COMP90015 Project1 Multi-Server Network

Team Information
Name: Fantastic Four
Members:

Name	LMS Login	E-mail
Ning Kang	ningk1	ningk1@student.unimelb.edu.au
Yiru Pan	yirup	yirup@student.unimelb.edu.au
Nannan Gu	nannang	nannang@student.unimelb.edu.au
Wenyi Zhao	wenyiz4	wenyiz4@student.unimelb.edu.au

1 Overview

This project is a multi-server system for broadcasting activity between a number of clients, critical parts include load balance, authentication mechanism, overlay network. All communication is based on TCP sockets.

1.1 Challenges

- Message Handler: multiple type of messages need to be processed in this project. We categorized into client handler and server handler to process specific messages respectively.
- Tree Structure Server Network
- Broadcast between servers

1.2 Outcomes

- Any number of authenticated severs join the system
- Users register to system with a unique username
- Users can log in any server within the network if he/she has registered in anyone server of this network. Anonymous users also can login.
- Users can send activity to the system and all other online users will receive this activity.

1.3 Architecture

The main idea of designing server architecture is using *Message-Handler Mapping pattern*. For every type of message, a handler will be developed to process its data. Different message handlers' instances are registered for particular types of messages when the server starts, so that the server can call relevant handler when message comes. For example, method *processMessage()* of the instance of *RegisterMessageHandler* will be called when the server receives a *{"command":" REGISTER" ...}* message. Figure 1-1 briefly illustrates this pattern.

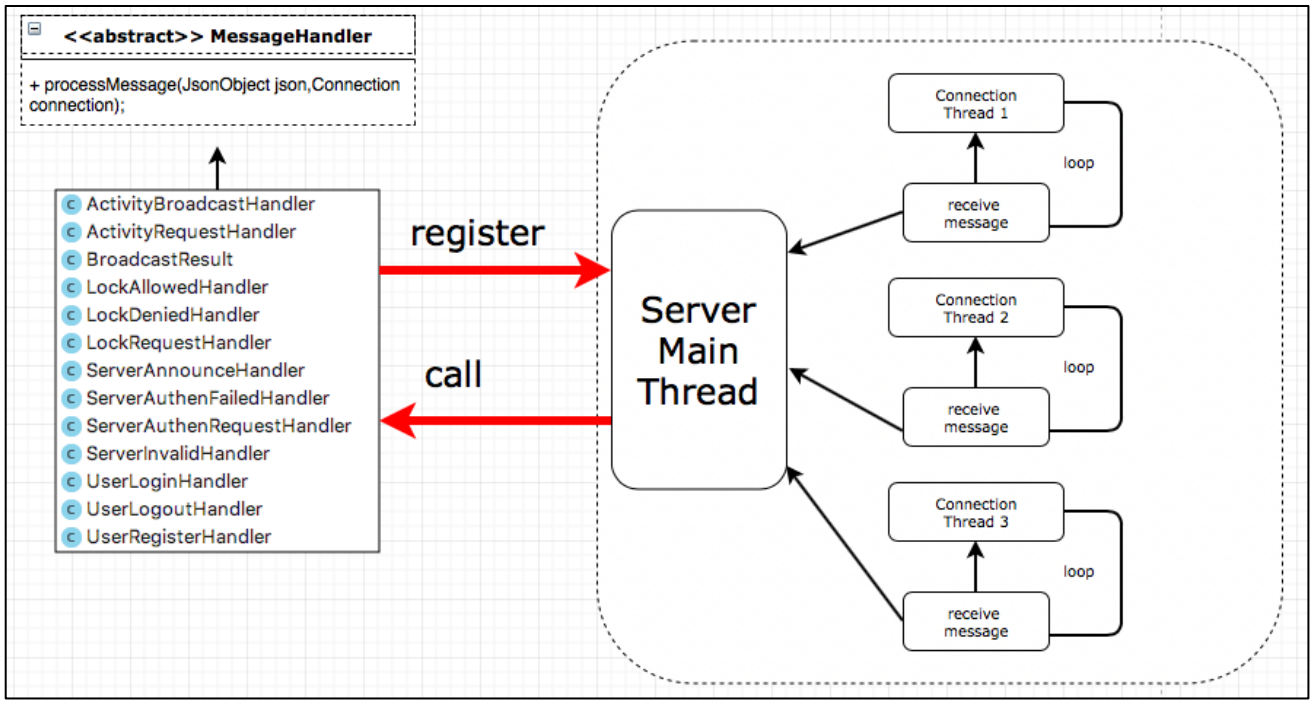

Figure 1-1 Message-Handler Mapping Pattern

2 Server Failure Model

2.1 Issues for original system

In the original system, no matter a server quits with or without a quit message(crash), 2 situations will happen:

- If only one server is connected with the quitting server: All clients connected with this server will not work normally which means they cannot use any services of the system unless they connect to a working server again.
- If two or more servers are connected with this quitting server: Despite the effect above, the whole system will be divided into several parts and each one works well as an independent system. But this is not expected as clients in different parts cannot send activity to each other.

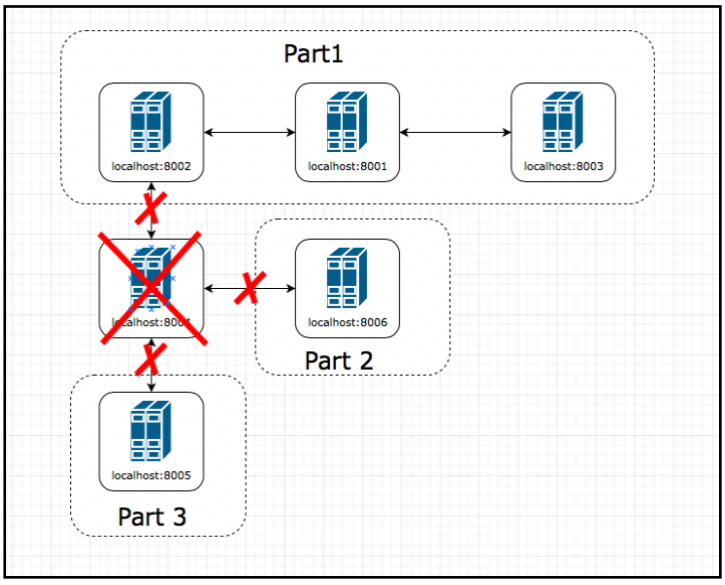

Figure 2-1 Bridge Server Failure

2.2 How to improve

2.2.1 Quit with a message

In order to keep providing services to all existing servers and clients, a strategy will be applied:

1. If more than one servers are connected with it, then pick one server as a "main" server randomly, let's call it Server M.
2. Send a message with below format to all other servers and clients.

Table 2-1 QUIT Message Format

```
{
   "command":"QUIT",
   "new_server_ip":"Server M's IP",
   "new_server_port":"Server M's port"
}
```

3. Servers and clients received this kind of message will redirect themselves to the given server.

With this strategy, the whole system can work well for server quitting with message.

2.2.2 Crash

In this case, there is no good solutions for clients/servers (who directly connect to this crashed server) and servers (who directly connect to this crashed server) to redirect themselves to a working server automatically.

A defensive strategy that backing up all relevant data of running server into a file every, for example, 10-15 minutes can be a very helpful way for recovery this server after crashing.

2.2.3 Server Restart

For the original system, the restarted server will lose user information that registered before its restart time. To improve this, a new type of message which contains all registered user information need to be replied when an existing server receives an "AUTHENTICATE":

Table 2-2 USER_INFO Message Format

```
{"command":"USER_INFO",
   "user_info":{
     "username01": "secret01",
     "username02": "secret02",
     "username03": "secret03",
      ...
    }
}
```

With this message, the new/restarted server can have a copy of all user information which will allow registered user login to this server.

And for our solution, once a server is crash, all data (information of users who registered in this server) will be lost. But as the report points above, an improvement plan is made that the program can back up all relevant data into a file so that the crashed server can use it to restore that kind of data.

3 Concurrency

3.1 Issues for original system

In the original system, there are two obvious concurrency problems:

1. In the register and login process, a client can log in the system before he registers the username successfully. Once a server receives a *LOCK_REQUEST* message, if the server has not recorded the username, it will record it and then client can log in to this server by using this username since the login system will only check the username and secret pair in the server's local storage. At this moment, because of the delay, it is possible that not all the servers in the system have received the *LOCK_REQUEST* message so the client has not received the *REGISTER_SUCCESS* message. Maybe the client will at last receive a *REGISTER_FAILED* message but he has already logged into the system with the invalid username and secret pair. (For example, one potential reason for the *REGISTER_FAILED* is that another client is registering by using the same username but different secret at the same time. Some servers have recorded this username and secret pair first, then clash occurs and another client win at last.)
2. Another obvious concurrency problem occurs in the redirect process. Because of the delay and interval of the server announce, a server is incapable of knowing other server's latest load state in any moment. So, if many clients log in the system at same time, they are possible to be redirected to a server which are not actually idle. Eventually, some clients may experience several times redirection.

3.2 How to improve

3.2.1 Improve register/login subsystem

To improve the register and login system, in our group's design, only the server which the client registers to will eventually store the username and secret pair after the server sends a *REGISTER_SUCCESS* message. So, the client can't log into the system until the register process is entirely finished since other servers won't know this client's username.

Besides, we can also arrange a *registerLockHashMap* on every server. Once a server receives a *LOCK_REQUEST* or *REGISTER* message, it will put username in the map and after replying these messages, the username will be removed. Through this mechanism, the system will deny the second client's request of registering a same username as the first client if the first client haven't finished the register process.

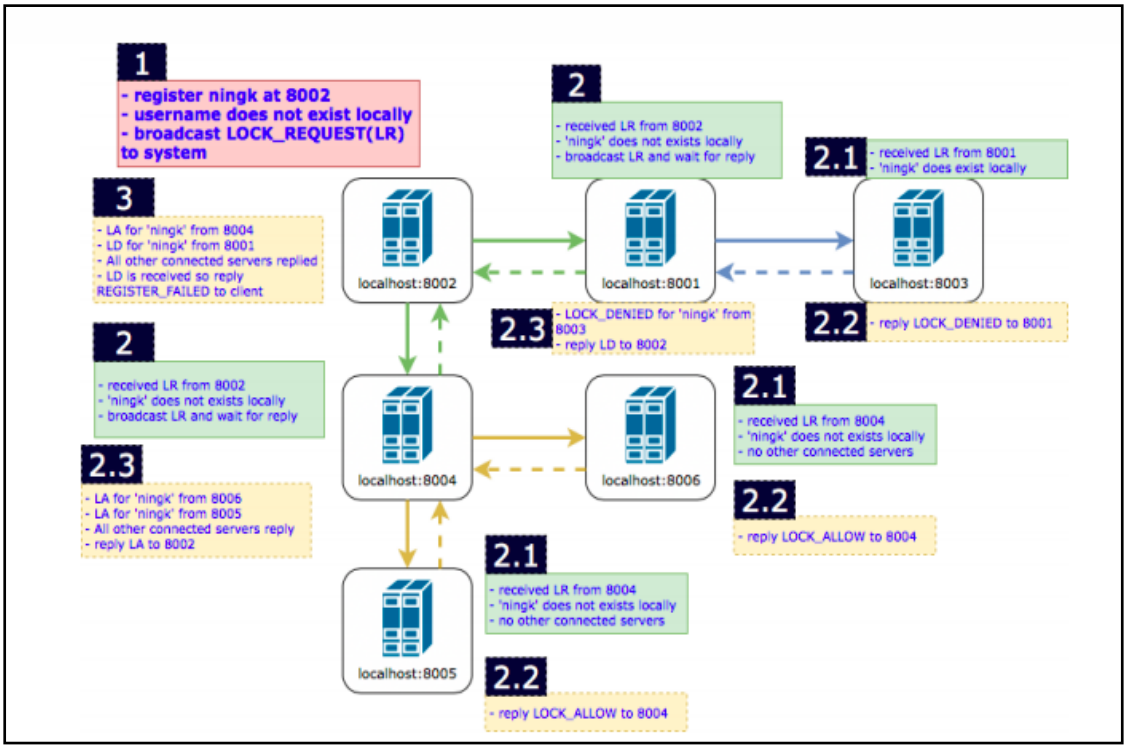

Figure 3-1 Improved Register Process

3.2.2 Improve redirect subsystem

To improve the redirect system, one solution is arranging a load balance server. This server will record all other servers' loads and handle all register or login requests first and then assign redirection. When the load balance server receives the login request, it will redirect the server to the lowest-load server in its local record.

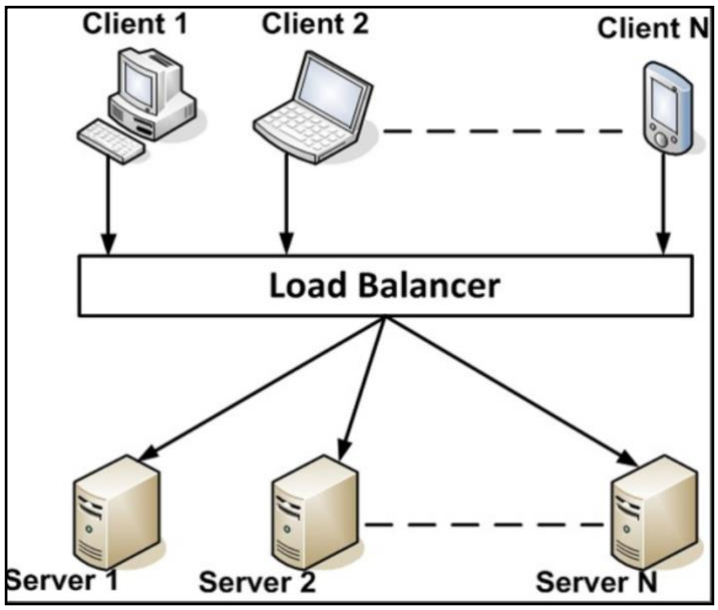

Figure 3-2 Load Balancer

4 Scalability

Per Specification, when a new user is registering in system, the server needs to broadcast lock requests to rest n-1 servers, and receive n-1 lock allow or lock deny. Thus, the total message in the system is $(n-1) *$

$(n-1)$, and message complexity is $O(n^2)$. The complexity is relative high, but the latency is low, clients don't need to wait too long when they are registering.

The alternative approach is as the Figure 3-1 Improved Register Process illustrates:
The registered server broadcasts lock requests to the connected servers, and the intermediary server won't reply lock allow or lock request immediately, conversely, the lock request will transverse to the "leaf" server in the tree structure, which is the last server in each branch. Then the "leaf" server will reply lock allow or lock deny along the branch until the message returns to the registered server. Regarding this approach, the total message when registering a new user in system is $(n-1) * 2$, and message complexity is $O(n)$. This approach indeed decreases complexity to linear, but it increases latency, since the registered server has to wait the message traverses to the "leaf" server then reply, which will increase the waiting time of clients when they are registering.

5 Appendix A – Meeting Minutes

Date: 29 Mar 2018
Duration: 2 hours
Attendances: Yirun Pan, Ning Kang, Nannan Gu, Wenyi Zhao

- Kick off
- Discuss overview requirement of this project and document high-level design.
- Initialize development environment.
- Analysis client side code and assign tasks for client development and Update WBS accordingly

Date: 13 Apr 2018
Duration: 2 hours
Attendances: Yirun Pan, Ning Kang, Nannan Gu, Wenyi Zhao

- Finalize and explain whole system architecture of communication
- Discuss message-handler mapping pattern
- Assign tasks

Date: 16 Apr 2018
Duration: 2 hours
Attendances: Yirun Pan, Ning Kang, Nannan Gu, Wenyi Zhao

- Task process tracing
- Simple overall testing of current project by task owners
- Test cases designing
- Test cases tasks assignments
- Useless code clearance

Date: 23 Apr 2018
Duration: 2 hours
Attendances: Yirun Pan, Ning Kang, Nannan Gu, Wenyi Zhao

- Testing results discussion
- Report questions analysis
- Documentation tasks assignment

Date: 27 Apr 2018
Duration: 1 hours
Attendances: Yirun Pan, Ning Kang, Nannan Gu, Wenyi Zhao

- Report finalizes

6 Appendix B – WBS

Task	Description	Owner	Days	Assign Date	Status	Update Date
Architecture Design	Detail design of the system	Ning Kang	4	29-March-2018	Done	13-April-2018
User Skeleton Implemention	Client command args, input validations and connection estimation	Yiru Pan	2	29-March-2018	Done	13-April-2018
User Register Handlers of Client	RegisterSuccHandler RegisterFailedHandler	Yiru Pan	2	13-April-2018	Done	16-April-2018
User Login Handlers of Client	LoginSuccHandler LoginFailedHandler RedirectHandler	Yiru Pan	2	13-April-2018	Done	16-April-2018
Other Handlers of Client	ClientInvalidHandler ClientAuthenFailedHandler ClientActivityBroadcastHandler	Yiru Pan	2	13-April-2018	Done	16-April-2018
User Register Handlers of Server	UserRegisterHandler LockRequestHandler LockAllowedHandler LockDeniedHandler BroadcastResult	NIng Kang	2	13-April-2018	Done	16-April-2018
User Login Handlers of Server	UserEnquiryHandler UserFoundHandler UserLoginHandler UserLogoutHandler	Ning Kang	2	13-April-2018	Done	16-April-2018
Server authen Handlers of Server	ServerAuthenFailedHandler ServerAuthenRequestHandler	Nannan Gu	2	13-April-2018	Done	16-April-2018
Activitiy Handlers of Server	ActivityBroadcastHandler ActivityRequestHandler	Nannan Gu	2	13-April-2018	Done	16-April-2018
Load Announce of Server	ServerAnnounceHandler	Nannan Gu	2	13-April-2018	Done	16-April-2018
Other Server side Handlers	ServerInvalidHandler	NIng Kang	1	13-April-2018	Done	16-April-2018
Test case design	Design test cases based on reguirement	Ning Kang	1	15-April-2018	Done	16-April-2018
Testing of Functionality of Server Auth	All related behaviours of server auth, only Server side	Yiru Pan	1	16-April-2018	Done	23-April-2018
Testing - Functionality of Register	All related behaviours of user register, includes both Sever side and Client side	Nannan Gu	1	16-April-2018	Done	23-April-2018
Testing - Functionality of Login	All related behaviours of user login includes both Sever side and Client side	Wenyi Zhao	1	16-April-2018	Done	23-April-2018
Testing - Functionality of Logout	All related behaviours of user logout includes both Sever side and Client side	Wenyi Zhao	1	16-April-2018	Done	23-April-2018
Testing - Functionality of Load Announce	All related behaviours of load announce, only Server side	Ning Kang	1	16-April-2018	Done	23-April-2018
Testing of Functionality of Redirection	All related behaviours of redirection, includes both Sever side and Client side	Ning Kang	1	16-April-2018	Done	23-April-2018
Testing - Functionality of Activity	All related behaviours of sending/receiving activities, includes both Sever side and Client side	Ning Kang	1	16-April-2018	Done	23-April-2018
Documentation - Readme.md	How to use this system and the basic logic of implementing functionalities	Ning Kang	1	23-April-2018	Done	27-April-2018
Analysis - Server Failure Model		Wenyi Zhao	1.5	23-April-2018	Done	27-April-2018
Analysis - Concurrency issues		Nannan Gu	1.5	23-April-2018	Done	27-April-2018
Analysis - Scalability		Yiru Pan	1.5	23-April-2018	Done	27-April-2018