Haoyang Yuan
haoyangy
15-640/PIT
04/21/2016

Report for Project4

This report will describe the implementation of the failure-resistant voting system, it will describe the protocol between User and Server, the timeout threshold, the message lost handle and the recovery from failures.

1. Protocol between Server and User

For the protocol between Server and User, I am using a Message Class to handle all the communication issues. User or Server can easily construct the Message Instance by different constructor, and message type will instruct the Server or User on how to process the message.

```
public class ImgMessage implements Serializable {

    /* The information contained in the message class*/
    public int type = 1;
    public String filename = null;
    public String user = null;
    public LinkedList<String> imgele = new LinkedList<>();
    public byte[] img = null;
    public boolean vote = false;
```

2. Timeout threshold

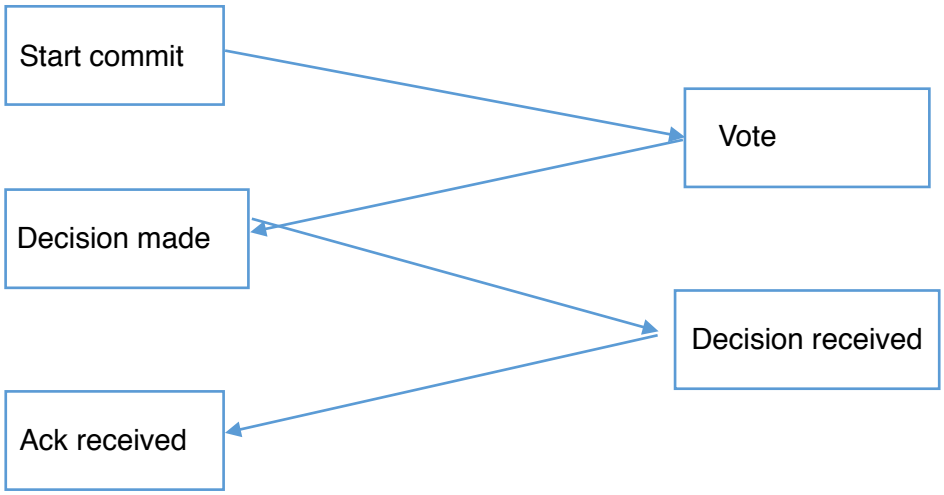
As mentioned in the write out, the message that is not lost is guaranteed to arrive within 3 seconds of being sent. So here I will use the timeout threshold as 3s to process. The details of the threshold work will be described in the next part.

3. Message lost handle

Since the message between the User and the Server will be lost. We will use the time out and resend scheme to handle the failures. And since the Server is the one aggregates all the process information, it will be responsible for the time out count. That is, for the Server, when the vote request has been sent for 3s and there is no response, it will assume the voting result is No and distribute the decision. For the decision distribution, if the there is no Ack response in 3s after distribution, the Server will have to distribute the decision again to the missing user until all Ack are received. This scheme will guarantee that there is no conflict, even though the unnecessary failure may happen, which is the at most once scheme.

4. Recover from failure

The way to help recovery from failure is using log to store the current status. I will set up 3 checkpoints for a collage in the Server, and 2 checkpoints for a collage in the User. Whenever the checkpoint is reached, a new entry will be added to the log, and the sync call will store the log to permanent status.



The Server will have 3 check points:
Start Commit: when the new commit command is received, the server will log the collage name as the task, and log down all the related user nodes. When ever the server recoveries, all collages in this status will be regarded as false decision, and the server will distribute false.
Decision Made: when the decision on the server side has been made, (it will write down the collage image if it is true) and the decision will be logged. When the server recoveries, the collage tasks in this status will distribute the decision again to continue the Ack process.
Ack Received: when all Ack of the collage task have been received, the server will log this task as finished. So, when recovery, collage task in this status will be ignored, it will not be added to the task queue for any process.

The User will have 2 check points:
Vote: When the decision for certain collage has been made, this event and decision will be logged. Whenever the User recovers, it will lock the file if the decision is true just like before it breaks down.
Decision received: When the final decision from server has been received, the User will need to log the final decision. When it recovers, it will delete the file or unlock the file according to corresponding decision (if not before breaking down).