Distributed System Project 1 Report

Name: Shixun Liu
Login Name: shixunl
Student Number: 766799

1. Introduction

The report claims the consideration of five key challenges, namely Heterogeneity, Scalability, Failure Handling, Concurrency and Transparency, under the implementation of chat-room application.

2. Challenges

2.1 Heterogeneity

Heterogeneity in context of distributed system applies to the diverse in networks, computer hardware, OS, programming languages and even different implementation approaches from developers.

In term of network, TCP protocol is implemented to address potential issues as it is a widely used standard protocol, which is available in wired or wireless network, grantee the delivery of message without errors. However, the message sent as soon as the client change the wireless network may fail to arrive at server due to the break of TCP/IP connection between client and server. This issues is not fixed yet in this application.

Considering the operating system, I found abrupt disconnection is differently dealt in Mac OS and Windows system. The Mac OS does not throw IOException while Windows does, so I convert the IOException to Exception to make sure any abrupt disconnection happened in Mac OS can be captured successfully.

From programming language respect, the key point is to keep the message consistency in the system no matter what language is used. So JSON format can address this issue well even though different end is coded with different programming language

2.2 Scalability

The key issue in context of scalability of this application is that once the configuration file is fixed, no extra server can be added to the network. With the increase of client members, somehow, the application needs to add more servers in the future. So the next modification of this application is that if any other server joins in the network, it should send its IP and port information to other existing servers. Each server can remove or add server from or into its storage list.

2.3 Failure Handling

In this application, any failure is handled by try-catch structure in Java, mainly focus on the IOException. Especially when the client is closed abnormally, the system can catch the exception and process this issue as normal quit. However, there are two situations that will result in system break down. One is the application needs to run up all the servers per the configuration file, if partial servers do not run, the application will not operate normally; another issue is that during the operation, if any server shut down, the system will break.

The next development of this application is to make each server can detect any other servers' existing, once the message cannot be delivered to other server or the connection breaks, the current server should remove the remote server from its list. If any server breaks down during the operation, it needs to move all its clients to any other server to so that the client can keep connection to the application.

2.4 Concurrency

Each client connection or server connection is handled in different thread, so there is a possibility that server clients try to access the some sources at the same time and this will result in some conflicts. So here comes the concurrency issue.

Is this system, a typical situation is that several clients send out the message at some time and the server has to decide to send whose message first. The broadcast method in this system can be seen as a kind of resource, so I address the issue by synchronized the method so that each client will compete for capturing this method and get it randomly, this is mainly decided by the CPU. Once a client gets it, the others have to wait until the former one release it and compete for it again. In this case, the resource will accessed by only one thread at one moment. As shown below, one the client get the instance and recall its method, the other client needs to wait to get the instance later.

```
//Get the unique and same instance of class
public static synchronized ServerDatabase getInstance(){
    return instance;
}
```

However, there is a probability that one client cannot get the resource during each competition even though it sends message early than others. This will bring in a delay in client end especially there is a big amount of users.

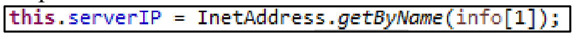
2.5 Transparency

Transparency in context of distributed system applies to the integrity of a collection of independents computers which is perceived as a whole that client will not recognize the inner components.

For access transparency, this system hides all the inner processes when a client access the server, such as the communication between servers for identity validation is not shown to client. Also, whenever a client access the resource in the system, such as #who and #list, it just need to type in the command, with no need to worry about how the system the store and process these information.

For concurrency transparency, it is obviously that each client does not know if there are others also attempt to access the same resources. However, as synchronized method is used to assign resource to each client request, so the client end may feel delay of its operation if network is under high load, so the concurrency transparency is not fully achieved in this application.

For location transparency, this application utilizes the InetAddress class in JAVA to make the client can refer to the server even through a name so that the server does not have to stay on the same device to keep its address valid.



```
this.serverIP = InetAddress.getByName(info[1]);
```

In this application, each server stores all its user info, other server info and also room info locally, no resource is replicated among different locations. The server communicates with others to share information. So the replication transparency is realized.