COMP 90015_2018_Semester_1

Distributed Systems

Project 1

Group Name: The Walking Head

Member Name	LMS Login Name	Email
Mengfeifei Zhang	mengfeifeiz	mengfeifeiz@student.unimelb.edu.au
Sichen Zhao	sichenz	sichenzsichenz@student.unimelb.edu.au
Xiaohan Fan	xiaohanf2	xiaohanf2@student.unimelb.edu.au
Fengke Song	fengkes	fengkes@student.unimelb.edu.au

1 Introduction

The aim of this project is to build a multi-server system which has several functionalities, such as client registration, client login(anonymously or non-anonymously), activity-objects broadcasting. During the designing and implementing the system, our team has faced a few challenges, and all of which have been successfully dealt with. The challenges include:

1. Some difficulties in implementing multiple threading management and resources release after one thread is ended.
2. There are many types of exceptions that need handling, which stand in the way of successfully designing and running the system.
3. Some protocols we are asked to implement are kind of ambiguous, which affects the designing of the system.

The outcome (the submitted version of the system) is a runnable, stable, and fast one which has all the features and characteristics required. During the developing process, not only has our team well-met all the requirements mentioned in the plan, but we have learned a lot from one another and generated a friendly and harmonious environment, which is beneficial to both our studying and our living.

2 Server Failure Model

According to the revised assumptions, there might be 4 possible scenarios about server quitting or crashing:

- Server A quits (by sending a quitting message)→A never starts again
- Server A quits (by sending a quitting message)→Server A restarts eventually
- Server A crashes (no quitting message)→Server A never starts again
- Server A crashes (no quitting message)→Server A restarts eventually

Below lists the potential problems and solutions to them (suppose server A is the quitting or crashing server):

1. As for quitting scenarios (scenarios 1 and 2), server A will send a quitting message to other servers, which shall be aware of the no longer existence of server A and kick it out from the server lists (Every server has a server list which contains the IP address of any other server) and make no more communications with it. Because all the servers user lists are the same, it would not be a problem if server A never starts again (scenario 1) since the registered users could be connected to other servers. However, if server A restarts eventually, since its user list is empty, none of the registered users can successfully login to this server, which is a problem. Therefore, a server must update its user list right after its authentication. Firstly, instead of keeping users information in an online HashMap, we could use a local XML file to store data. And whenever a new user registers, the XML file gets overwritten. Secondly, when a server tries to join in, its XML file must be updated to the latest version, which could be done by broadcasting object.Figure 1 shows the details:

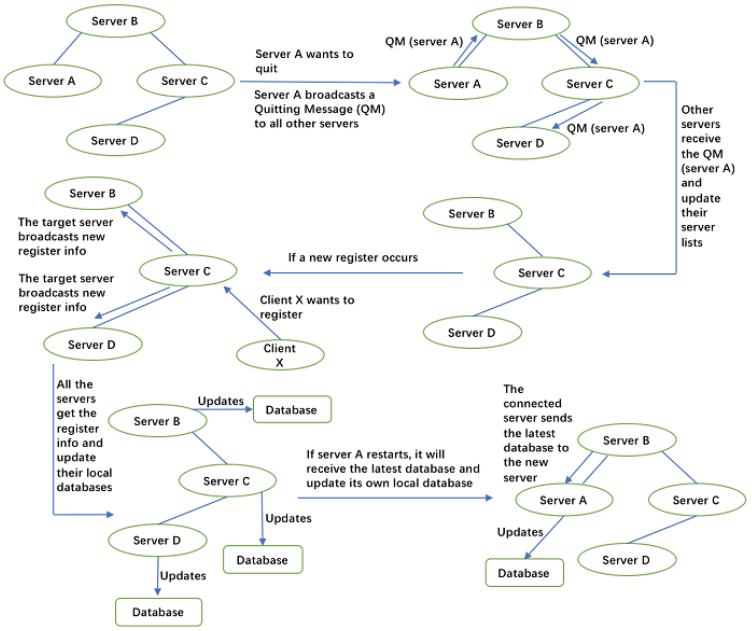

Figure 1: Server failure model 1

2. As for crashing scenarios (scenarios 3 and 4), since no messages are sent from server A, other servers will not know it is crashed and keep sending messages to it, which could lead to forever waiting of the connected server and freeze the whole system. Therefore, a timer is necessary to keeping the system active. If server A was not able to reply in time, then it is considered to have quitted, in which case the connected server will broadcast a message to other servers to declare its quitting. And server A will be deleted from the server lists. Once the crashed server restarts again, it would be treated as a new server even if the IP address and port number remain unchanged. Figure 2 shows the details:

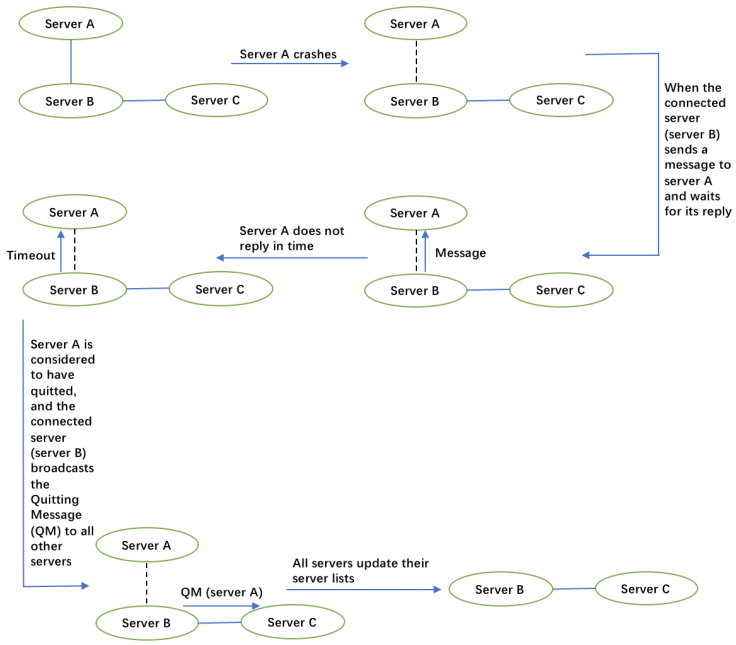

Figure 2: Server failure model 2

3. There is another situation that could lead to a severe problem. Since every server is connected to at least one of the other servers, if a server with more than one connections quits or crashes, the servers linked to it can no longer communicate. In other words, the server network is broken. To solve this, we came up with an idea that whenever a quitting message is received, one of the connected servers will broadcast a checking message to all other servers. And if any one of the servers does not reply, a new connection will be made between these two servers. Figure 3 shows the details:

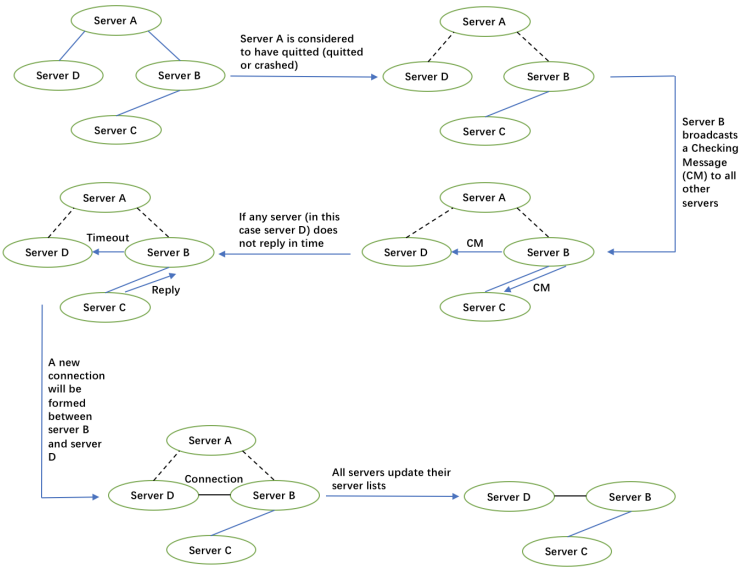

Figure 3: Server failure model 3

3 Concurrency issues

There are mainly three concurrency issues in this system which are discussed below in detail:

1. The first issue occurs when at least two clients register at the same server at the same time, and the intended registering usernames are also the same. For example, as Figure 4 illustrates, client A and client B have connected to the same server S. Suddenly at some point, each of the two clients sends a registering request to the server, in which the usernames are the same (XXX) but the secrets are different.In this case, server S receives two requests at the same time by two separate threads. After that, these two threads will check the user information table at the same time and neither of them finds XXX. As a result,both two requests are approved, which is against the rule that one username can only have one secret.

   To address this issue, a buffer mechanism is required. In fact, there exists a certain thread lock called synchronized in JAVA which can deal with the concurrent requests. In detail, when making the register method synchronized, it will select only one request at a time to execute.In this way, the first concurrency issue can be avoided.

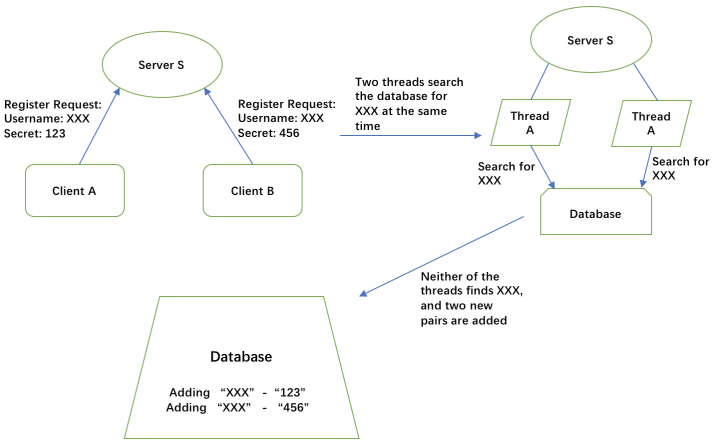

Figure 4: The first concurrency issue

2. Another concurrency issue arises when at least two clients register at different servers with the same username at the same time. As is shown in Figure 5, client A is sending a register request to server X. Meanwhile, client B is sending a register request to server Y.In this case, both server X and server Y will send LOCK REQUEST to server Z at the same time. As a result, server Z must find a way to deal with the two requests, otherwise there will be a concurrency issue.

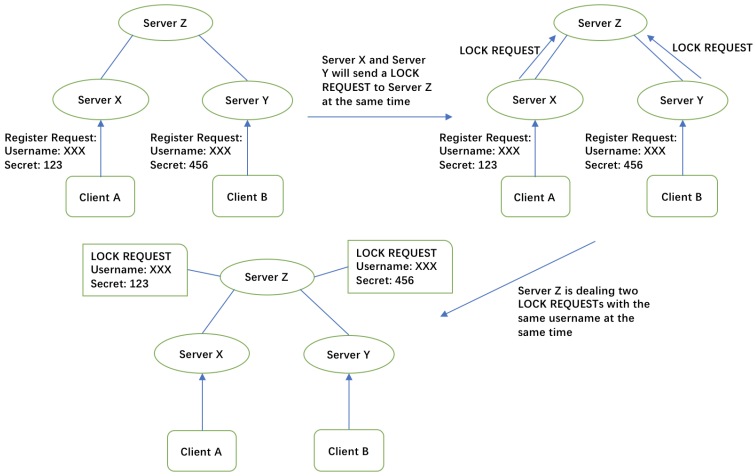

Figure 5: The second concurrency issue

   To cope with it, the synchronized mechanism is also applied, which permits only one thread to be running at one time. In specific, server Z will firstly accept the request from Server X and broadcast this information to all other servers. And only after server Z gets the first result will it deal with the request from server Y. In this way, the concurrency issue can be avoided.

3. Since each server needs to broadcast announcement information to all other servers once in every 5 seconds in this system, if the server network contains thousands of individuals, the broadcast may not be finished in time.To deal with it, we can use the concurrent method. In this way, servers can allocate the announcement tasks to different threads and use different CPU cores to finish the job.

4 Scalability

4.1 Message Complexity Analysis

The process of registering a new user in the system is shown as follows (assuming there are n servers in the server network):

Step 1: Server A receives a registering request and broadcasts the LOCK_REQUEST message to the other n-1 servers.

Step 2: Any of the other n-1 servers will broadcast the LOCK_ALLOWED message or the LOCK_DENIED message to all other n-1 servers respectively (not including the broadcasting server itself).

Step 3: Server A returns the final reply to the client as to whether the register is successful or failed.

Figure 6 illustrates the process:

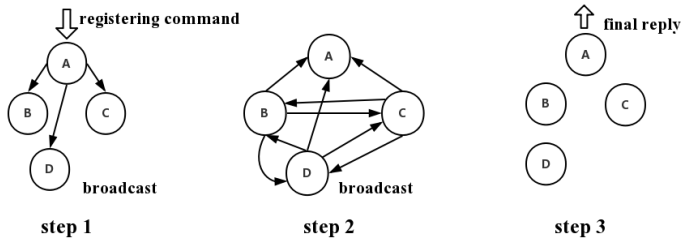

Figure 6: Registering process

Therefore, the original complexity of this process is$1 + (n - 1)(n - 1) + 1$, which approximates to n^2.

4.2 Approaches to Improve the Scalability

The following section shows two approaches to improve the scalability of this system.

- **Reducing Message Complexity**

  A tree data structure can be used in the process of message sending to reduce the message complexity by making all severs as tree nodes. The modified process of registering a new user is shown as follows:

  Step 1: Make server A (the server which receives the registering request) the root of the tree.

  Step 2: From the root on, each parent sends the LOCK_REQUEST message to its children unless it is not a parent.

  Step 3: From each leaf, the server sends the LOCK_DENIED message or the LOCK_ALLOWED message to its parent unless the parent is the root. If anyone of the childrens reply is LOCK_DENIED, the parent server will send a LOCK_DENIED up to the root. Otherwise the parent will check its own database and return the corresponding message.

  Step 4: Server A returns the final reply to the client as to whether the register is successful or failed.

Figure 7 illustrates the process:

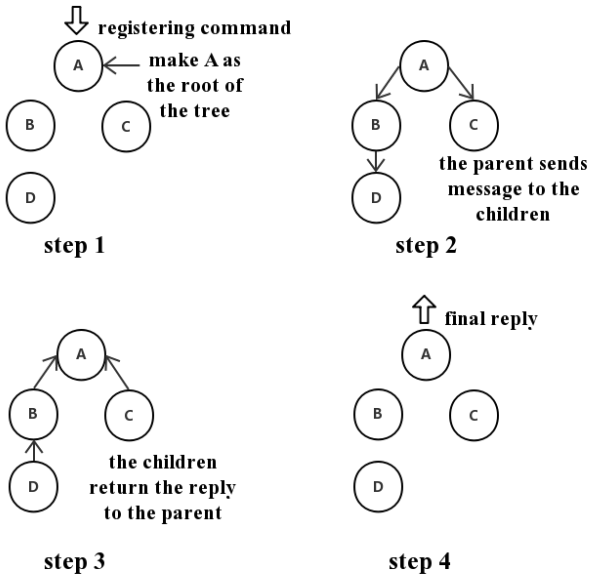

Figure 7: Modified registering process

Therefore, the new complexity of this process is $1 + (n - 1) + (n - 1) + 1$, which approximates to n.

- **Transmitting User Information to the New Server**

  In the current system, the scalability is not guaranteed. One example is told by the following story:

  1. The system has run finely for some time and has a number of registered users.
  2. A new server is connected to the server network.
  3. A registered client wants to login at the new server.
  4. The login process will fail because the new server does not know any information of the registered users (the database is empty).

  In order to solve the problem, a data transmitting approach can be used. In detail, whenever a new server has joined in the system, it will automatically update its user database by receiving the data from its parent server (a part of Figure 2-1 also illustrates this process). In this way, the login process will be successfully implemented, and the scalability is guaranteed.

Appendix

The Discussion Records

During the project, the team has held several meetings and one video conference, and all four team members had attended them, which simplified the job a lot due to the good attitudes and friendly communication environments. All of the meetings have been recorded, and Table 1 shows the details:

Date	Place	Period	People	Contents
29th, March 2018	UoM	2 hrs	All team members	Division of work
4th, April 2018	UoM	2 hrs	All team members	Discussion of project concept
7th, April 2018	UoM	3 hrs	All team members	Discussion of class models
12th, April 2018	State Lib	3 hrs	All team members	Discussion of completed parts
16th, April 2018	UoM	1.5 hrs	All team members	Discussion of completed parts
20th, April 2018	UoM	2 hrs	All team members	Discussion of of project exception
22th, April 2018	UoM	4 hrs	All team members	Project function test
25th, April 2018	UoM	3 hrs	All team members	Project wrapping up

Table 1: Discussion records

Besides, during the project, the team members have used WeChat as the communication tool. Although the online discussion records are not provided due to the characteristics of WeChat, a lot of great ideas were shared and files were exchanged in our WeChat team group.