COMP90015 Project 1: Multi-server Network

Group: NoTime4Kendo
Members: Yunxue Hu – yunxueh
yunxueh@student.unimelb.edu.au
Nian Li - nianl
nianl@student.unimelb.edu.au
James Liao - chienchaol
chienchaol@student.unimelb.edu.au
Zihua Xue - zihuax
zihuax@student.unimelb.edu.au

1. Introduction

The project aims to build a multi-server system for broadcasting activity objects among all connected clients. The system allows a client to register its username and secret with the system, to login using the username and secret combination once registered successfully, and to broadcast activity objects to all clients connected to the network once logged in. Also, the system will authenticate an incoming server connection to prevent mischievous actions. While receiving incoming connection from client, the system will also conduct load-balancing through redirecting client to another server in the system with less load.
The two greatest challenges we have faced is to merge command handlers wrote by different team members since each made different modifications to the Control and Connection classes, and integration testing since it requires a complex enough server architecture and frequent enough client activities. Overall, we have achieved all requirements listed in the project specification. But the system may expose potential problems under more extensive testing.

2. Server failure model

2.1 Arbitrary failure 1: message corruption

For the command of REGISTER, the server that receives the command will send out message containing username and secret to other servers. If the message corrupts during transmission, it will cause arbitrary failure consequently.

When the username or secret within the message corrupts, the original server that send outs the message will not have any action if it receives response from its connected servers. The reason behind is that the original server will store a suspended set of username and secret at the point of broadcasting message and the suspended set of username and secret will not match the corrupted username and secret.

For example, a client wants to register a username 'distributed' with secret 'abc' at server A and server A sends out LOCK_REQUEST to all the other servers. However, the message gets corrupted and username becomes 'distribute'. No matter what other server response, the username 'distribute' contained in the reply will never match the correct username. As a result, the original server is trapped and cannot reply to client.

To solve the problem, a timeout limit could be introduced to avoid infinite waiting. If the timeout limit is reached, the server will send out error information to the client and close the connection.

2.2 Arbitrary failure 2: server alternation

When the original server is dealing with the REGISTER command, the original server will send out LOCK_REQUEST to all other servers within the system. In the meanwhile, the variable 'lockCounter' will be initiated to record LOCK_ALLOWED received from original server's connected servers. While lockCounter equals to the number of original server's connected servers, the registration process is considered as success, and corresponding actions are executed.

However, the arbitrary failure will happen if a new server is connected to the original server after LOCK_REQUEST has been broadcasted from the original server. In this case, the number of original server's connected servers will always be one more than the number of LOCK_ALLOWED received. As such, the value of the number of original server's connected servers is set wrong. Consequently, the REGISTER command from the client will never receive any response.

To avoid this kind of failure, a variable 'currentServerNum' can be used to record the number of connected servers at the point when LOCK_REQUEST is broadcasted. In this way, the original will compare lockCounter to currentServerNum to check if all servers have replied its status successfully. When the lockCounter equals to currentServerNum, the original server will return REGISTER_SUCCESS message and reset currentServerNum to zero. On the other hand, if the original server receives LOCK_DENIED, it will return REGISTER_FAILED and reset currentServerNum to zero.

3. Concurrency issues

3.1 Possible concurrency problem

Basic assumption: Network becomes enormous, countless servers will be connected. We will see several possible situations stated as below:

3.1.1 Registration failure due to duplicate request on two different servers within the system

(See table 1) Server A is the leftmost server, and

server C is the rightmost, problem arose when 2 servers receive same register request. Server A has a client send REGISTER request, and serve A immediately broadcasts LOCK_REQUEST and waits for reply. Server C has another client wants to register same username and secret. We assume server B receives server A's request first, server B then checks its local storage first, and no match is returned, then put it in to its hash map. Then server C's request has arrived, it finds out there's already same username existed, return LOCK_DENIED. In fact, same kind of problem will happen to server A, server A and server C will both receive LOCK_DENIED, and client fails to register.

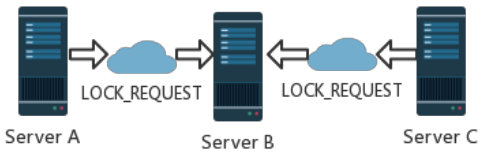

Table 1: model structure

3.1.2 Login and registration take place on the same server at the same time

After server receives registration request, it needs time to get response. If client A wants register username "Tom" with "tom_1" as secret, server has checked local storage, added new user information and send out LOCK_REQUEST. Meanwhile, client B gets connected with same server and insist to log in as "Tom" with "tom_1", server check local storage, gives permission to log in. In fact, there's already "Tom" with different secret "tom_2" existed in system, server refuses client A's registration request, while client B has still logged in.

3.1.3 Multiple registration on the same server

Another possible failure is multiple registration issue. To explain the situation, we assume two clients attempting to register on the same server. When the original server is dealing with the first registration command coming in and send out LOCK_REQUEST, the second registration command then reaches the original. According to the current process, the information (username and secret of the second client connection) will replace the first client connection's information in the suspended storage area. This will lead to the consequence that original server loses track of replies to LOCK_REQUEST for the first client. Registration for first client is completely abandoned after second client's registration's command reaches. The previous introduction of the variable 'currentServerNum' can also be used as a solution to multiple registration issue. When the registration command comes in every time, the process will check if currentServerNum equals to zero. If the variable equals to zero, it means that the original server is not dealing with any registration command. Under this circumstance, LOCK_REQUEST will be sent. Otherwise, the original servers will decline the registration command, send back error message, and close the connection.

3.2 Possible Solutions

3.2.1 Add timestamp to each process

Each command related to read and write (LOGIN and REGISTER) will be attached with a timestamp, to indicate when does it initially take place. Servers read timestamp first, if new process happens early, store current process into buffer and execute new process first.

3.2.2 Make login has priority in one server

When registration requests to update current user table, this couldn't be committed unless server has no pending login request.

4. Scalability

4.1 Scalability analysis of current approach

Registering a new client: Supposedly there are N servers connected in the whole system. When registration happens in sever D, server D would broadcast to all servers, "LOCK_REQUEST" takes place N-1 times.

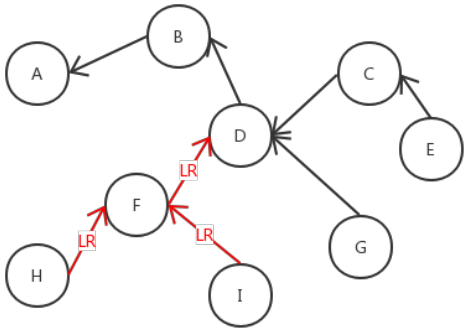

Table 2: structure of a large-scale system

Server F wouldn't send back LOCK_ALLOWED, unless received allowed request from H and I. Meanwhile, once server received LOCK_DENIED, registration immediately failed. Hence, successful registration takes more time, which means registration needs at most 2(N-1) messages.

4.2 Further improvement

Caching: When processing login and registration, if a new user information is received, server can add it to local cache store, rather than discard. The next time, server would check cache first, to avoid broadcast every time.

Network layering: When the system is getting bigger in terms of size, the system could divide its servers into several smaller groups. For each group, we pick a random server in the group as representative for the group, which contain all user information that is stored within the servers. If the command such as LOGIN or REGISTRATION

comes in, the system will contact representatives for each group at first attempt. The benefit of this approach is to decrease the number of message delivery.

Replication: Coupling with the group representative structure mentioned above, we use replication as a supporting method. For each representative, they will exchange their information with other representatives. As a result, each representative will gather all the information in the system. When each server in the group tries to send out REGISTER or LOGIN request, it only has to check with its representative of the group.