CSCI 6510 Distributed Systems and Algorithms
Project 2 Write-up
Wei Liu and Chang Sun

Twitter Service using Paxos Algorithm

Data Structures Design

In this project, we used python implementing Paxos Algorithm. In the design of primary data structure, we used list of dictionaries as log, 2d list as block matrix and list as timeline. We employed python package "pickle" to store local information so it is easy for one site to recover and read its previous log entries.

There are three types of operation events, tweet, block and unblock. Each event is a dictionary containing site id, operation name, operation content, operation time and the id number in log. Each event would be stored in the log which is a list of dictionaries. When one site blocked another site, the related position in the block matrix would be 1, otherwise it would be 0 by default.

We need set each site's id and IP address in the configuration txt file. Before one site "log in" the service, it requires user to input the corresponding user id.

Communication Protocol

In this project, TCP is applied on the establishing of the connection between each site. All sites are both proposer and acceptor at the same time, which means they will connect to themselves as clients when they listen to others as servers.

Synod Algorithm

The key part of this project is to implement Synod Algorithm. We have five attributes of the site class for synod algorithm: prepare response counter, ack response counter, the list recording the value of promise, the list recording the prepareNum, paxos variable

which is set as default (0,0,None) with the sequence of max prepare number, accNum and accVal.
When a proposer in the phase of prepare, it would choose prepare number (n) as following formula：

$$n = m * N + id$$

where n is the prepare number, m is the number of rounds it has propose in this synod instance (initially 0), N is the number of total sites (5 in this project) and id is the user id of it self (1, 2, 3, 4 and 5 respectively).

Leader election is simplified in this project. Site who won the K-1 log entry can propose the K log entry as a leader. If the site is the leader, it will skip the prepare phase. The site would wait 0.4 second (long enough) to receive the response. It will store the response in the list of ack_response. If more than two sites send back response, the leader would send commit to all acceptors. If there are not enough sites sending back response, it will start full synod algorithm.

If the site is not the leader, we would execute full synod algorithm. If we did not receive the enough response, we would increase the prepare number and try again after 30s. After trying more than 5 times, we would give up this proposal. If we receive enough promise, the site will choose value with largest accNum number and the site would send accept information to the acceptors. Then acceptors would send ack after saving accNum and accVal to the proposer. If the proposer receive ack from majority sites, The proposer would send commit to acceptors so the acceptors would write the event to the log. After one site complete the synod process and broadcast one event successfully, the log recording this event would be written in all online sites' logs. In this way, we have an agreement among all sites.

Learning mechanism

When one site recover, it will automatically learn missing log entries from others. First it will broadcast "ask" to request the max log id of other site's log. Then it will know how many logs it missed. It will execute synod algorithm and propose dummy value, and tell other site which log id it want to update.