Choong Kai Wern [Individual Personal Reflection]

Through the assignment I have learned a few key concepts of distributed system. The first one is related to the importance of the architectural model of a distributed system to produce a more reliable system. The architectural model of the system describes the high level representation of how each component communicate with each other. It is important to select the right architectural model for our system. By implementing a peer-to-peer architecture, we can ensure that our system does not have a single point of failure.

Another concept I have learned is related to the communication paradigm of a distributed system. I have learned about message passing. Message passing is when one process pass messages to another process. It is widely used in interprocess communication. However, in our system, through the use of Akka framework, we use the same concept to pass message to remote actors asynchronously to produce a reactive system.

One of the main problem we faced in this assignment is unstructured and redundant code base. The original implementation is to implement all the functionality into one Node actor, which involves in several tasks such as handling join, creating chat room and chatting with other user. This approach is not maintainable in long term.

One of the possible approach is to use State. However, since our architecture is peer to peer, this approach is less suitable. In the chat system, there is only two possible state, which is Initial and Joined. However just splitting into these 2 state still result in a huge chunk of code. At the same time, we have to handle disassociated event in both state, which might results in redundant code.

Hence we use another approach, inspired from one of the [Akka tutorial](https://doc.akka.io/docs/akka/1.3.1/scala/tutorial-chat-server.html). In this approach, we utilize Scala trait to have a separation of concern. Our Node actor consists of three abstract methods joinManangement, sessionManagement and chatManagement which handle different category of messages separately. Those methods are implemented in a separate trait.

The strength of the distributed system is as followed:

* Heterogeneity
  + Works in different operating systems, because of JVM
* Fault Tolerance
  + Automatically reassign supernode if supernode fail or disconnect
* Reliability
  + Each node has data backup

The weakness of the distributed system is as followed:

* Data does not persist
  + All the data in the chat system is stored in memory, which will be gone after the application is closed
* Security
  + No end-to-end encryption is involved to secure the messages
  + No password is required to join