Distributed System concepts and implementation

As we always defined, distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages. The Internet enables users throughout the world to access its services wherever they may be located. We have built a distributed chat system with such definition. The entire system is implemented through messages passing and without the uses of shared memory. As this system is distributed, each node will keep their own sets of data rather than accessing a centralized set of data.

Moreover, we have used peer-to-peer architecture where there is no server acting as a middleman. In this architecture, all of the processes involved in a task or activity play similar roles, interacting cooperatively as peers without any distinction between client and server processes or the computers on which they run. In practical terms, all participating processes run the same program and offer the same set of interfaces to each other. In our system, a peer will only need to join to the supernode to get the list of peers. After the list of peers is received, the peer can talk to all of the peers directly without the supernode.

Furthermore, our system is also fault tolerance because the system will be still working if the supernode crashed or disconnected. We are managed to solve this problem through the reassigned of supernode to another node and all the nodes will be updated. We have also used the disassociate event to manage the crashes of other nodes. If a node disconnected from the supernode, the supernode will process the disassociate event and remove name of the node from the system to avoid any new messages sent to the user or node.

On the other hand, heterogeneity is one of challenges of all distributed systems. As we know, heterogeneity means variety and difference of networks, computer hardware, operating systems, programming languages, implementations by different developers. In this system, we have used a middleware to map the heterogeneity of the underlying networks, hardware, operating systems and programming languages. The task of middleware is to provide a higher-level programming abstraction for the development of distributed systems and, through layering, to abstract over heterogeneity in the underlying infrastructure to promote interoperability and portability. Therefore, we have used Akka as our middleware of the system where every process is an individual actor. Each actor will have to communicate with each other through messages passing.

Although all of the necessary functions are implemented in our system, we are still obsessed in providing a system that ensure the quality of service. In this case, quality of service is measured through the performance, reliability, user interfaces and user experience of the system. We hoped that a system with good user interfaces and user experience can be developed besides other non-functional requirements. Therefore, we designed the interfaces nicely with responsive screen, intuitive user interactions and organized layout of elements.

One of the major challenges in creating the Graphical User Interface (GUI) is that the limited resources available online for scalafx. It is because the limited documentation for scalafx and limited number of blogs, forums and extra documentations. Hence, we are trying to search for javafx instead of scalafx because since scalafx is just wrapper for javafx and we are able to write scalafx according to javafx resources.

We also meet a problem in customizing the list cell of the list view of the chatroom. Since it is a chatroom, we hope to create an interface that is familiar to the user like all other chatrooms. When we are trying to achieve this, we ran into a problem which is the limited customization of default list cell. Therefore, we implemented our own custom list cell with custom font size, font family, padding, background colour, alignment, background radius, minimum width, and maximum width.

After we customized the list cell for chatroom, we found that changing the cell alignment to left and right is still insufficient for group chat because the sender is unable to be identified only through changing the cell alignment to left and right. Thus, we created another type of list cell that can show the sender of the message and use it in group chat.

Rather than sending single line messages, we are trying to send both single line and multi-line of messages. Meanwhile, we are trying to keep the "enter to send behaviour" for our text area. Therefore, we had customized the text area where it will add a newline while the key combination of shift and enter is pressed, and send the messages when enter is pressed.

Also, we also encountered a problem when we are trying to make the entire sy to be responsive because the components will be misaligned and shrink into inappropriate size. Therefore, we are making great efforts in modifying the layout settings and properties of every component to ensure responsiveness of the interfaces.

We think that user interface and user experiences are playing important roles in determining the successfulness of the entire system. As first, we have already designed the interfaces with the most basic layout and components. However, we decided to redesign and implement the entire system with a better design. We have even design the interface with the properties and layout of each components before the actual implementation of the interfaces. As we decided to majorly change the interfaces, we faced another challenge because every controller of the interface needed to be rewritten.

When we are nearly done with our system, we found one minor issue where the user cannot know when other people message him or her. Therefore, we implemented an unread function where the user can see the message that he or she received but have not read yet. Initially when implementing this function, we done it through refreshing the cells every time a user received a message and display it on the particular cell. However, we found it is very inefficient to refreshing the cells every time and practically a bad practice. Therefore, we decided add listener to every cell and they will always listen to the changes of the items they are displaying.

Strengths

* Heterogeneity
  + Works in different operating systems, because of JVM
* Fault Tolerance
  + Fault masking
* Reliability
  + Each node has data backup

Weaknesses

* Data does not persist
* Security