## Programming Assignment 1: A simple MapReduce-like compute framework

Yuanli Wang wang8662 Ruoyan Kong kong0135

## 1 Design document

### 1.1 System overview

We implemented a MapReduce-like compute framework based on Thrift to do sentiment analysis task. The whole system is composed of 1 server, 1 client, and several compute nodes. The client will submit job to server. The server could listen to requests from client, and dispatch tasks to compute nodes. There are 2 types of tasks: (1). Map task, which means the node will analyze the content of a given input file, and write the sentiment score into an intermediate file; (2). Sort task, which means the node will sort the sentiment score of all files, and write the result to the result file.

### 1.2 Assumptions

We made these assumptions in this system:

1. The duration of load injecting is 0.5s
2. In load-balancing, if a task is rejected by a node #X, then it will be reassigned to a random node(this node might be #X again).
3. Do load injecting only after accepting the task
4. Do load injecting in both random and load-balance scheduling mode
5. There will be no faulty node during a job execution

### 1.3 Component design

#### 1.3.1 Common part

In the common part, we defined Task as a class, and it has 2 sub-classes: SortTask and MapTask. These classes will save variables of a task(like input file, output file, task type, etc.). Also, we defined a Thrift struct Address, which is used to describe a computer object(could be server or node).

#### 1.3.2 Client

The client accepts a task from users. It takes the directory of the data, server IP and port as input, with the pattern “java -cp ".:/usr/local/Thrift/\*" Client inputDir serverIP serverPort”. Then it makes an RPC call to connect to the server, send task requests with inputDir as variable to the server. Then it listens to the server and retrieves the information of success or failure. At last show user the success or failure information.

#### 1.3.3 Server

The server builds a TServerSocket with command java -cp ".:/usr/local/Thrift/\*" Server port. It listens to the call to this port, accepts the task requests from the client and accepts the parameters (location of data).Then it runs a multi-threaded Thrift server (TThreadedServer) to accept and execute multiple calls from client and nodes.

The server creates a ServerServiceHandler, which implements the methods in ServerService.thrift. The methods include:

bool addNode(1: Address.Address address): accept the address of the node, when a node call this function, it register the node in currentNodes

bool getSentVal(1: string inputDir): accpet the inputDir from the client, then register each file in inputDir as a map task in currentTasks, assign map tasks to the current nodes randomly (with an RPC call in doMapTask). If the task is rejected, put that task into currentTasks and reassgin it. Watch the complete situation of the map tasks (synchronized n\_complete\_tasks). If all map tasks are completed, build a sort task and assign (may also be rejected and reassign) it to a node (with an RPC call in doSortTask). The function also impletements logging and statistics of the tasks. After completion, it returns success or failure to the server, then to the client.

bool update(1: Address.Address address, 2: string taskFilename, 3: i64 duration, 4: string taskType): it will be called by a node once the node completes a task, then it increment n\_complete\_tasks by 1. It will also log the time spent to complete that task.

#### 1.3.4 Node

The compute node could do Map or Sort task. In each node, we implemented a QueueProcessor(which could store all tasks that accepted by this node) and a TaskProcessor(which is used to process each task in the queue). The server could make RPC call on AddMapTask() and AddSortTask() to add task on a specific node. In AddMapTask() and AddSortTask() functions, the node will check the load probability to inject delay, and decide whether to accept this task. If the task is accepted, the task will be placed into the queue.

In QueueProcessor, there is a while(true) loop to continuously check whether there are tasks in the queue, and create a TaskProcessor object to handle it if task exists.

In TaskProcessor, the task will be processed. The TaskProcessor class extends Thread class so the tasks could be run in parallel. Each task will be started in an independent thread. After finishing each task, it will make a RPC call to the server to announce that this node has finished a task.

#### 1.3.5 Map() and Sort() Function on Node

In Map() function, we will read the input file, replace non-alphabet symbols to whitespace by using regular expressions, and split each line into an array. In this way we could extract each word of the input file. Then these words will be inserted into a hashmap. Then we implemented a for loop to traverse each word in positive/negative word list, and look up the frequency of current word in the hashmap. In this way we could get the number of positive/negative words in the input file, calculate its sentiment score, and this will be saved into the intermediate file named by its original filename in med\_results folder.

In Sort() function, we will read each intermediate file in med\_results file, and insert the <filename, sentiment score> pair into a hashmap. Then we could do a quicksort on this hashmap by overriding the compare function of sort() method. Then the final result will be written into result\_file.

## 2 User document

### 2.1 How to compile

We have written a make script to compile the whole project.

cd pa1/src

chmod +x make.sh

./make.sh

### 2.2 How to run the project

1. Run server

cd pa1/src/

java -cp ".:/usr/local/Thrift/\*" Server 9090

2. Run compute node

The parameters are as follows:

java NodeHandler <Mode> <ServerIP> <ServerPort> <NodePort> <LoadProb>

<Mode>: Indicate whether it is random scheduling or load-balancing.

1: Random

2: Load-Balancing

<ServerIP>: The ip address of server

<ServerPort>: The port of server

<NodePort>: The port of node

<LoadProb>: Load-probability of this node (between 0 to 1.0). The node will run in random scheduling(accept all tasks assigned) if LoadProb==0.

Eg:

cd pa1/src/

java -cp ".:/usr/local/Thrift/\*" NodeHandler 1 csel-kh4250-01 9090 9060 0

OR

java -cp ".:/usr/local/Thrift/\*" NodeHandler 2 csel-kh4250-01 9090 9060 0.5

3. Run client

The parameters are as follows:

java Client <inputDir> <serverIP> <serverPort>

<inputDir>: Path of input data

<ServerIP>: The ip address of server

<ServerPort>: The port of server

Eg:

cd pa1/src/

java -cp ".:/usr/local/Thrift/\*" Client ../data/input\_dir csel-kh4250-01 9090

2.3 What will happen after running

The result will be generated in pa1/results/result\_file

Also, a log file will be generated in pa1/src/Logging.log

## 3 Testing document

### 3.1 Testing Environment

Machines:

We use 6 machines to perform the test, including 1 server machine (csel-kh1250-01), 4 computeNode machines (csel-kh1250-02, csel-kh1250-03, csel-kh1250-04, csel-kh1250-07), 1 client machine (csel-kh1250-08).

Test Set:

We use a test set (../data/input\_dir) including 500 txt items, totalling 204.4 MB. The size of a single document change from 21.4kB to 4.8 MB. The data uses a shared directory via NSF.

Logging:

Logging.log is stored under /src.

Testing Settings:

We test random-scheduling, load-balancing strategy with load probabilities range from 0.1~0.9, or different across different machines There is a 500ms load injection regardless of scheduling policies.

### 3.2 Random Scheduling

1) 1 node

number of map tasks: 500

number of sort tasks: 1

number of nodes: 1

map time (seconds): 39.719

reduce time (seconds): 0.688

2) 2 nodes

number of map tasks: 500

number of sort tasks: 1

number of nodes: 2

map time (seconds): 21.653

reduce time (seconds): 0.914

3) 3 nodes

number of map tasks: 500

number of sort tasks: 1

number of nodes: 3

map time (seconds): 14.458

reduce time (seconds): 0.733

4) 4 nodes

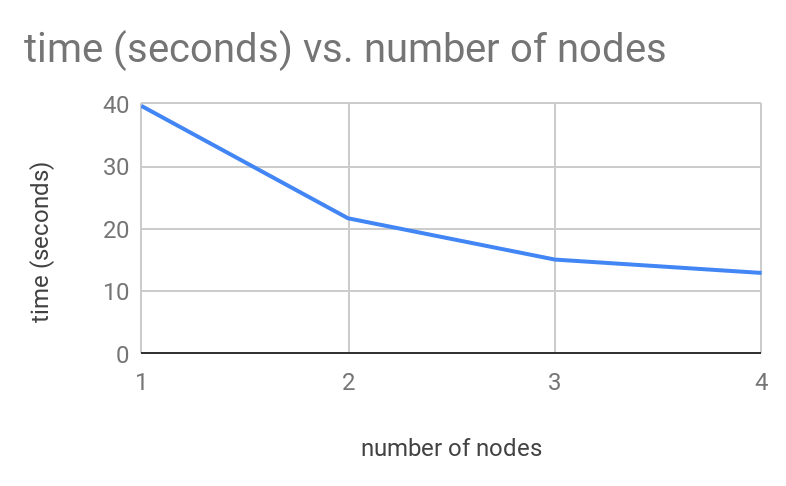
number of map tasks: 500

number of sort tasks: 1

number of nodes: 4

map time (seconds): 12.117

reduce time (seconds): 0.923



As the number of nodes increase, the time take to complete tasks decreases. But the decreasing ratio is becoming smaller and smaller. It seems that the total time spent will be close to 10 seconds but no less than 10 seconds. It shows a limit of increasing the computing capability by increasing the number of nodes. This may be caused by more time spent in communication and reallocation.

### 3.3 Load Balancing

Load Probability (0.1, 0.1, 0.1, 0.1)

----Statistics on Server----

number of map tasks: 500

number of sort tasks: 1

number of nodes: 4

map time (seconds): 20.08

reduce time (seconds): 0.929

----Statistics on Node----

|  |  |  |
| --- | --- | --- |
| Node | Received jobs | Accepted jobs |
| csel-kh4250-02 | 162 | 145 |
| csel-kh4250-03 | 130 | 116 |
| csel-kh4250-05 | 145 | 124 |
| csel-kh4250-06 | 130 | 116 |

Load Probability (0.3, 0.3, 0.3, 0.3)

----Statistics on Server----

number of map tasks: 500

number of sort tasks: 1

number of nodes: 4

map time (seconds): 33.291

reduce time (seconds): 0.746

----Statistics on Node----

|  |  |  |
| --- | --- | --- |
| Node | Received jobs | Accepted jobs |
| csel-kh4250-02 | 170 | 120 |
| csel-kh4250-03 | 205 | 144 |
| csel-kh4250-05 | 178 | 120 |
| csel-kh4250-06 | 161 | 117 |

Load Probability (0.5, 0.5, 0.5, 0.5)

----Statistics on Server----

number of map tasks: 500

number of sort tasks: 1

number of nodes: 4

map time (seconds): 45.296

reduce time (seconds): 1.425

----Statistics on Node----

|  |  |  |
| --- | --- | --- |
| Node | Received jobs | Accepted jobs |
| csel-kh4250-02 | 231 | 137 |
| csel-kh4250-03 | 230 | 126 |
| csel-kh4250-05 | 246 | 124 |
| csel-kh4250-06 | 241 | 114 |

Load Probability (0.7, 0.7, 0.7, 0.7)

----Statistics on Server----

number of map tasks: 500

number of sort tasks: 1

number of nodes: 4

map time (seconds): 59.888

reduce time (seconds): 1.264

----Statistics on Node----

|  |  |  |
| --- | --- | --- |
| Node | Received jobs | Accepted jobs |
| csel-kh4250-02 | 413 | 111 |
| csel-kh4250-03 | 422 | 126 |
| csel-kh4250-05 | 400 | 138 |
| csel-kh4250-06 | 404 | 126 |

Load Probability (0.9, 0.9, 0.9, 0.9)

----Statistics on Server----

number of map tasks: 500

number of sort tasks: 1

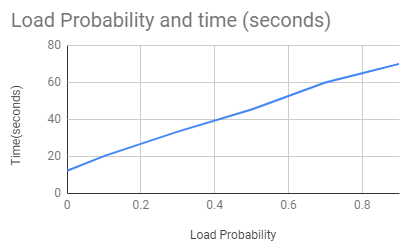
number of nodes: 4

map time (seconds): 69.963

reduce time (seconds): 1.281

----Statistics on Node----

|  |  |  |
| --- | --- | --- |
| Node | Received jobs | Accepted jobs |
| csel-kh4250-02 | 1183 | 130 |
| csel-kh4250-03 | 1157 | 126 |
| csel-kh4250-05 | 1145 | 128 |
| csel-kh4250-06 | 1101 | 117 |



As the load probability increases, the time take to complete tasks increases. The increasing ratio is becoming bigger and bigger. This may be caused by more time spent in communication and reallocation.

Load Probability (0.1, 0.5, 0.2, 0.9)

----Statistics on Server----

number of map tasks: 500

number of sort tasks: 1

number of nodes: 4

map time (seconds): 36.735

reduce time (seconds): 0.897

The time for (0.1,0.5,0.2,0.9) spent is longer than (0.3,0.3,0.3,0.3) and shorter than (0.5,0.5,0.5,0.5).

----Statistics on Node----

|  |  |  |
| --- | --- | --- |
| Node | Received jobs | Accepted jobs |
| csel-kh4250-02 | 251 | 221 |
| csel-kh4250-03 | 213 | 103 |
| csel-kh4250-05 | 211 | 157 |
| csel-kh4250-06 | 219 | 20 |

Load Probability (0.1, 0.5, 0.5, 0.9)

----Statistics on Server----

number of map tasks: 500

number of sort tasks: 1

number of nodes: 4

map time (seconds): 47.676

reduce time (seconds): 0.901

The time for (0.1,0.5, 0.5,0.9) spent a little longer than (0.5,0.5,0.5,0.5), which implies that a weak node (0.9) decreased the capability of the system.

----Statistics on Node----

|  |  |  |
| --- | --- | --- |
| Node | Received jobs | Accepted jobs |
| csel-kh4250-02 | 241 | 27 |
| csel-kh4250-03 | 235 | 117 |
| csel-kh4250-05 | 256 | 137 |
| csel-kh4250-06 | 249 | 220 |

### 3.4 Testing on negative cases

3.4.1 Invalid server ip / port

wang8662@csel-kh4250-03:/home/wang8662/CSCI5105\_UMN/pa1/src $ java -cp ".:/usr/local/Thrift/\*" NodeHandler 2 csel-kh424150-01 9090 9060 0.9

IP Address of this node: csel-kh4250-03.cselabs.umn.edu/128.101.37.3

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.

Failed to connect to server

wang8662@csel-kh4250-03:/home/wang8662/CSCI5105\_UMN/pa1/src $

If the server does not exist, the node will indicate that could not find the target server, and quit.

wang8662@csel-kh4250-01:/home/wang8662/CSCI5105\_UMN/pa1/src $ java -cp ".:/usr/local/Thrift/\*" Client ../data/input\_dir csel-kh12622-19 9090

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.

Client: Failed to connect to Server, retrying

Client: Failed to connect to Server, retrying

Client: Failed to connect to Server, retrying

Client: Failed to connect to Server, retrying

^Cwang8662@csel-kh4250-01:/home/wang8662/CSCI5105\_UMN/pa1/src $ ^C

If the server does not exist, the client will show the error information, and keep on retrying.

3.4.2 invalid input dir

wang8662@csel-kh4250-02:/home/wang8662/CSCI5105\_UMN/pa1/src $ java -cp ".:/usr/local/Thrift/\*" Client ../data/input1\_dir csel-kh4250-03 9090

Contacted to server csel-kh4250-03:9090

Invalid input dir

wang8662@csel-kh4250-02:/home/wang8662/CSCI5105\_UMN/pa1/src $

The client will indicate that the input directory does not exist, and quit.