COMP90015 Project 2: High Availability and Eventual Consistency

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

Project 2 aims to improve our previous project that builds a multi-server system for broadcasting activity objects to all connected clients. In the report, a new server protocol will be designed and implemented to ensure better availability consistency given possible server failure and network partitioning. The focus will be more on availability since availability and consistency cannot be both optimized, according to CAP theorem. We assume eventual consistency for our system. The biggest challenges we have faced is to design appropriate data structure to restore user targeted buffer by decoupling users with connections or servers, so that no matter which server the user logged back in, the user would receive the message targeted to him.

2. Server connection restoration

As stated in project specification, our protocol should guarantee that 1) all clients that are connected to the network at the point when one specific client conducts activity will eventually receive message, and 2) all the activity message sent by client will be delivered at the same order at each receiving client. It is noteworthy that the order that messages are sent by different clients does not need to be enforced at receiving clients.

To achieve the above two aims, we leverage message buffer mechanism as our solution. Before discussing on how message buffer works in our system, we will firstly explain connection recovery between parent and child server. While connection error happens between servers, the existing server cannot know if the error is triggered by temporary connection break, which the connection will be fixed eventually, or the server crash. We will set up a timeout of 6 seconds to determine the actions we will take to restore the network operation.

2.1 Temporary connection break

Before timeout, the system believes that connection error comes from temporary connection break. Consequently, each child server of the lost server will try to restore the connection after the child server is aware of connection lost. Server identification plays an important role in this searching mechanism. We use the combination of server's localhost name and local port to identify every server in the network. As our network has a tree structure, every server (except for root server) has only one parent server. We will store this connection separately and regulate the child server to check if parent connection is still on and try to restore connection based on the server identification after it detects the parent server broke.

2.2 Server crash

After timeout, our system will judge that the server has crashed. To restore the network, the child server of the crashed server will try to connect the grandparent server. For each server, it will store its grandparent server's hostname and port number. In the special case that the server itself is a root server of the network, the server will have to choose a backup server among its children.

In this section, we will talk about how a server get its grandparent (or backup) server information. When a new server wants to join a network, it will send out an authentication command to the parent server. After the authentication succeeds, the parent server will examine whether itself is a root server or not. There are two situations to be discussed from this step. If the parent server is not a root server, it will trigger a 'sendingParent' command. The parent server will send out its parent information along with the command to the child server (newly connected server). After the child server receives the command, it will update the grandparent information based on the message received. As shown in Figure 1, server B and D will get A's information from C, then store it as grandparent address. The other case is that parent server figures out that itself is a root server, and it does not have parent as a result. It will search through all its connected server and picks the first connection that is open as its backup server. In the next step, it will send out its backup information along with a 'sendingBackupRoot' command to the child server. The child server will consider the backup server as its grandparent. Figure 2 demonstrates that server A chooses server B as backup server, and all his children get this information, including B.

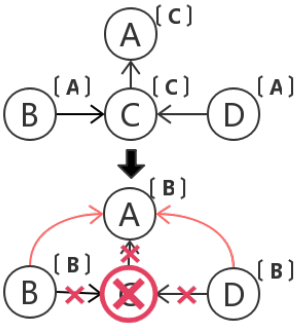

Figure 1. Server crash (not root server)

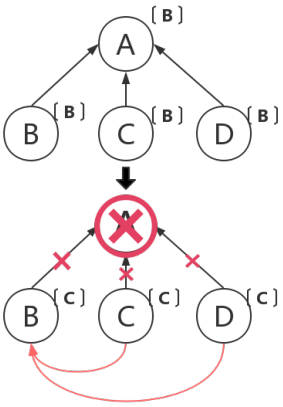

Figure 2. Root server crash

After a server crash incident happened, the children of this server will try to reach to the original server within a limited number of attempts. If the connection still unable to establish, those servers will connect to pervious grandparent or backup server. As shown in figure 1, after C crash, server B and D connect to server A; figure 2 demonstrates server C and server D connect to B. Hence, the child server has restored its connection.

The whole system need further steps to recover. The child server will send out a authentication command to its new parent server as the network structure has been changed. When the new parent receives the authentication command, it will examine whether itself is a root server and send back 'sendingParent' or 'sendingBackupRoot' command accordingly. Also, as the state of network is dynamic, every server in our system will leverage server announce to check if its grandparent server equals to its parent's parent server (or parent's backup server, under the circumstance that the parent is a root server). When inconsistency detected, the server will update its grandparent information.

3. Message buffer

Message buffer aims to make sure if there's an exception happening, servers and clients are still able to get the message. To deliver message buffer function, we will store two different hash maps in each server, which are named 'serversConnections', and 'activeUserBuffer'. The 'serverConnnections' map will use each server connection as key and store a string set of the usernames of all connected clients that could be reached through the connection. The server state will be updated through every server announce mechanism in our system. In our system, server announce will be executed in three situations: 1) every 5 seconds time interval past, 2) login of new client and 3) logout of existing client. The 'activeUserBuffer' hash map will build pairs for every client that is connected to the server and client that has message not successfully delivered. The 'activeUserBuffer' hash map will use username as the key and user buffer as the value. User buffer is a queue for buffering those messages not delivered to the specific user yet.

When the server detects that one of the connections is not available, the server will check the 'server' hash map to find out all the usernames that only can be delivered through that server. Based on the list of usernames, the server will put message into the user buffer. The server will also check every message buffer, send out and all the messages in it and delete the client's connection once the server receives server announce that shows the client has come back to the network by connecting to another server. In this way, every client will get its messages when it logins to the network again.

4. Client crash and duplicate client login

In our system, we also realize the aim that allows clients to login at any time. There are two situations that might cause potential failure to the system, which are client crash and duplicate login. When the client crash, the client will leave the system without any notice to the network. After the failure, the lost client will try to rejoin the network, but might not login to the previous connected server. Once the client joins the network again, the newly connected server will broadcast the server announce to notify the network. When the previous connected server receives the server announce, it will send out the message buffer that belongs to the lost client and delete the client's connection from the server as it is no longer connected to the previous server.

The other situation is called duplicate login. For example, there are two servers, which are named A and B, and there exist only one client that is connected to server A in the network. Then, the user uses the username and secret of the client to login on server B. This is a typical situation of duplicate login. In the event of duplicate login, the pervious server will also proceed the same procedure as client crash. As a result, the original client that was connected to

server A will be deleted, and the newly login client that is connected to server B will exist in the network. In this way, the system also stays consistent through maintain only one client that uses a certain pair of username and secret.

5. Redirect client to achieve load-balancing

We complete the aim of ensuring clients are evenly distributed over the servers as much as possible in our system. When a new server is setting up to join the system, clients can be redirect to this new server. After receiving server announce, the server will get the latest information of client load for every server in the network. The redirection mechanism we designed will be executed every 5 seconds interval. If there exist a server that meet the requirement, the original server will send redirect message that contains address and port number of the available server to the first open client connection among all client connections and disconnect that specific client. After receiving the redirect message, the client will automatically login to the designated server. By doing this mechanism, the network will try to achieve load-balancing at least every 5 seconds. Eventually, the clients will be distributed evenly over the servers.

6. High availability & Eventual consistency

Users are able to join the system whenever they want. Once they have successful logged in, they are able to send message at any time. Even this message runs into server crash, message will be write into buffer ("activeUserBuffer" in each server) until system connection is eventually fixed. If client suddenly crashed, server will store all the message in its buffer and inform all server this client has crashed, making sure every server has noticed this. After a while, this user can log into any servers while accessing to the same message. Until now, this server can broadcast again to clear user's message buffer.

The system will be eventually consistent within limited time. Even though there's a server crash or client crash, unreached message is still in the message buffer, waiting for user to read. On the other hand, this buffer is defined as queue, the order of sending message is preserved. The redirection mechanism makes sure whenever server find an unbalanced situation, client will be redirected.

Appendix: illustrative example

Note: The state of the network will be updated through server announce. Due to the fact that the updated time interval is 5 sec, some functions will have delay to a certain degree.

- **Aim 1&2: allow server and client to join the network at any time**

Step 1:

We initiate a single server A and a client A login to server A. Then, client A logout from the server A.

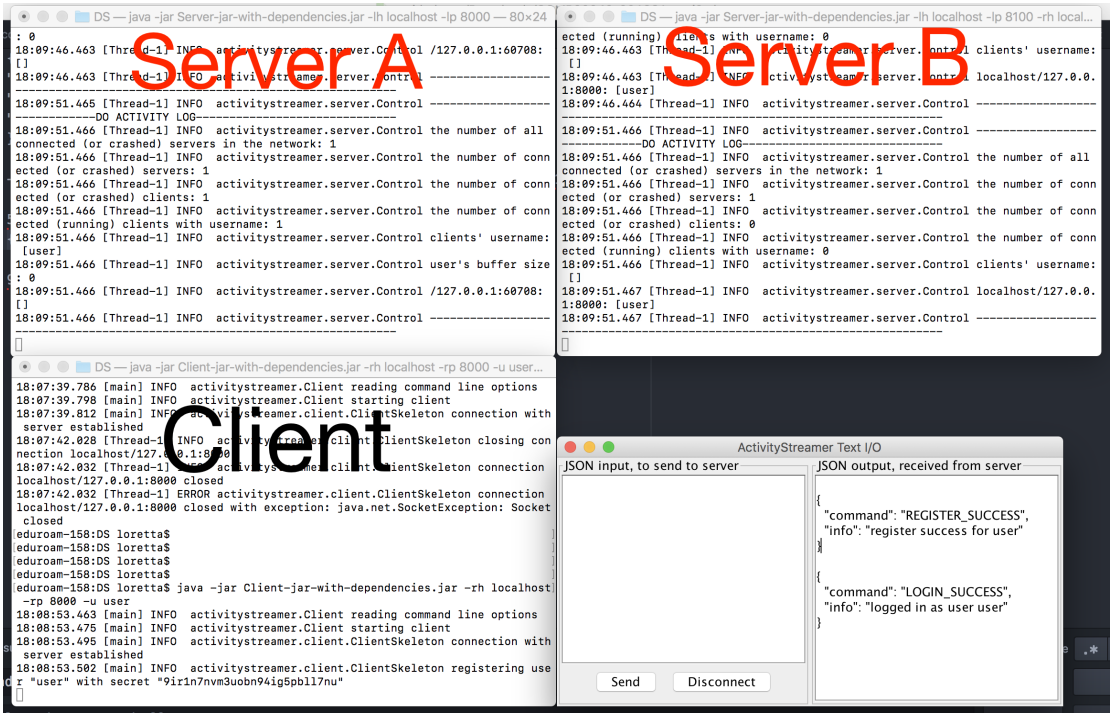

Figure 1-1. Server A, client A, log in, log out

Step 2:

We initiate a server B connecting to server A. Then, client A successfully login to server B.

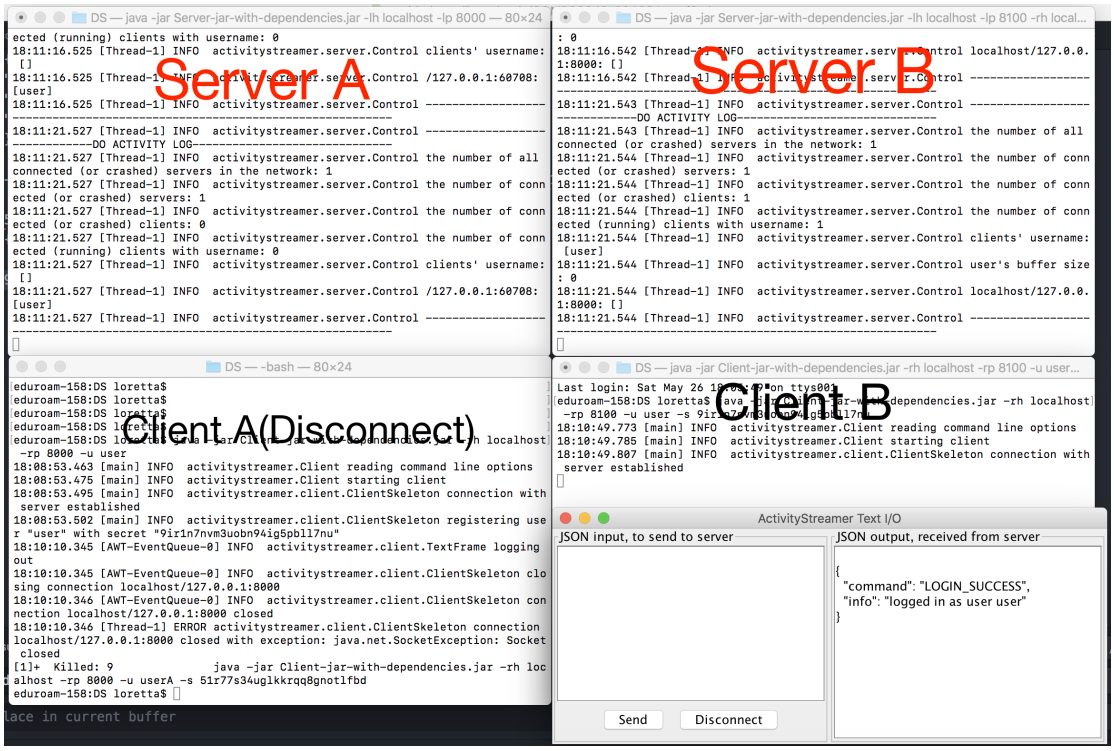

Figure 1-2. User logs in Client B

Step 3:

We let client A login on server A and this action will succeed. In the meanwhile, client A's connection to server B will be disconnected. As we can see in Figure 1-3, Server A accepts the connection, while Server B doesn't have any connection anymore.

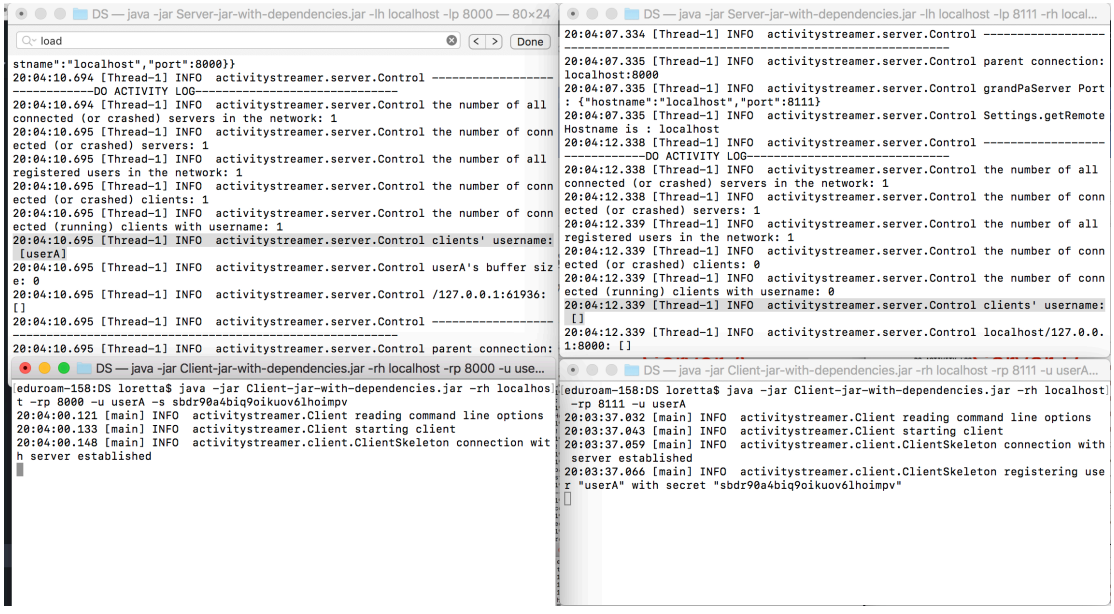

Figure 1-3. User logs in Client A

- **Aim 3: maintains that a given username can only be registered once over the server network**

Step 1:

We initiate server A and server B. Server B is connected to server A. Then, we register a client with username ‘userD on server A.

Step 2:

We logout the client with username ‘Tom’. Then, another user tried to register username ‘Tom’ on server B, and it fails.

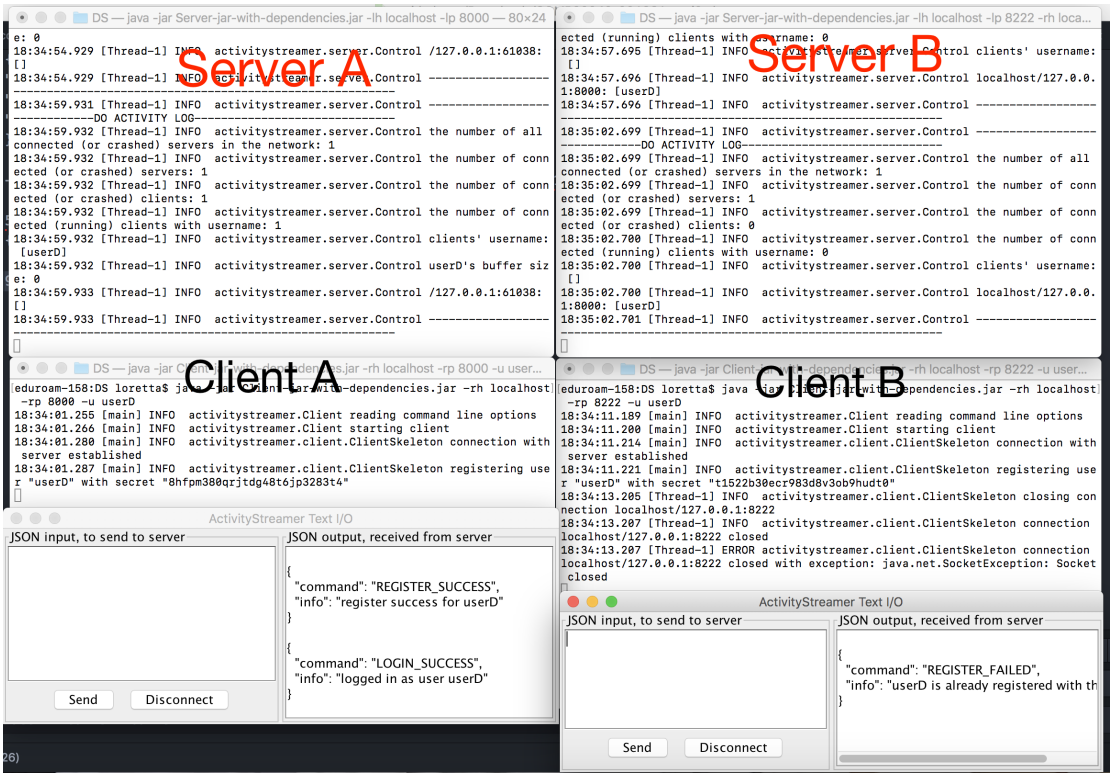

Figure 2-1. Server 2, use "userD", register failed.

- **Aim 4: guarantees that an activity message sent by a client reaches all clients that are connected to the network at the time that the message was sent**

Step 1:

We initiate a network with three server A, B, and C with server B connected to server A and server C connected to B. There exists a client A connected to server A, and a client C connected to server C. Then, we crashed server B. After the crash, client A send out a message.

Through our connection restoration mechanism, server C will automatically be reconnected to its grandparent server, Server A. After connection, Client C received the message sent by Client A.

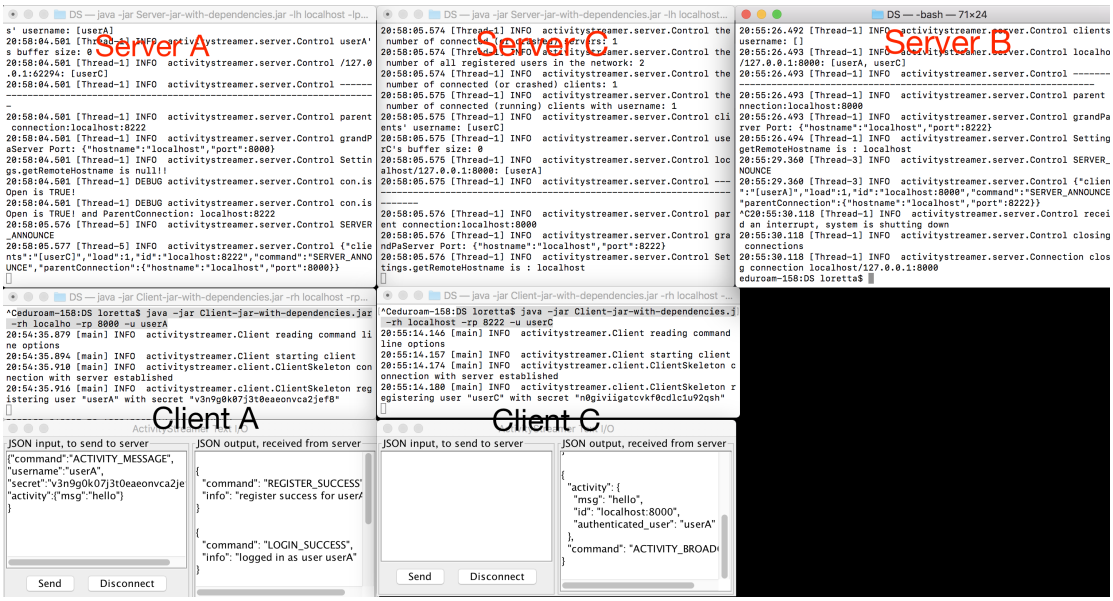

Figure 3-1. Reconnect. User B connect to A. B receives msg.

- **Aim 5: guarantees that all activity messages sent by a client are delivered in the same order at each receiving client**

Step 1:

We connect one client as client A to server A, so does client B to server B

Step 2:

Client A send multiple messages

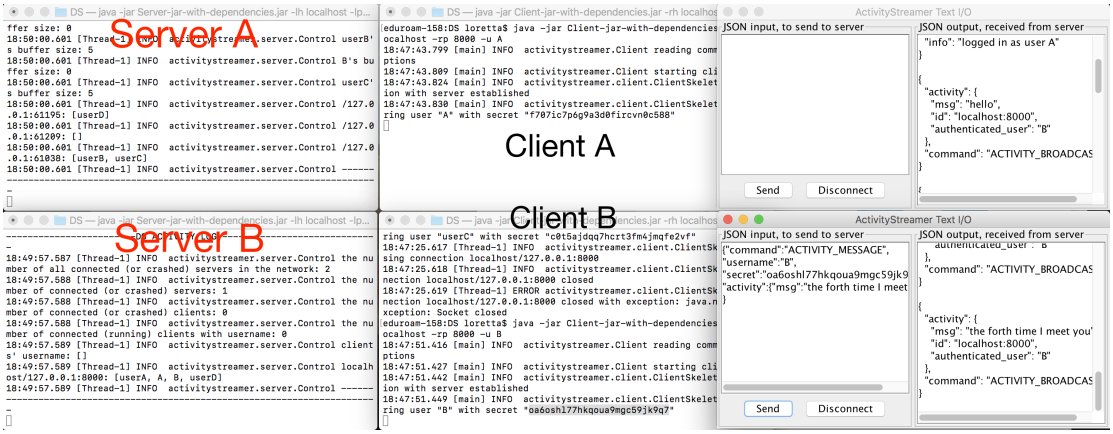

Figure 4-1. server A has client A, server B has client B. A sends several msg to B

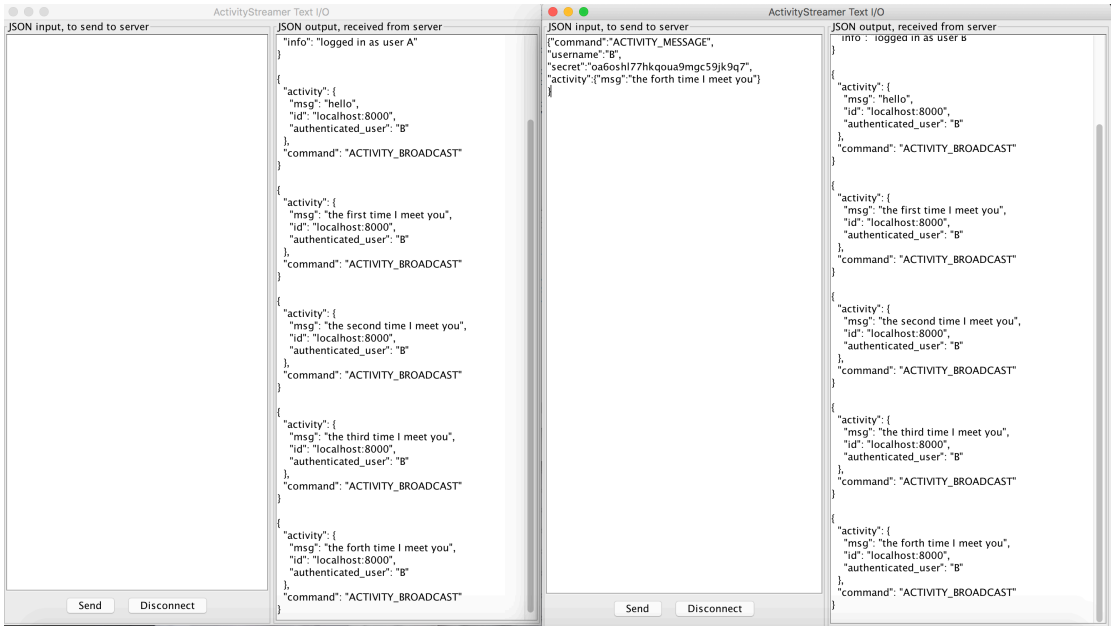

Figure 4-2. Details of message queue

- **Aim 6: Ensures that clients are evenly distributed over the servers as much as possible**

Step 1:

We connect 2 clients to server A.

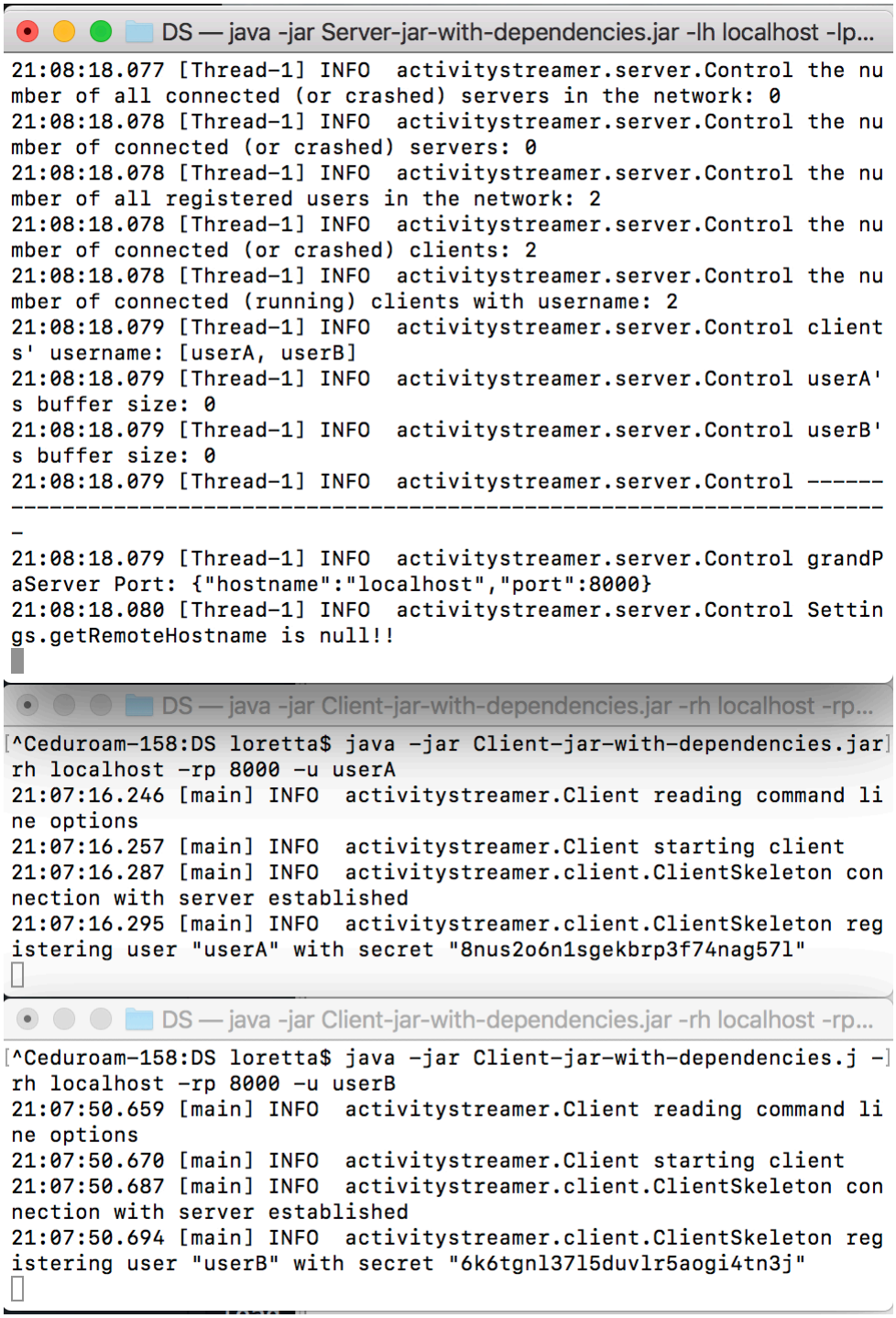

Figure 5-1. server A has 2 clients.

Step 2:

Introduce new server B into the system. As shown in Figure 5-2, userA has redirected to new server B, as userB still in server B, and their load are changed accordingly.

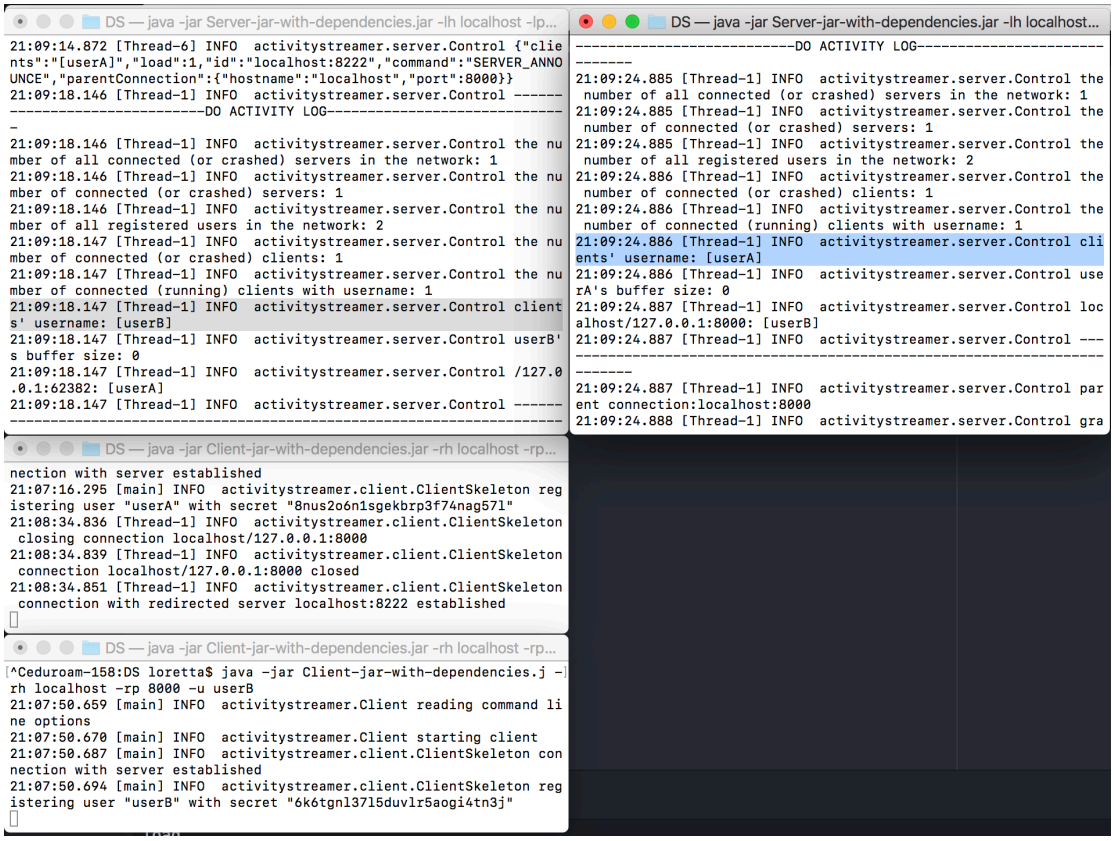

Figure 5-2. server B connect to A. redirect client. Check load.

- **Failure model 1: Server crash**

Step 1:

We connect server B,C and D to server A, as A is the root server for all others. All clients in the network can send and receive message smoothly.

Figure 6-1. A is the root server (before A crash)

Step 2:

Terminate A. Server B, C and D will form a new network after fail to reconnect to Server A several times. Server B will become the new root server, and their clients can send and receive message without known A has crashed.

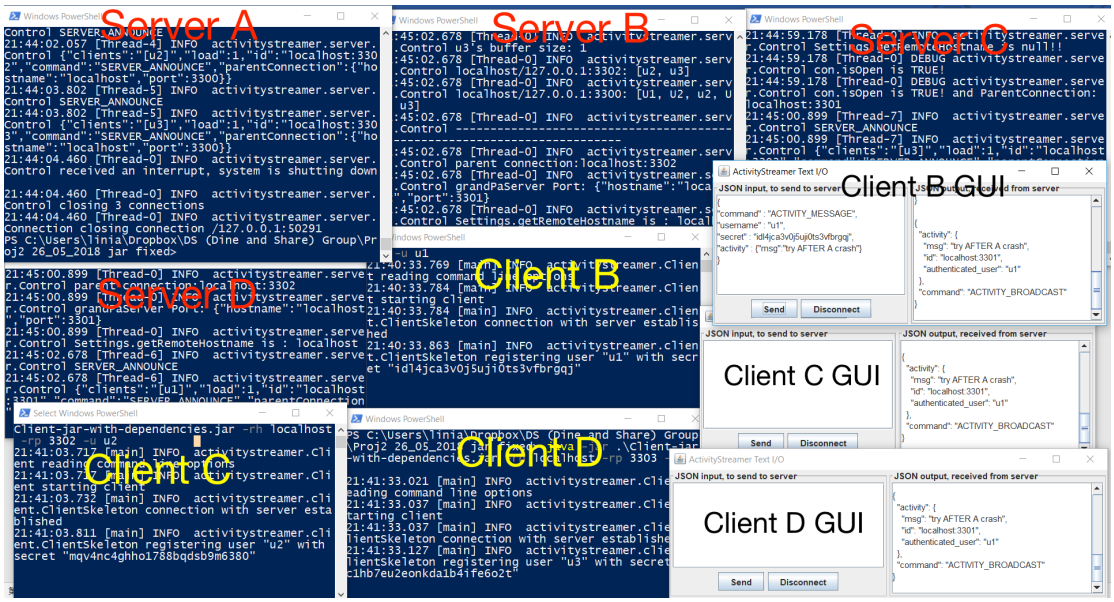

Figure 6-2. B is the new root server (after A crash)

- **Failure model 2: Client crash:**

Step 1:

We connect client A to server A, client B to server B. When Client A sends a message, Client B can receive it immediately.

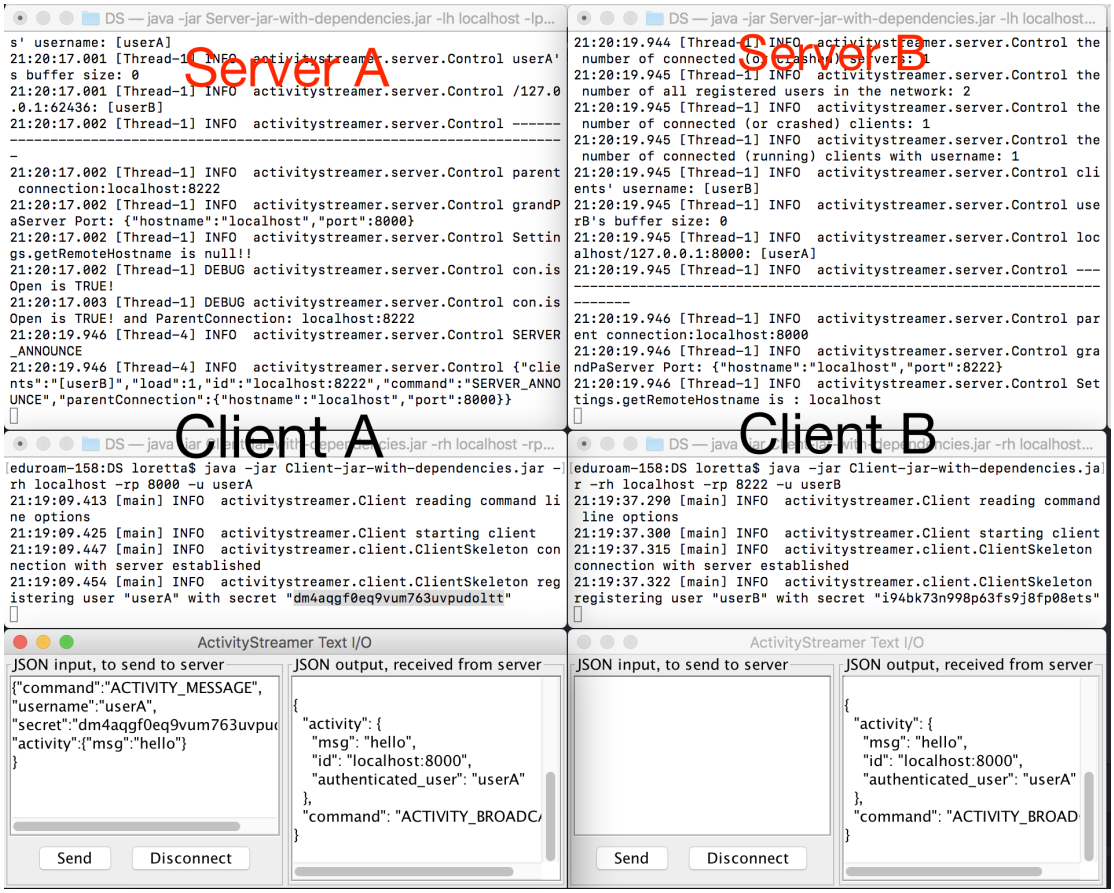

Figure 7-1. Client A and B in the network

Step 2:

Terminate client B. Then Client A sends out activity message. As user B doesn't intend to logout, the system will write current message into buffer. When user B login again, he could still receive the message Client A sent before.

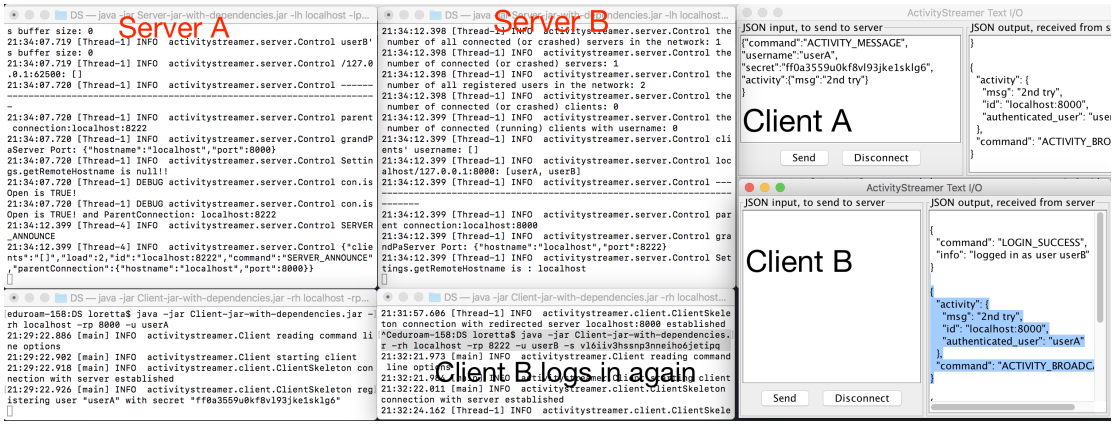

Figure 7-2. Client B login again to receive message