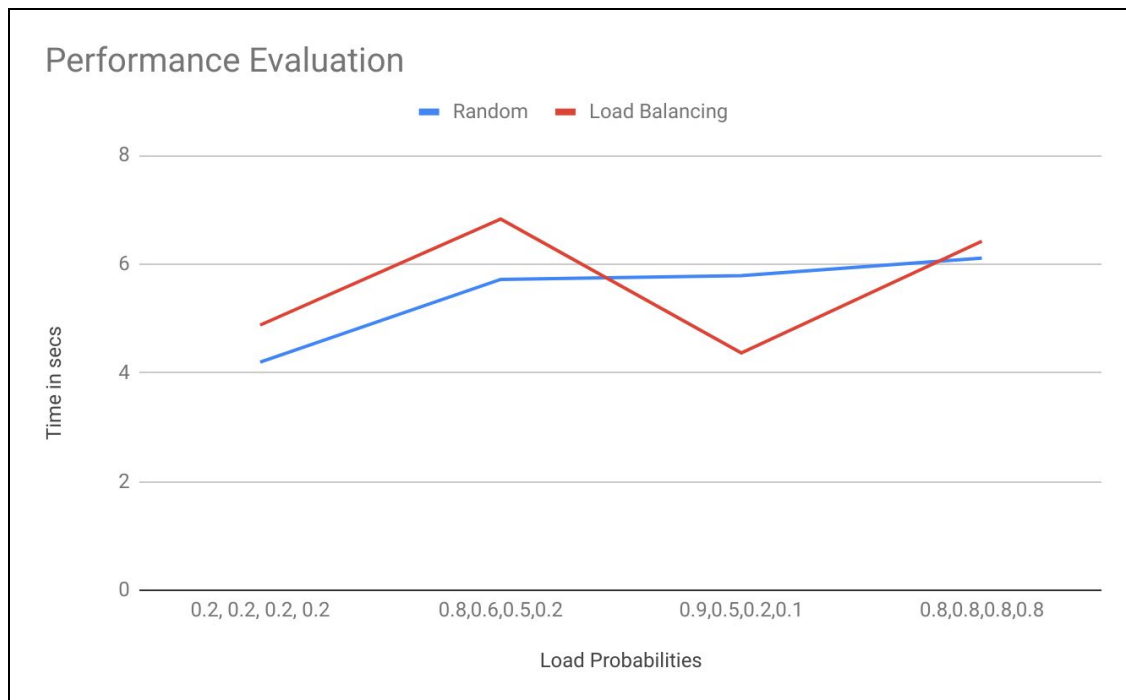
```
SLF4J: The requested version 1.5.8 by your slf4j binding is not compatible with [1.6, 1.7]
SLF4J: See http://www.slf4j.org/codes.html#version_mismatch for further details.
Runnable Servers:
Invalid Load Probability (should be between: 0.0 and 1.0)
```

Load Probability for each server should be between 0.0 and 1.0. In case of incorrect input, it will end with a message gracefully.

6. Performance Evaluation

We compared the performance of both the scheduling methods under various combinations of load probabilities and based on the results we plotted the chart below:

	0.2, 0.2, 0.2, 0.2	0.8,0.6,0.5,0.2	0.9,0.5,0.2,0.1	0.8,0.8,0.8,0.8
Random	4.196	5.72	5.787	6.111
Load Balancing	4.88	6.83	4.366	6.422



7.Limitations

1. The task numbers do not reset to 0 after each successive run. The WorkerNode retains the state of the task number.