Implementing a simple MapReduce-like compute framework

Shrivardhan S. Bharadwaj(5478571)

Himanshu Kharkwal(5488345)

University of Minnesota

INDEX

1. Purpose

To implement a simple MapReduce-like compute framework for implementing a sentiment analysis program. The framework(server) receives jobs from a client, split job into multiple tasks, and assign tasks to compute nodes which perform the computations. A user will send a job for sentiment analysis. This framework uses different scheduling policies e.g., Random and Load-balancing, to assign tasks like, mapping files to sentiment scores and sorting files according to their scores, to compute nodes.

2. Method

2.1 System Design

We implemented three components:

1. Client : The client sends a job to the server. The job includes filenames which store data to be analyzed. When the job is done, the client gets a result: an ordered list of filenames based on the sentiment score, and elapsed time for a job.
2. Server : The server receives the job submitted by the client. It splits the job into multiple tasks and assigns each task to a compute node. The server needs to prints the status of job e.g., the number of running and complete tasks for a job, etc. When the job is completed, the server prints the elapsed time to run it and returns a result to the client.
3. Compute Node : The compute nodes execute tasks (either map or sort) sent to them by the server. Each compute node runs on a different machine.

2.2 List of Files

1. hostnames.txt : User should provide the IP addresses of all the compute nodes which are being used.
2. MapReduceClient.java :
    I. Asks user to input directory which contains all the files to be computed.
    II. Creates logs of statistics of the tasks allocated.
    III. Returns final scores and filename to the client.
3. MapReduceHandler.java : receive jobs from a client, split job into multiple tasks, and assign tasks(map and sort) to the compute nodes provided by the user.
4. MapReduceComputeServer.java : Receives task allocated by the server and sends them to handler.
5. MapReduceComputeHandler.java :
    I. Checks load probability, accepts and rejects tasks accordingly.
    II. Compute sentimental scores for each file and prints time taken by the task.
    III. Returns the score and time taken by the task to the server.

6. Make : Compiles all the java files
7. Run : Starts the framework
8. MapReduceService.thrift
9. MapReduceComputeService.thrift
10. Sentiment files : Positive.txt and Negative.txt
11. Intermediate files are in ./intermediate_dir/ and output files are in ./output_dir/

Note: All intermediate files in ./intermediate_dir/ are deleted after execution. The output.txt and stat.txt in output_dir are deleted next time the code is run

2.3 Scheduling Policies

In this project, we are implementing 2 scheduling policies.

1. Random: The server will assign tasks to any compute nodes chosen randomly.
2. Load-Balancing: The server tries to assign tasks to any compute nodes. However, the compute node may reject accepting a task based on the 'load-probability'. If a compute node rejects a task, the server tries to assign the task to other compute nodes until the tasks is accepted. The scheduling policy is to be provided as a parameter when the server starts.

2.4 Performance Evaluation Metrics

1. For our evaluation we have used 4 compute nodes.
2. The performance of the system is being evaluated for different task scheduling policies:
    a. Random
    b. Load-balancing for different load probabilities:
        I. 0.2, 0.2, 0.2, 0.2
        II. 0.8, 0.8, 0.8, 0.8
        III. 0.1, 0.3, 0.2, 0.4
        IV. 0.1, 0.5, 0.2, 0.9

V. 0.8, 0.6, 0.7, 0.9

3. Data Collected for statistics:

I. Server:

<Host node ID : jobs received, jobs executed, total time, average time>

II. Client:

<elapsed time for entire job>

<filename, score>

2.5 Logs generated

1. Client:

<elapsed time for a job>

<filename, score>

2. Server:

<number of running tasks>

<number of tasks completed>

3. Compute Node:

<filename, time>

3. How to run

1. SSH to client and server(in our code both lie on the same machine) machines

   $ ssh <client>

   $ ssh <server>

2. Download and extract the tar to some location in client

   $ tar -xzf <tarfile>

3. Edit hostname.txt and add the names of all the machines(in seperate lines) which you want to use as compute nodes

   $ nano hostnames.txt

   > <hostname_1>

   > <hostname_2>

   > .

   > .

   > <hostname_n>

4. Execute Make file at client for compiling all thrift files and Java Code

   $ sh Make

5. Execute Run file at each machine to start the framework. Note: Start server and compute nodes before the client node.

   In server machine:

   $ sh Run MapReduceServer

   In compute node machines:

   $ sh Run MapReduceComputeServer

   This will open a menu and prompt the user for the following inputs:

   > Menu

   > 1. Enter Load Probability (default = 0 {random policy})

   > 2. Enter Sleep Time in seconds (default = 3s)

   > 3. To start server

   In client machine:

$ sh Run MapReduceClient

This will prompt the user for the following input:

> Enter the path of the input directory:

Note: The above steps are to be followed for NFS Distributed File System.

If you are using any other type of DFS then download and extract the tar at every machine.

Note: Hostnames can be changed while the server is running. There is no need to kill the server to scale the compute nodes.

4. Test cases

1. Case : Input folder is empty

   Output : Returns and prints “No files present. Add files to the input folder” and Menu starts again. The user at this point can edit hostnames.txt and continue with the current execution.

2. Case : hostnames.txt file is empty

   Output : Returns and prints “No hosts present. Edit hostnames.txt” and Menu starts again. The user at this point can edit hostnames.txt and continue with the current execution.

3. Case : Input file does not contain positive or negative words

   Output : Prints “File has no sentiments” and returns a 0 sentiment score for the file.

4. Case : Expected results for Random Scheduling

   Output :

```
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.
Menu:
1:Sort Directory
2.Quit
1
Enter Directory (Eg: ./input_dir/)
./example/
./example/poems : -0.118738085
./example/histories : -0.043182626
./example/tragedies : -0.04125664
./example/comedies : 0.06519989

Time taken for the entire job = 5055.0ms


Menu:
1:Sort Directory
2.Quit
```

5. Case : Random scheduling

   Output :

```
./input_dir/12676.txt : 0.18483412
./input_dir/12739.txt : 0.18518518
./input_dir/12366.txt : 0.18666667
./input_dir/12575.txt : 0.18814139
./input_dir/12538.txt : 0.18814433
./input_dir/12597.txt : 0.1884058
./input_dir/12737.txt : 0.18882681
./input_dir/12543.txt : 0.18907988
./input_dir/12606.txt : 0.18922089
./input_dir/12817.txt : 0.18923955
./input_dir/12553.txt : 0.18955731
./input_dir/12466.txt : 0.18965517
./input_dir/12752.txt : 0.1904762
./input_dir/12771.txt : 0.19227812
```

6. Case : Different Load Probabilities
    a. Load Probabilities : (0.2, 0.2, 0.2, 0.2)

        Output :

```
./input_dir/12458.txt : 0.25925925
./input_dir/12589.txt : 0.2596685
./input_dir/12769.txt : 0.2602115
./input_dir/12628.txt : 0.26603997
./input_dir/12429.txt : 0.2675906
./input_dir/12887.txt : 0.26818952
./input_dir/12469.txt : 0.27226463
./input_dir/12862.txt : 0.28019324
./input_dir/12529.txt : 0.28062677
./input_dir/12632.txt : 0.2810038
./input_dir/12897.txt : 0.28323698
./input_dir/12567.txt : 0.28584474
./input_dir/12386.txt : 0.28797603
./input_dir/12518.txt : 0.29036295
./input_dir/12854.txt : 0.29130295
./input_dir/12700.txt : 0.29238564
./input_dir/12426.txt : 0.29702455
```

b. Load Probabilities : (0.8, 0.8, 0.8, 0.8)

Output :

```
./input_dir/12594.txt : 0.31388438
./input_dir/12375.txt : 0.31709203
./input_dir/12492.txt : 0.3235639
./input_dir/12358.txt : 0.3372434
./input_dir/12824.txt : 0.33742332
./input_dir/12340.txt : 0.34024897
./input_dir/12585.txt : 0.34138486
./input_dir/12741.txt : 0.3428899
./input_dir/12359.txt : 0.34333333
./input_dir/12376.txt : 0.34820393
./input_dir/12379.txt : 0.3580247
```

c. Load Probabilities : (0.1, 0.3, 0.2, 0.4)

Output :

```
./input_dir/12897.txt : 0.28323698
./input_dir/12567.txt : 0.28584474
./input_dir/12518.txt : 0.29036295
./input_dir/12854.txt : 0.29130295
./input_dir/12700.txt : 0.29238564
./input_dir/12460.txt : 0.2974305
./input_dir/12693.txt : 0.301059
./input_dir/12899.txt : 0.30833334
./input_dir/12649.txt : 0.31351507
./input_dir/12594.txt : 0.31388438
./input_dir/12375.txt : 0.31709203
./input_dir/12492.txt : 0.3235639
./input_dir/12824.txt : 0.33742332
./input_dir/12585.txt : 0.34138486
./input_dir/12741.txt : 0.3428899
./input_dir/12359.txt : 0.34333333
./input_dir/12376.txt : 0.34820393
```

d. Load Probabilities : (0.1, 0.5, 0.9, 0.2)

Output :

```
./input_dir/12376.txt : 0.34820393
./input_dir/12671.txt : 0.35303777
./input_dir/12379.txt : 0.3580247
./input_dir/12856.txt : 0.35825968
./input_dir/12754.txt : 0.36092716
./input_dir/12867.txt : 0.36361957
./input_dir/12404.txt : 0.36514246
./input_dir/12857.txt : 0.36775818
./input_dir/12389.txt : 0.37014925
./input_dir/12365.txt : 0.37755102
./input_dir/12883.txt : 0.3807426
./input_dir/12896.txt : 0.38351595
./input_dir/12513.txt : 0.39534885
./input_dir/12701.txt : 0.4011976
./input_dir/12631.txt : 0.4150289
./input_dir/12657.txt : 0.4200078
./input_dir/12549.txt : 0.43188512
./input_dir/12648.txt : 0.46173254
./input_dir/12626.txt : 0.4919517
./input_dir/12642.txt : 0.54862845
./input_dir/12868.txt : 0.6614173
```

e. Load Probabilities : (0.8, 0.6, 0.9, 0.7)

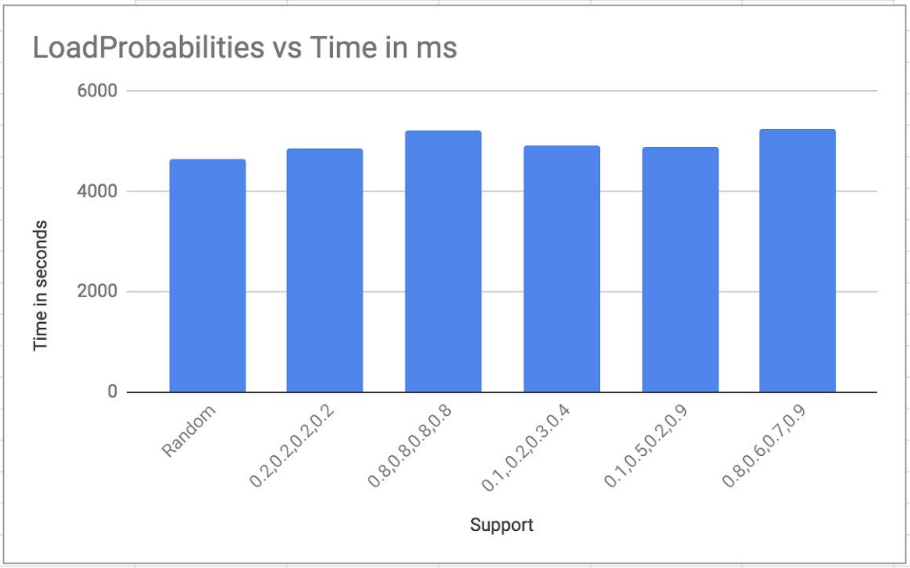
   Output :

```
./input_dir/12585.txt : 0.34138486
./input_dir/12741.txt : 0.3428899
./input_dir/12359.txt : 0.34333333
./input_dir/12376.txt : 0.34820393
./input_dir/12671.txt : 0.35303777
./input_dir/12754.txt : 0.36092716
./input_dir/12857.txt : 0.36775818
./input_dir/12389.txt : 0.37014925
./input_dir/12365.txt : 0.37755102
./input_dir/12883.txt : 0.3807426
./input_dir/12896.txt : 0.38351595
./input_dir/12513.txt : 0.39534885
./input_dir/12701.txt : 0.4011976
./input_dir/12657.txt : 0.4200078
./input_dir/12549.txt : 0.43188512
./input_dir/12648.txt : 0.46173254
```

5. Results and Analysis

1. Load probabilities vs time taken for a job

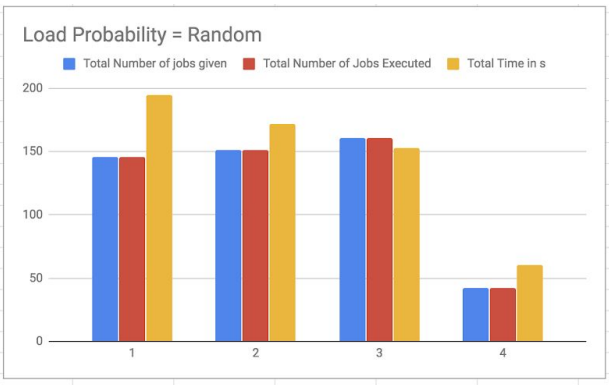
LoadProbabilities	Random	0.2,0.2,0.2,0.2	0.8,0.8,0.8,0.8	0.1,.0.2,0.3.0.4	0.1,0.5,0.2,0.9	0.8,0.6,0.7,0.9
Time Taken in ms	4640	4866	5217	4924	4886	5259



2. Histogram of Tasks ran by compute nodes based on different load probabilities

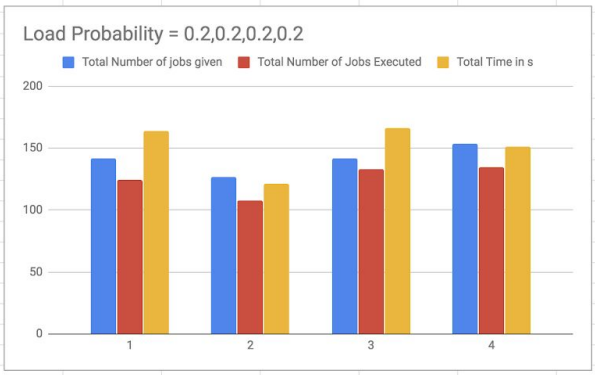
   Random

Compute Nodes	1	2	3	4
Total Number of jobs given	146	151	161	42
Total Number of Jobs Executed	146	151	161	42
Total Time in s	195	172	153	60



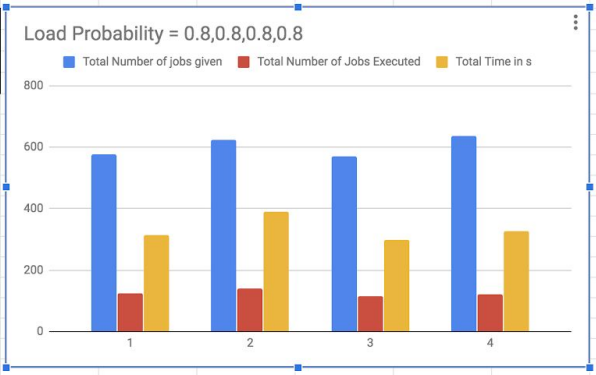
   Load Probability : (0.2, 0.2, 0.2, 0.2)

Compute Nodes	1	2	3	4
Total Number of jobs given	142	127	142	154
Total Number of Jobs Executed	124	108	133	135
Total Time in s	164	121	166	151



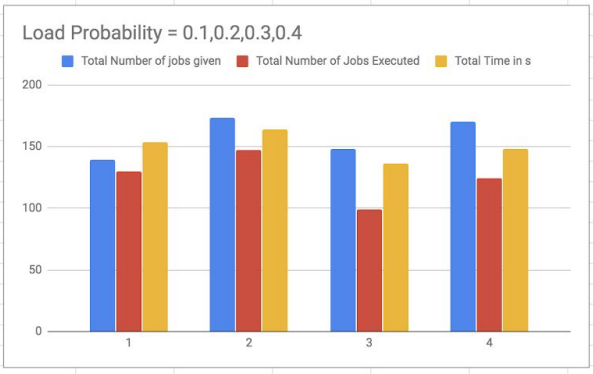
Load Probability : (0.8, 0.8, 0.8, 0.8)

Compute Nodes	1	2	3	4
Total Number of jobs given	577	623	569	637
Total Number of Jobs Executed	125	139	114	122
Total Time in s	314	389	300	326



Load Probability : (0.1, 0.2, 0.3, 0.4)

Compute Nodes	1	2	3	4
Total Number of jobs given	139	173	148	170
Total Number of Jobs Executed	130	147	99	124
Total Time in s	154	164	136	148



Load Probability : (0.1, 0.5, 0.9, 0.2)

Compute Nodes	1	2	3	4
Total Number of jobs given	217	251	225	214
Total Number of Jobs Executed	189	133	20	158
Total Time in s	313	267	62	299

Load Probability : (0.8, 0.6, 0.9, 0.7)

Compute Nodes	1	2	3	4
Total Number of jobs given	143	445	501	482
Total Number of Jobs Executed	109	173	76	142
Total Time in s	324	351	148	336

6. References

https://itss.d.umn.edu/services/file-storage

https://www.geeksforgeeks.org/java/

http://www-users.cselabs.umn.edu/classes/Spring-2019/csci5105/pas/pa1/pa1.pdf

https://thrift.apache.org/tutorial/java

https://ai.google/research/pubs/pub62

https://www.cloudera.com/documentation/other/tutorial/CDH5/topics/ht_example_4_sentiment_analysis.html