Distributed Systems and Algorithms Project 1
Stacy Patterson
October 15, 2017

# Distributed Twitter Service

Ziwei Dai & Lixin Chen

## 1. Implementation.

We used the C++ language to do this project. We designed four classes: a) TweetClass, b) EventRecord, c) Message, d) Client. Also we have three helper function: (bool)hasRec(), (bool)sortByTime() and printTweet(). For recovery, we have four files "log.txt", "dictionary.txt", "timeMatrix.txt" and "tweet.txt". In the Client class, which is our basic class, we have the following major functions:

tweet() - create a TweetClass object and put this object into the local tweets vector. Tick the counter and update the local matrix clock. Create an EventRecord object to store the tweet events' information. Put this EventRecord to the PartialLog and write the change to the files.

block() - insert the pair to the Dictionary set, if insert success, tick the counter and update the local matrix clock and create an EventRecord object to store the block events' information. Write the change to the file on the disk. Otherwise, you already blocked this follower, do nothing.

unblock() - check the Dictionary, if didn't find the pair, this means that you didn't block the follower and do nothing. Otherwise, erase the pair of the Dictionary set, tick the counter and update the local matrix clock, create an EventRecord object to store the block events' information. Write the change to the file on the disk.

sendMsg() - check the Dictionary, if the receive end is not blocked by this site, create the NP following Wuu-Bernstein algorithm. Create a Message object and using msg2string() convert this object to a char * which can be sent to. Send the char *.

receive() - using string2msg() to convert the received char* to a Message object (NP and matrix clock) and create NE and using Wuu-Bernstein log-dictionary algorithm to update the Dictionary, matrix clock and PartialLog. Also write these change to the file on the disk.

view() - sort the local tweets vector to have the right timeline and check the dictionary for each tweet. If the creator of the tweet block the user of this site, we will not print the tweet.

**2. Duplicate Insert**

To handle this situation, we modify the algorithm to check every block(insert) and unblock(delete) events in the NE. When the event is block, we add the corresponding entry to the local Dictionary. When the event is unblock, we erase the corresponding entry of the local Dictionary. Therefore, in the case of block->unblock->block, we add the entry, then erase the entry and then add the entry again so that we will finally add an entry to the Dictionary.

We treat multiple continuous same operations as one operation. If A block(B) and block(B) again, we will treat A only blocks B once. The local event counter will also increase only once. Therefore, this change still solves log and dictionary because duplicate insets or deletes will be treated as invalid command so that they will not change anything.

**3. Socket() implementation**

In order to make one computer is a server and a client at the same time. We use multi-threading programming to do this. We totally have 3+n threads running in our program, where n is the number of other clients. The use of each threads is listed below:

1. Connection thread for server to wait for different clients to connect. Using accept() function.
2. Data getting thread for server to receive all the message sending from its clients. We use select() function here to improve the robustness and avoid message confusion.
3. Input Reading thread to read the input from the command line in the terminal as an mini UI. As is requested, our program could excite four operations: tweet, view, block and unblock.
4. Client thread. Each computer is a server and a client to each other clients. Therefore, in order to connect with different servers, we run different threads to connect with different other servers.

Our application could tolerate crash-failures and recoveries no matter how many sites crashes (even all!). The method for the server to bind the same address immediately if it crashes is using the setsocketopt (s, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on)) to enable the address to be reused. As for each computer, once the server is build again, it will automatically try to connect with the other servers. Our application could be executed on multiple virtual machines in different regions. We use public IPV4 to connect and we solved the timing order issue by using UTC time.