Personal Reflection

15071863

Ong Li Sheng

An explanation of your understanding any distributed system concepts that you have apply in your assignment.

I have learned a lot throughout this assignment. One of the few concepts that I learnt is heterogeneity. Heterogeneity states that the components in a distributed system should be interoperate across different operating systems, hardware architecture, programming languages and more. Our system should be able to work in different operating system.

I have learned that the architectural model is the structure of a system that specifies its components and their relationship between the components. Architecture model is important for making the system reliable, manageable adapter and cost-effective. Peer-to-peer architecture is used in this chat system. Peer-to-peer architecture states that all of the peer have similar responsibility and there is no client separation of responsibility such as in client and server architecture. In practice, all the participating peers should behave the same way and have same set of interfaces.

Failure model helps us to understand the effects of failure by describing the ways in how failure may happen. In the real world, the distributed system must be ready for any expected failure. The main reason of a process omission failure of a process is to crash. When we said the process crashed, meaning that the process has stopped working and will not execute any further steps. A process crash is called fail-stop if other processes can detect that the current process has crashed.

A description of how you applied the distributed concepts in your assignment.

Since we are building a Scala application, Scala application runs on a Java Virtual Machine(JVM), which makes our application able to work on different operating systems. Windows, macOS and Linux can run the chat program as long they have a JVM in their environment.

Peer-to-peer is used in this assignment as all the peers are only responsible for sending their message out and receiving messages that is directed towards them. The peers do not need to carry out the responsibility of other peers that are not related to them. In our peer-to-peer architecture, there is a supernode who is also a node, but has extra responsibility of handling new incoming nodes that are looking to join the conversation.

In our assignment, when the current supernode process crashes, the actor system of the application will be informed, and it will send out a notification to all the nodes connected. Then a new supernode will be assigned automatically alphabetically. So once the original supernode crashes, all the remaining node can continues chatting without the need of a supernode. When a new node wants to join, they can join using the new supernode address. The process of can be said as fail stop as the actor system is able to detect on the supernode crashes.

The problems encountered during this assignment and how you solved these problems.

Initially in our implementation, the Supernode will keep track of all the currently connected Nodes and Room. A separate RoomActor is created in the Supernode whenever a Node request to chat with another Node. The RoomActor is responsible to keep track of the chat room information such as messages and the users’ ActorRef. Every time it receives a message from the Supernode, it will broadcast to all the Node in the room. Hence, by doing this the Node have to go through the Supernode, where the Supernode then tell the RoomActor to broadcast the message. We realized this approach is not reliable and contains a lot of communication overhead. The main problem with this approach is that when the Supernode dies or crashes, because all the Node go through the Supernode to communicate with each other, the Nodes cannot communicate with each other anymore. This would mean that the whole chat system would fail as a whole as none of the Nodes are able to send message to each other.

Hence to solve this problem, we have change our approach towards this implementation, the Supernode and all the Nodes will be responsible to store the list of ActorRef. In that case, the Node do not need to communicate through the Supernode and the Supernode burden is lessen. The supernode is only responsible for relaying the current Nodes and incoming new Nodes. The Node now can send message directly to other Node since they also hold a local copy and do not need send messages through the Supernode. RoomActor is no longer needed to track the Node to chat with another Node. In conclusion, this implementation does not need to rely on the Supernode to pass the message. So even if the Supernode fails, the core functionality can still be carried on. The overhead of the system can also be reduced since they can communicate directly with each other.

An evaluation of the strengths and weaknesses of your submitted work.

Strength

* Heterogeneity
  + Our program works in different operating system because we are using JVM.
* Fault Tolerance
  + Reassignment of supernode
* Reliability
  + Each node has their own local copy
  + Each node can become a supernode

Weakness

* Data does not persist
  + There is no database hence the once the user quit, it cannot be traced back
* Security
  + There is no password required to join
  + No end-to-end encryption

Include each group’s member contribution percentage

Choong Kai Wern – 100%

Lim Shi Hern – 100%

Mah Qi Hao – 100%

Mu Chun Khang – 100%

Ong Li Sheng – 100%