David Chen & Hannah Deen
October 15, 2017
Distributed Systems & Algorithms

Project 1 Description

Dependencies:

To be able to implement and maintain site stable storage, the Jackson Project, the standard JSON library for Java, was used to serialize the logs, dictionaries, and configuration files.

The Design:

For the solution to the Log & Dictionary Twitter problem java code base was created. The repository includes the network, the Wuu-Bernstein algorithm, and site stable storage.

Beginning with the network implementation, the server client pair are created together but functions individually. For N number of servers and clients (the number of ip addresses and ports dictated/available), each client has its own 'home' server. When the client it told to Tweet (send message) it first adds the tweet to its own log, detaches from its 'home' server by closing the server and in/out stream connections. The client then loops through the non-blocked servers of all other sites Tweeting, then reattaches to its home server to receive other tweets. When a message is sent to a site, it is sent to the server. The server then writes to the clients log, which the client receives once its reattached to its home server. Since clients can only be attached to one server at a time, but servers can have multiple clients this assures that all tweets can be delivered without deadlock. This implementation also takes advantage that a server client pair does not have to be permeant, but can be reliable. To handle the servers and clients being independent the implementation used threads to create a multi-functioning Twitter base ie the server and client are run together, but are able to function independently.

The algorithms implementation begins when a user calls Tweet, view, block, and unblock. Each of these events are denoted as an 'event' object that is stored within a Site object. Each machine has one site object which contains the id, log, blockTable(dictionary), and timeTable. We also created Event objects which can have three types (0,1,2) which specify tweet, block, and unblock events respectively. If an event is a tweet event it contains a Tweet object which the message being sent. If it's a block or unblock event, a field called blockee specified the target of the action.

1. Beginning with Tweet, the user enters their Tweet then the sites first increments its Lamport clock, creates the Tweet event, then stores the event in its own log. Then each site is looped through, skipping blocked sites and itself, for each of the sites. The sender site then goes through its log and find whether or not the receiving site has received its logs and other sites logs (primary and secondary information). The sender site then truncates its log to only send what it knows the receiver does not know. The sender site then connects and sends the Tweet and the partial log.

2. When a user decides to block another user, the site first increments its Lamport clock, creates the block event, then add the event to its own log. The block is then added to the block table. This is indexed by site ids and the blockee id. If a site 2 is blocked by site 3 blockTable[3][2] is then set to 1. The table is local to each site and other sites are only updated when a process receives a Tweet event. The sites then update their own block tables to make sure they only send information to sites of unblocked sites.
3. The unblock event is implemented the same way expect when process i unblocks process j, blockTable[i][j] = 0;
4. For view functionality, the site parses through its log and only displays to the console unblocked users sorted by the site's local time. The newest Tweet is listed first, newest is defined by UTC of the process that created the tweet.

In our implementation we created a Message object which contains a tweet, the timetable of the process, and the partial log. When a Message is received by site j from another site i, site j first goes through the partial log and eliminate events that it's already knows about (based on Lamport timestamps in its own timetable). The site then loops through the events looking for block and unblock events. The site then updates its Lamport time table with the direct and indirect knowledge it gained from the message. The received partial log of events is then added to the sites own log. Once the site knows that everyone knows of a certain event that event is truncated out of the log (ie. when hasRec is true for all processes).

To make stable storage a reality, every time and event is recorded to a log locally a object mapper is created. From here sterilization is enabled and a new file is created for the site. If the file already exists, then the old one is overridden. The mapper then writes the entire log, from first event to most current, to the file. If a site crashes this file is then loaded to reinitialize the site.

Handling Duplicate Inserts:

To handle duplicate inserts (ie block, unblock, block), the implementation relies on an adjacency table where 1 means blocked and 0 means unblocked. When a partial log is received that contains multiple block and unblock events for the same pair of processes (i, j), the sum of the blocks and unblocks are tallied along with the current state blockTable[i][j]. If the resulting value is greater than or equal 1, then process j is blocked by i (blockTable[i][j] =1), else it's unblocked (blockTable[i][j] =0). This ensures that you cannot block a process that's already blocked and cannot unblock a process that's unblocked. In addition, a process cannot block an already blocked process and cannot unblock an already unblocked process. No new events are created for errors such as these.

Maintaining the Solution to the Log & Dictionary Problem:

Find an algorithm for maintaining the dictionary such that, given an execution $< E, \rightarrow >$, for every event e:

x ϵ V(e) iff Cx $\rightarrow$ e and there does not exist an x-delete event g, such that g $\rightarrow$ e.

We modified the restrictions to the dictionary problem by allowing multiple inserts and deletes, but our implementation prevents an unblock event from being created if a process is not already blocked. As such a block event always casually proceeds an unblock event ensure the property holds.

Additional explanation:
Every site has a personal copy of the log. Messages are only being sent when the sender is (1) tweeting and (2) know that the site it is sending to has not received the specified log and if it has will truncate its (sender's) own log. The sharing of primary and secondary knowledge among logs ensures that even when a user is 'blocked' that the 'blockers' logs are updated, but NOT displayed (as due to assignment specifications). No excess knowledge is being sent, and if there is Tweet (message) loss from broken channels or a site shutting down and/or crashing. Site stable storage via JSON serialization has been implemented. Once a site is back up running, it can continue to participate since it's not part of any casual chain and will learn about the events it needs as it receives messages.

References:

http://www.cse.scu.edu/~jholliday/COEN317S05/GuptaSlides.ppt
https://dl.acm.org/citation.cfm?id=806750
http://www.cs.cmu.edu/~tcortina/activate/java/jar.html
https://github.com/FasterXML/jackson
https://coderanch.com/t/624637/java/switching-connection-servers
Many Stackoverflow Posts