Distributed Systems Project2

Team: TMD
Team Member: Tao Wang (taow3 707458)
Mengda Zhang(mengdaz1 734524)
Danping Zeng (danpingz 777691)

1. Architectural Model

The architectural model of this multi-server chat system consists of three layers, that is ***Presentation layer, Service layer and Data layer.*** The presentation layer, in other words, is presented through many Client Applications or Server Manager Applications running on several virtual machines. It provides an interface for a user to access and use the whole chat system. With respect to the service layer, it is the middle layer between presentation and data layer, and also, it abstracts business logic and data access. Back to the architectural model designed for this project, a centralized authentication server was adopted in order to centralize external access to data and functions. In addition, internal implementation and changes of service layer are hidden to presentation layer. The data layer is basically a repository for persistently storing and managing collections of data, including not only database but also simple store types. For example, predefined "userDB" and "configt" are simple two files stored on the data layer server.

There are two kinds of communications existing in the system, namely, the layer-crossing and inside-layer communications. To be specific, communications between presentation layer and service layer are supported by Client-Server connection, which is a long connection. However, the Server-Server connection which enables the communications work between servers is a short connection type. Particularly, the central authenticated server is the only instance of service layer that communicates with the data layer. In regarding to the secure communication architecture, this model set up encrypted communications using the ***SSL TCP*** protocol. Details are specified by the diagram below.

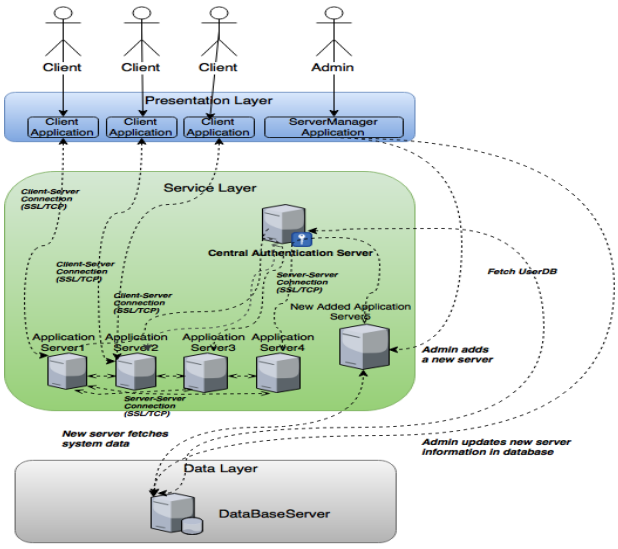

Diagram-1 System Architecture Diagram

2. Communications between components

2.1 Authentication

The authentication mechanism designed for this system involves three components that are the ***Client Application, Application Server*** and ***Central Authentication Server (CAS).*** As opposed to the project 1 where the client application will send "*newidentity*" request once the client connects to the server, the authentication mechanism in the project 2 requires clients to pass the authentication stage first, then they are allowed to send *"newidentity"* request to servers.

Shown as the figure ***Authentication-SD***, a client will be required to enter a <username, password> pair first. Then the client application is able to warp that pair to a JSON object and send it with "*LoginRequest"* message to the connected server. However, the application server is not able to authenticate the login request directly because the CAS is the only server that stores all predefined usernames and passwords, and it is designed to authenticate all login requests. Therefore, once an application server receives a "*LoginRequest"* message" from a client, it will send a *"LoginVerify"* message with the username and password pair to the CAS. The CAS then authenticates the request and send back a message with authentication result. Once the application server receives the authentication result, it will either send a message to inform the client application to send *"newidentity"* request if the username and password have been successfully authenticated, or inform the client with the denial of login message and close the connection, if the authentication fails. Sequence diagram for system authentication is displayed as follows:

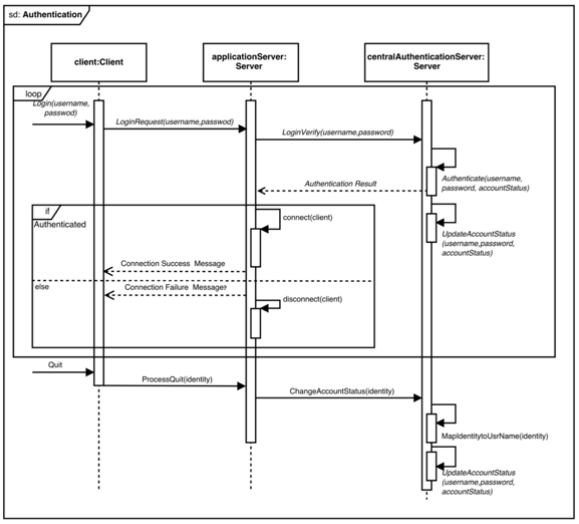

Diagram-2 Sequence Diagram of Authentication

2.2 Server Crash

This system handles server crash failure by using *"HeartBeat"* mechanism. To be specific, each server of the

cluster, including CAS, will send a heartbeat message to other servers between a constant time interval after all servers start. Also, each server will respond heartbeat signals, sent from other servers, with a declaration message about its alive status. There is no way to detect the crash except by set timeout, in other words, for example, if the server s1 does not receive s2's heartbeat response within the set up timeout, then s0 will be considered as a crashed server because it either stops sending alive status message or has crashed. Once s1 detects s2 crashed, s1 will update its data such as global room list. Particularity, if it is the CAS that detects a crashed server, CAS will change the login status of the usernames that belong to crashed server to be offline. The process is illustrated in detail by the following sequence diagram.

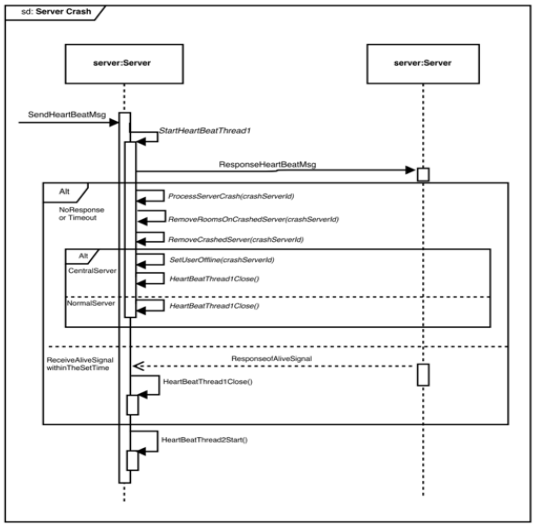

Diagram-3 Sequence Diagram of Server Crash

2.3 Scalability

Scalability is a new added feature which enables Admin to manually add a new server by entering four basic server construction parameters: (serverId, hostName, clientPort, serverPort) in a server constructor. The validness of those provided parameters will be checked and only if all parameters pass the verification, a new server instance can be created.

Once a new server, let's say sNew, is created, then sNew will broadcast an *Introduce* message with its own server information in the server network. Other servers who have received the *Introduce* message from sNew will update its global server list, by adding "sNew", and global room list, by adding "MainHall-sNew". After that, those servers will send a message to sNew with updated global system information. A sequence diagram for system scalability is presented below.

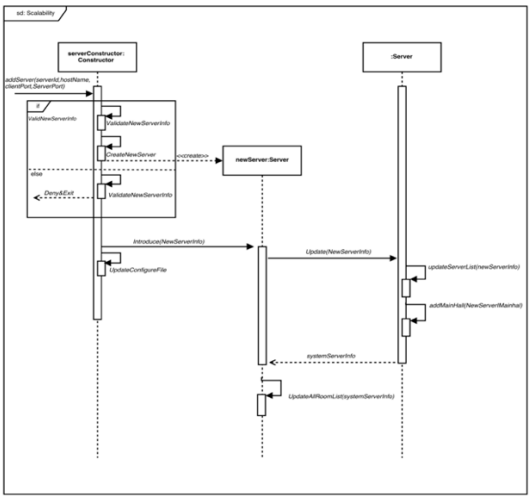

Diagram-4 Sequence Diagram of Scalability

Additionally, our system supports the capability of restarting a crashed server, however system is not able to back up the missing data.

3. Technologies Applied

Considering the scale of this project and development language, the Secure Socket Layer (SSL) protocol is the primary technology involved to solve the task.

The SSL is a cryptographic protocol, widely utilized to have communication secured between computers (including clients and servers) over a network which is not likely to be secure enough. The SSL protocol consists of two primary layers, which are handshake protocol, typically used to start a SSL session, and record protocol to encrypt the messages data exchanged in case of being eavesdropped, tampered or forged by malicious attackers.

In the practical application, by taking advantage of Java Secure Socket Extension, the fundamental mechanism has already been implemented by a number of packages, which enables to facilitate manual work of programmers to a great extent.

With respect to the handshake protocol, on the one hand, each server is to create a key store file to save its own certificate and private key as a preparation for establishing a SSL session. Specifically, this file is password protected in order to ensure the unique access by the server. On the other hand, there is a copy containing all reliable certificates authenticated by the application on each client end.

The rest of SSL connection work and to maintain the communication between the client and server is delivered to Java Secure Socket Extension to complete. In addition, from the perspective of programmers, the necessary code is similar to establishing a normal connection between client and server.

Therefore, by using the SSL protocol in Java development, it is not only able to secure the setup of TCP connection and guarantee secure communication, but also capable of simplifying the implementation process for the developers.

4. Team Responsibility

In task allocation, considering the system complexity and difficulty in applying the technology of SSL protocol from theory into practice, our team splits the aspect of security into two parts, which are establishing all connections by SSL TCP protocol and identity authentication.

In short, this project is divided into three parts. One is implementing SSL TCP connection established through sockets that is distributed to Danping Zeng. Identity authentication and failure handling are implemented by Tao Wang. Besides, Mengda Zhang is responsible for system scalability. With respect to debugging, Mengda Zhang is the main role, Tao Wang and Danping Zeng provide assistances in deuging SSL. In addition, Danping Zeng set up the cloud environment and all members tested the project deployed in cloud environment.