CSCI 6510 - Distributed Systems and Algorithms Project 1

Team: Lili Li, Yi Liu

Programming Language:

Python 2.7.6

Implementation Details:

In this project, we created Event class that contains information of an event. The attributes include *op* (operation type), which is either "*tweet*", "*block*" or "*unblock*", *time* which is the local clock time, *node*, which is the *server ID*, *content* and the *UTC time*. *Content* could vary depends on the *op*. With operation type to be "tweet", *content* would be the actual tweet message, with the *op* to be "block" and "unblock", *content* would be the follower ID that is blocked/unblocked by current server.

Global Variables:
We defined *blockInformation*, *PL* (Partial Log), *clock* (local clock time), *tweetInformation, T* (timestamps) as global variables.

Threads:
In server.py, we created two threads: local thread and remote thread

1. In *LocalThread*, we implemented local operations: *view, block, unblock.*
2. In *RemoteThread*, we handled messages sent from other sites. The system will only send message to other sites when the current site has the operation "tweet". *RemoreThread* also listens on message sent from other site, After receiving messages, *PL*, *T*, *blockInformation*, *tweetInformation* will be updated correspondingly. *NE* will be created after we deserialized the *NP* message from other sites.

Functions:

1. *View()* : In view function, we display all tweets sorted by local time on current machine. Tweets will be read from *tweetInformation.*
2. *Block(NodeId, UserId)*: In block function, we added the *pair(NodeId, UserId)* into block information and increased local timestamp. Block event will be created in the view function and added into *PL*.
3. *Unblock(NodeId, UserId)*: The *pair(NodeId, UserId)* will be removed from *blockInformation* in *unblock* method. In the meanwhile, *unblock* event will be created and added into *PL*.
4. *Update_T()*: Update *T(Timestamp)* based on Wuu-Bernstein algorithm.

5. *Update_blockInformation(NE): blockInformation* will add or delete *pair(NodeId, UserId)* based on whether there's a single insertion or deletion record in partial log. If there's multiple insertion pairs for same user, the *blockInformation* will only add the one that causally happens after the same pairs.
6. *Update_PL(NE)*: *PL* will be truncated based on *NE* and comparing *T*.

Other Functionalities:

File recovery:

Whenever there's an update in *PL*, *blockInformation*, *tweetInformation* or *T* for current site, our program will dump these fields to local disk using python pickle. As the system executes, it will load all the global fields *blockInformation*, *PL* (Partial Log), *clock* (local clock time), *tweetInformation, T* from local disk if there exists a record for the current site. If not, these fields will be initialized as empty sets accordingly.

Multiple insertions:

As described in project instruction, multiple insertions could happen in this application. For example, if a user execute [block "*serverID*"] and immediately execute [block "*same-serverID*"], this action cannot be completed since this (user, *serverID*) pair has already existed in *blockInformation* and a warning message will be given.
A user could also execute [block "*serverID*"], [unblock "*same-serverID*"], [block "*same-serverID*"] multiple times before this server send a tweet and push these information via partial log. To handle this case, besides comparing timestamps across different sites, it's also important to define "unique" insertions and add needed ones. The insertions and deletions will all be sent to other sites regardless of the situation whether it's inserting the same action. However, upon receipt on the other site, the user will check if there exist pairs of insertion and deletion for same event (block and unblock the same user), and only insert the event of blocking that same user that causally happens after that pairs. This functionality is achieved in *update_blockInformation*() function.

Deployment on AWS:

Three instances were setup in AWS. Each of the instances was created in different regions: US EAST (N. Virginia), US WEST (N. California), ASIA PACIFIC (Tokyo).
We assigned these three instances with port numbers: 8081, 8082 and 8083, these port numbers will be combined with the public IP of each instance in the system.