Distributed Systems and Algorithms
Stacy Patterson
December 3, 2017

Distributed Twitter Service Using Paxos Algorithm

Ziwei Dai & Lixin Chen

1. Implementation.

We designed two classes: a) TweetClass, b) Site. We have two helper function: (bool)sortByTime() and printTweet(). For recovery, we have one files"log.txt". In the Site class, which is our basic class, we define each site acts as an acceptors as well as a proposer. We numbered each sending messages (prepare is 0, promise is 1, accept is 2, ack is 3, commit is 4, recovery is 5) to determine the type of each messages then call the related function. To achieve the desired features we have the following major functions:

- Site operation

tweet() & block() & unblock() - When we tweet a message or block (unblock) someone, we will create a new value and call sendPrepare() (a private helper function).

view() - sort the local tweets vector to have the right timeline and call printTweet() to show all tweets.

recovery() - call prepare() many times until this site have the same log entries as the site which have max number of log entries.

- Helper function

sendPrepare() - will determine whether this send site is leader or not. If it's leader, it will call leaderAccept(). Otherwise, it will call prepare().

- Act as a proposer

prepare() - choose a new proposal number (we use wall clock time to be the proposal number, a long). Then it will send prepare message to every active sites (including itself). After that, it will create a thread (checkPromise()) to check the response from majority in 5 seconds. If didn't hear from majority, it will start over and call sendPrepare() again.

accept() - count received promise message for each log entry, if it hears response from majority, it will choose an accept value and send accept message to all active sites (including itself). After that, it will create a thread (checkAck()) to check the response from majority in 10 seconds. If didn't hear from majority, it will print a message and not start over.

leaderAccept() - send accept message to all active sites and create a thread to check the response.

commit() - count received ack message for each log entry, if it hears response from majority, it will send commit message to all active sites (including itself).

• Act as an acceptor

recv_prepare() - when this site receive prepare from other sites, if n > maxPrepare, it will renew the maxPrepare number and call promise() to send promise. In case that the site need previous log entry's accepted information (when recovery), we use three vectors to store each log entry's accNum, accVal and maxPrepare.

recv_accept() - when this site receive accept, if n >= maxPrepare, it will store the accNum and accVal and call ack() to send ack message.

recv_commit() - when this site receive commit, it will write the value to log file and call checkInfo() to upgrade the two in-memory data structure: a vector of tweets and a set of blockInfo.

2. About Recovery

When one site crashes, it can recover from its own log file (may missing new log entry) or from nothing (lost its own log file). Our design has two part: 1) recover from stable storage: read its own log file and use checkInfo() to store tweet, block and unblock information in in-memory data structure; 2) after part1, send its own count of log entries to other active sites and if it receive the number of log entries bigger than its own number, it will choose a max number passed to recovery().

3. Socket Implementation

For the socket part, we use select() to receive messages from different socket number. Each site is a server and a client. For the server part, we have two threads: the first one is the ReadInput thread which is used to read the input from the user on keyboard, the second one is used to receive the data sent by its client. As a client, we use a thread to connect each of its server(including itself). Therefore, if the total number of user is 5. We will have the main thread plus 2+5 child threads for each site. We have a map which is used to relate the userid and the socket descriptor so that sites recovery will not be a problem for communication.

We have a helper function called Recvmsg() which is used to figure out the relationship between the messages from the socket and the operation we should do for our algorithm.

For recovery, when the sites recovers, it will tries to connect all the active servers and sent its number of log entries to them. As a server, when it receives the recovery messages from its clients, it will sent back its own number of log entries. In this case, every site will know the number of log entries of any other active sites so that the ones who need to recover will know the number of times it need to do the Synod Algorithm.