5105 Project 3 Mijia Jiang

How to Run Test:

step 1:

./init.sh (This will generate Folder and latenc file for test)

step 2:

./filesys S latency_table_file time_out_max(sec) server_port

generate a Tracking Server which a latency file, a default time out(s) and a given port. The ip address will shown in screen.

step 3:

./filesys C matchID server_ip server_port

generate a client

./filesys C matchID server_ip server_port error_rate

generate a client with will send file with error (e.x. error_rate = 1)

Client UI Description:

[1] CONCURRENT DOWNLOAD FROM x.txt - y.txt (x, y and number)

- For concurrent test, in order to run this test, please provide enough files starting with number by init.sh or run following before starting server:

./filesys G folder_total file_total

file_total = n, it will generate 1.txt-n.txt under 1 to folder_total.

file_total = 1, it will generate i.txt under folder i (1<=i <= folder_total)

file_total = 0, it will generate empty folders.

./filesys R -> erase all folders with number index

[2] DOWNLOAD

(1) run "FIND" to get a client list.

(2) if success in "FIND", run "GET LOAD".

(3) execute "Peer Secletion"

(4) execute real DOWNLOAD operation

[3] SHOW INFO

This is will show average time and the source of downloaded files.

Test 1 (Same Latency - Even Distribution)

./clear.sh

./init.sh

./filesys S latency-table-20-MIN.txt 7 server_port

./filesys C 1 server_ip server_port (server_ip and port shown in server screen)

./filesys C 2 server_ip server_port

./filesys C 3 server_ip server_port

./filesys C 4 server_ip server_port

./filesys C 11 server_ip server_port (Please make sure 11 folder is empty)

You will see from the server screen that those clients are registered to server. The server screen will print out file name with the client id which owns this file. The client screen will print latency table got from server.

In the screen of client 11 (which has no files under it's folder):

1 - Concurrent downloading

1 20   -range (from 1.txt -20.txt)
3   -SHOW INFO
You will see the load distribution is around the same.

Test 2 (Different Latency - Uneven Distribution)

Change "latency-table-20-MIN.txt" to "latency-table-20-RANDOM.txt" from test. Wait for a while and run [3]. You'll see the file origin distribution is uneven.

Test 3 (Server Recover)

./filesys S latency-table-20-RANDOM.txt 7 server_port
./filesys C 1 server_ip server_port
Close server and then open server at the same port. You will see recovery.

Test 4 (Client Error)

./clear.sh
./init.sh
./filesys S latency-table-20-RANDOM.txt 7 server_port
./filesys C 1 server_ip server_port
close client 1
./filesys C 11 server_ip server_port
2
1.txt
You will see "Can't get load from available nodes". This means client is down so "GET LOAD" route fails.
./filesys C 2 server_ip server_port 1
In screen of client 11:
2
1.txt
You will see "Download 1 file from peer 2." and "Successfully update the new file list to the server". This means the algorithm is skipping dead node.

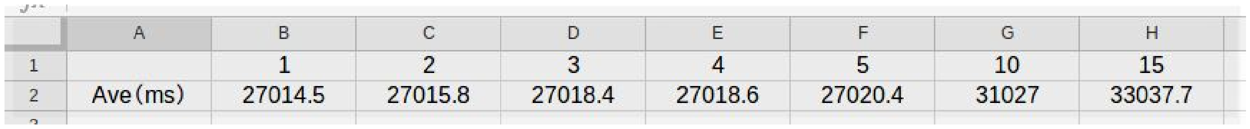
Test 5 (Transmission Error)

./clear.sh
./init.sh
./filesys S latency-table-20-RANDOM.txt 7 server_port
./filesys C 1 server_ip server_port 1
./filesys C 11 server_ip server_port
In the screen of client 11:
2
1.txt
You will see "Failed after trying all available nodes."
./filesys C 2 server_ip server_port
In the screen of client 11:
2
1.txt
You will see "Download 1 file from peer 2." and "Successfully update the new file list to the server". Because client 1 has no transmission error.

Negative Test 1:
Change range from "1 20" from "1 100" in case 1. You will see many resource unavailable and socket crash.
Negative Test 2:
Pass a wrong latency table name.

System Design

Message Encryption/Decryption
This is a file system built by UDP. There is a "SendableData" class holds int/bool/double/string, or a vector of SendableData. Before sending message via UDP, it'll have meta information such as length, checksum, status (OK or FAIL) There is also a MsgSt and Message class utilized SendableData.
Message Transmission
```
route 1:   register
client(registerSend)       ->  filename list     ->  server(registerRecv)
client(registerReceipt)   <-   global info    <-    server(registerRecv)
client(registerReceipt)   ->    ok/fail          ->   server(registerReceiptCheck)
```
The client should be able to register to the server by sending an array of filename that it owns, the server response with global configuration data.
```
route 2:    find
client    ->  file name       ->  server
client   <-   client list        <-    server
client   ->    ok/fail           ->   server
route 3 :    get load
client    ->                       ->  client
client   <-   load               <-  client
client   ->    ok/fail          ->   client
route 4 download
client    ->     file name     ->  client
client   <-   file                <-   client
client   ->    ok/fail          ->   client
route 5 update list
client    ->                       ->  server
client   <-   load               <-    server
client   ->    ok/fail          ->   server
route 6 pin
client    ->                       ->  server
client   <-                       <-    server
```
All routes will have corrseponding [routeName]Send/[routeName]Receipt/[routeName]ReceiptLogic and [routeName]Recv/[routeName]RecvLogic/[routeName]ReceiptCheck. The function attached by xxxLogic is handling the logic when message is normal. The message can also be corrupted or FAIL. Then routeNameReceipt and routeNameRecv will handle those different cases accordily.

Peer Selection Algorithm
Clients in client list is ordered by the downloader by (load+1)*latency. This ensures smaller latency will be select first. The "+1" comes from the case when load == 0 which will omit the effect of latency.
Error Handling
- Message Transmission Error
(1) For a corrupt message (checksum fails), the receiver will response FAIL message to notify sender this message is corrupt.
(2) For a FAIL message, the receiver won't respond because he knows the other side already received a corrupt message.
(3) For a OK message, the receive will run XXXLogic.
(4) For timeout, if it's route register of find, keep trying. For others, it stops.
- System Error
(1) File corrupt
    It's handled in transmission error handling. Also after the download route returns FAIL. The whole download process can select the 2nd best peer.
(2) Server is down.
    There is a global bool called "if_need_register" which'll set to false after registration. The client will pin server periodically, if it lose contact, it will set the bool into true. If it can pin, it will reregister.
(3) Client is down.
    The get load route will return FAIL and load will be set to -1. Download algorithm will skip those node.
(4) All available nodes are down.
    "Failed after trying all available nodes."
(5) Can't find a file.
    Notify user "Failed find the file: XXX". The get load and download route won't be accessed by download process algorithm.
Data Analysis
Average Duration Chart (1 client downloading from 4 clients with randomly generate latency)


	A	B	C	D	E	F	G	H
1		1	2	3	4	5	10	15
2	Ave (ms)	27014.5	27015.8	27018.4	27018.6	27020.4	31027	33037.7

The average time increases with size of concurrent downloading. But because of the benefit of concurrency operation. The increasement is not too large.

Load Distribution when download 10 files from 4 peers with same latency:
[2, 2, 3, 3] -> Because (load+1)*latency will be ordered based on only load.