COMP90015 Project2- High Availability and Eventual Consistency

Group Name: Fantastic Four

Group 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 based on project 1, and implements high availability and eventual consistency with several enhancements in the presence of possible server failure and network partition.

**2. Outcome**

* Clients can join (register/login) and leave (logout) the network at any time
* Servers can join the network at any time
* Unique Register: a given username can only be registered once over the server network
* Message ensure: a message sent by a client can reach all clients that are connected to the network at the time
* Message order: all activity messages sent by a client are delivered in the same order at each receiving client
* Load balancing: clients are evenly distributed over the servers

**3. System Architecture**

As Figure 1 illustrates, the server contains 3 layers, namely application layer, data layer and network layer. Network layer is the communication layer for the whole system and is responsible for maintaining connections, sending/receiving data and delivering different types of message to different data consumer. The reconnection will be conducted in this layer and will not impact the other layers.

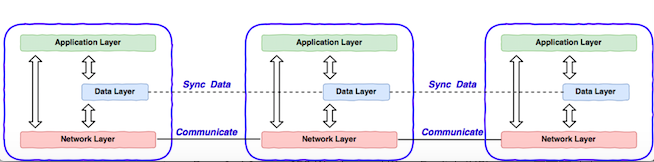


Figure 1

Data layer is used to store all local data and sync its data with other servers' data layer, which means there is a distributed database across the whole server system. Specific information is stored in tables. There are 3 tables designed in this project, as Figure 2 shows:

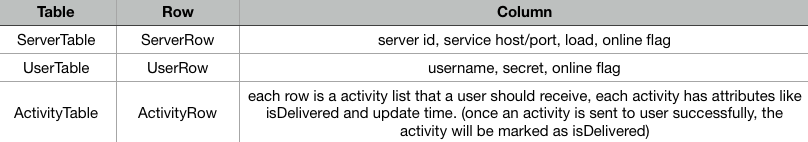


Figure 2

Application layer is a high-level layer, it can request network layer and data layer to execute their tasks respectively. Besides, application layer also needs to process activity message, authenticate, user login/logout and user register. Detailed communication diagram is as below:

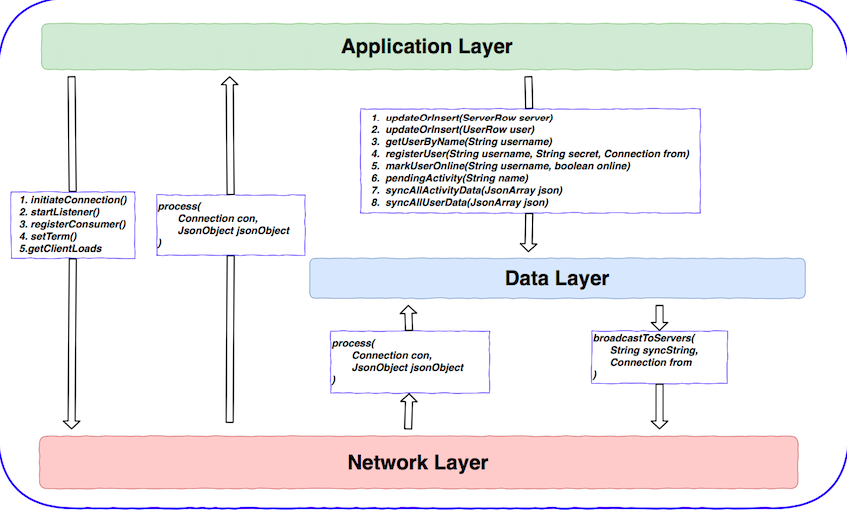


Figure 3

Specific messages in each layer are included in Appendix 1.

**4. High Availability**

The failure model suggests servers can crash at any time or network connections between servers and between servers and clients can be broken at any time, but failures are transient and broken network connections will eventually be fixed. Per CAP theorem, in network partition scenario, high availability and high consistency cannot achieve at the same time, system has to choose one between availability and consistency. This system focuses on availability so that once network partition occurs, each sub area of system can still work normally, clients are able to send messages and receive message.

Two mechanisms have been used to achieve high availability:

* Backup List

Each server has a backup list in its network layer. Backup list contains host and port information of neighbor servers. Backup list of a server is broadcasted to its neighbor servers. For example, if one server fails, other connected servers will try to reconnect to a server in backup list by order, until reconnection is successful. Thus, once reconnection is done, whole network will return a normal status.

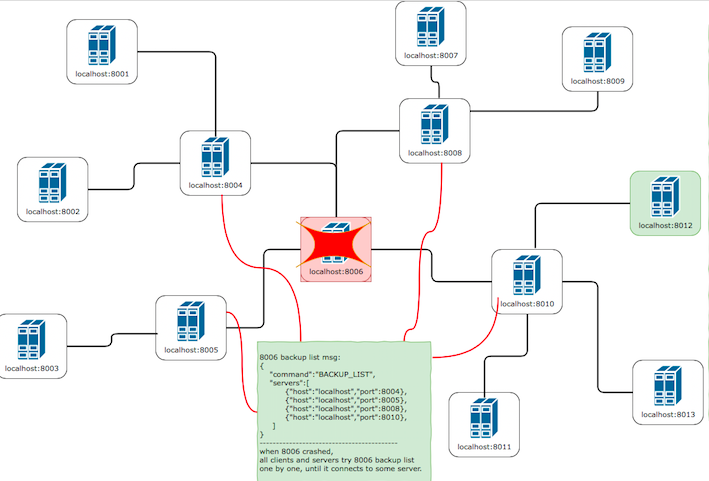
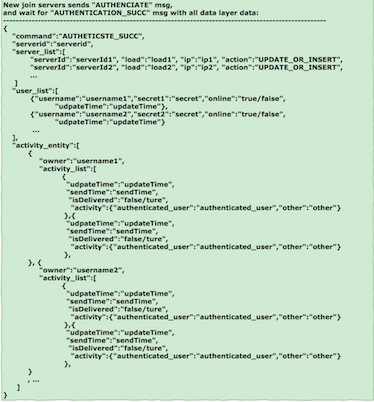


Figure 4

Per specification requirement, client part should be same as project 1. If a server crashed, all connected clients have to be redirected to a server in backup list manually.

* Full data copy of new join server

New join server has a full data copy with existing servers. Server authentication process has been improved by adding authentication success message “AUTHENTICATE\_SUCC”, when a new join server sends authenticate message to an existing server, the existing server will reply with authenticate success message if no error occurs in authenticate process. Authenticate success message includes all information in data layer ( ServerTable, UserTable, ActivityTable). Therefore, a new join server has a full copy data layer with other existing servers, and it can provide service to clients.

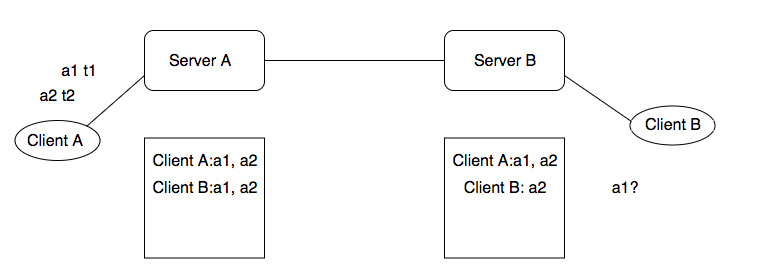


**5. Eventual Consistency**

This multi-server network achieved eventual consistency by synchronizing data layer in two ways:

* Data change synchronization: this is incremental synchronization. If there is any change in data layer, such as a row change in data tables, this change will be broadcasted to whole network.
* Periodical synchronization: this is a global synchronization, each server broadcasts its data layer information to whole network every 5 seconds, all tables in data layer will be synchronized.

This two synchronization mechanisms also can guarantee message delivery and message order, assuming scenario as below figure:



Client A sent activity 1 at time t1, activity 2 at time t2, Client B is supposed to receive activity1 and activity 2 by order. However, there is network failure that causes activity 1 hasn’t been transmitted to Client B. Now Client B only received activity 2, Server B’s data layer is inconsistent with Server A’s data layer. Periodical synchronization can solve this issue, Server B’s data layer will be synchronized at next period, activity 1 can be inserted before activity 2 per update time. Therefore, message sent by a client can eventually reach other clients by periodical synchronization, and message order can be ensured by “update time” in each activity.

If global synchronization period is less than server’s sending period (server sends receiving activity to its clients), message disorder issue can be resolved at next synchronization period.

**6. New Protocols**

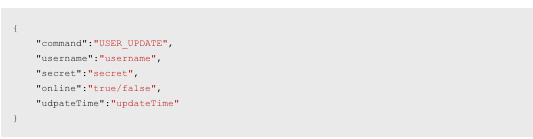
* SERVER\_ANNOUNCE

Add one attribute “action”, with the value “update” or “delete”. If a server crashed, other connected servers know this server failed and need to broadcast this failed information to whole network. In this scenario, “action” in announcement is “delete”, and only broadcast “serverId”, other fields can be ignored. In normal scenarios, “action” is “update”, announcement is same as project 1.



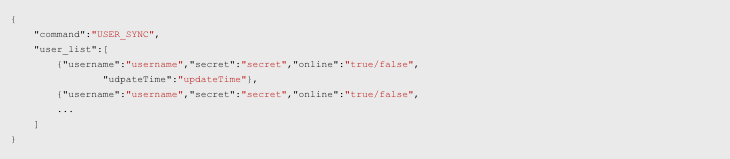
* USER\_UPDATE

This is used for incremental synchronization. Compare updateTime first and decide if it needs to update, if so, then compare a particular user information with local data layer and create/update accordingly.



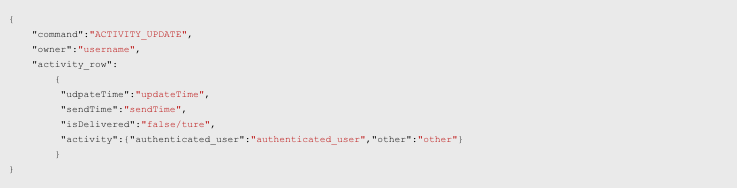
* USER\_SYNC

This is used for global synchronization. Compare updateTime first and decide if it needs to update, if so, then compare all users’ information with local data layer and update accordingly.



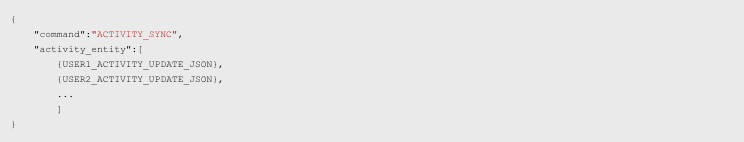
* ACTIVITY\_UPDATE

This is used for incremental synchronization. Compare updateTime first and decide if it needs to update, if so, then compare a particular activity information with local data layer and create/update accordingly.



* ACTIVITY\_SYNC

This is used for global synchronization. Compare updateTime first and decide if it needs to update, if so, then compare activity lists for all registered users with local data layer and update accordingly.



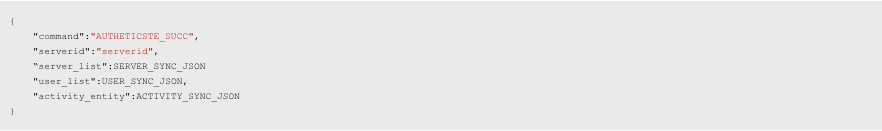
* AUTHENTICATE

Add “serverid”, “host” and “port”, when a new server is authenticating to an existing server, new server needs to tell its host and port so that backup list of the existing server can be formed.



* AUTHENTICATE\_SUCCESS

When a new join server authenticated successfully, it will receive AUTHENTICATE\_SUCCESS, which contains full information of data layer. Therefore, this new join server has same data cope with other existing servers and can provide service to clients.



* BACKUP\_LIST

Each server has a backup list in its network layer. Backup list contains host and port information of neighbor servers. Backup list of a server is broadcasted to its neighbor servers so that neighbor servers know how to reconnect next server once this server fails.

