COMP90015 Project 2
High Availability and Eventual Consistency

Thu Thao Le (thaol4@student.unimelb.edu.au)

Yicong Li (yicongl2@student.unimelb.edu.au)

1 Introduction

The goal of this project is to implement new server protocol to address the server failure issues. The server can provide availability and eventual consistency among the servers and clients when network partioning happens.

There are many challenges in this project. The hardest part is to rebuild the server protocol to handle the failures in the network. We have used tree protocol with some improvements.

The result:

2 Server failure

There are three properties in our distributed systems: Consistency, Availability and Network partitions tolerance. According to CAP theorem [1], we can only achieve at most two out of three properties. Since there are failures in the servers and the connections, we can only achieve either consistency or availability. If we choose availability, we can always return the messages even it is not up-to-date data. If we choose consistency, we have to update the latest data before returning the messages to clients. That means we can delay the sending process until all data is up-to-date or just return error when we cannot update the data.

There are three kinds of failures can happen in the system:

- Server crashes suddenly
- Client crashes suddenly
- Network is temporarily broken, then it can eventually be fixed.

The major problem in this project is how can we maximize the availability and consistency when network fails. There are three steps to handle the failure of the system [2]:

- Detect the beginning of the failure
- Limit some operations: There will be a trade-off between the consistency and availability. In this project, we focus on the availability during the partitions.
- Recover the network: After recorvering the failures, we can eventually reach the consistency.

3 Server topology

We use the tree structure to implement the server protocol in this project. However, we have some improvements to handle the network partitions. We set the hierarchy of the tree as in a family. As seen in the Fig. 1, server 1 is the root server, which has two children: server 2 and server 5. Server 3,4 and 6 are the grandchildren of server 1. If the servers have the same parent, they are sibling. For example, server 3 is the older sibling of server 4.

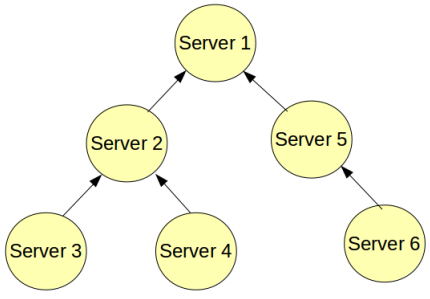

Figure 1: Tree topology

Fig. 2 and 3 describe two failure circumstances. In the first figure, server 2 crashes and then server 3 and 4 will connect to their grandparent, which is

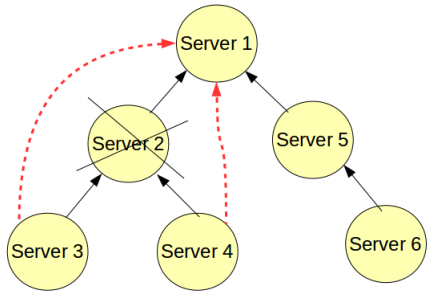

Figure 2: Parent server crashes, reconnect to root server

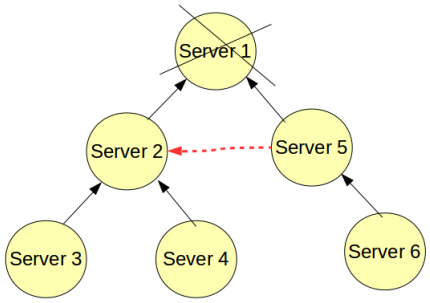

Figure 3: Root server crashes, reconnect to the older sibling

server 1. In the second situation as shown in Fig. 3, root server crashes and server 2 and server 5 have no grandparent; therefore, the older sibling, which is server 2 will become the root server and server 5 will connect to its older sibling.

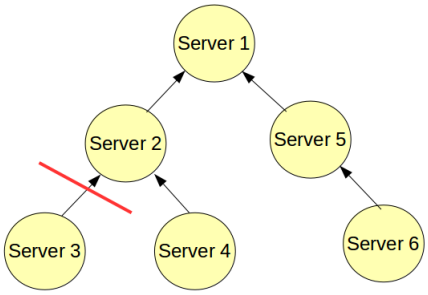

Figure 4: Broken network between S2 and S3

Another failure that can happen is the broken network due to many reasons (E.g. turn off the wifi, etc). In this situation, we handle by following the process.

As shown in Fig. 4, there is no connection between server 2 and server 3. We assumed that the network partitions is temporary and can eventually be fixed in less than 12 seconds. In this situation, server 3 will try to reconnect to server 2 in every 6 seconds. After two attempts of trying (exceeds 12s) and it still cannot connect to the server 2, server 3 will try to connect to its grandparent server, which is server 1. If server 2 does not have grandparent server, it will try to connect to its sibling server. If all attempts are failed, the connection of server 3 is closed and it will be removed out of the servers system.

4 Implementation

DESCRIBE MESSAGES

We implemented many functions in the server side to get the highest accuracy and consistency as possible, as well as an improvement of the first project. These functions will be described as follow.

The first improvement is that the servers can join the network after clients have joined. To fix this problem, we implemented a function to synchronize all user information from the remote server to the new server. For example, Fig 5 illustates a small network of two servers and one client. We can assume that Client 1 have joined the network before Server 3. When server 3 connects to server 1, server 1 will broadcast its user information from local storage to server 3. Therefore, if client 1 logout, it can login to server 3 again in the future.

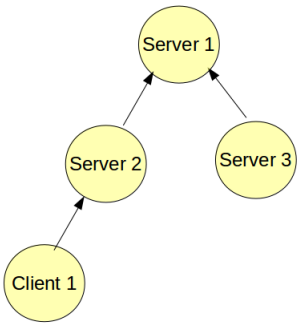

Figure 5: Example of a small network

Secondly, one of the hardest problem in our project is handling the registration process at any time. In the first project, if client is trying to register to one

server and one server crashes during the lock process, client cannot register to the network and the server will return exception. To deal with this issue, we come up with the idea that the server will response as soon as possible during the lock process without waiting for the network is fixed. An example of the process based on Fig 5 will be described as below:

- Client 1 send the register request to server 2
- During the lock process, server 3 crashes and cannot send the lock deny to server 1.
- Server 1 will know that server 3 has crashed, instead of waiting for the recovery, server 1 will response lock allow immediately to server 2. After that, server 2 responses register successfully to the client. However, server 1 has a cache to store the register information of the user.
- After the failure is fixed, server 1 will send the lock request to server 3 according to its cache. Server then send the lock deny message. In this case, server 1 will check if the secret of client 1 matches the secret in server 3, there is nothing to do. However, if the secret does not match, server 1 will broadcast a message to all other servers to notify that the username of client 1 should be deleted.

5 Availability and Eventual Consistency

In this project, we focus on the availability so every messages should be send as soon as possible without waiting for the state of network is recovered.

6 Conclusion

References

[1] Eric Brewer *Towards robust distributed systems.* Jan, 2000.

[2] Eric Brewer *CAP Twelve Years Later: How the "Rules" Have Changed.* Feb, 2012.