Distributed Systems and Algorithms CSCI-4510/6510 – Fall 2017
Project 1

Assigned: Sep. 19, 2017
Teams formed by: Sep. 26, 2017 (10% penalty for missing this deadline). Work in groups of two.
Project due: Sunday, Oct. 15, 2017 at 11:00pm
Demos: Oct. 16 – Oct. 20, 2017

Description

For this project, you will implement a simplified distributed Twitter service using the Wuu-Bernstein algorithms for replicated logs and dictionaries.

The distributed system consists of N sites, each site corresponding to a single user. Every user "follows" all users. Initially, a follower should see all tweets from the users he or she follows. A user can "block" a follower; when that follower is blocked, the follower should not be able to view any of the user's tweets (old or new). A user can also "unblock" a blocked follower, which once again allows the follower to view all of the user's tweets.

A tweet is a tuple, consisting of the following fields:

- User: creator of the tweet
- Message: the text of the tweet
- Time: from the physical clock of the node where the tweet was created (you must account for different time zones).

Each user can execute the following operations:

1. `tweet <message>` : creates a new tweet
2. view : displays the timeline, i.e., the entire set of tweets (including all fields), sorted in descending order by the time field (most recent tweets appear first), excluding tweets that the user is blocked from seeing.
3. `block <username>` : blocks the specified username from viewing tweets of the user who issued the command.
4. `unblock <username>` : unblocks the specified username; allows the viewing of tweets of the user who issued the command.

Every site will store a log containing tweet, block, and unblock events. This log should be replicated at all sites using the Wuu-Bernstein log algorithm. When a user executes the command, the application should list all tweet events in the local log, as specified above; `view` is a local operation.

Every site will also store a replicated dictionary that contains the block information for every user. Each entry contains a pair (user, follower); the entry may contain additional information if necessary. If this entry exists in the dictionary, this indicates that the follower is blocked from viewing the user's tweets. This dictionary should be maintained using the Wuu-Bernstein algorithm (using the log described above). This log-dictionary maintenance should include truncating block and unblock events from the log when possible.

To reduce communication overhead, the `block`, and `unblock` operations should be executed as local operations. These operations only update the log and dictionary on the site where the

commands are executed. When a user creates a tweet, it should be added to the local site's log, and a message should be sent to all non-blocked follower sites. Any log entries necessary to maintain the log and dictionary should be piggybacked on this message, as per the Wuu-Bernstein algorithm, So, if a non-blocked follower enters the `view` command immediately after this message is received, he or she should see the new tweet at the top of the timeline.

A correct implementation of the algorithm will ensure that when a user views a tweet T in his or her timeline, the user also views all tweets that causally precede T that he has permission to view (is not blocked from viewing). Because the application does not replicate local events immediately, a blocked follower F will not learn about a block issued by user U until there is a causal chain of events from the block operation to some event at site F. As a result, until F learns about the block, he or she will still be able to view tweets from U. However, he will not be able to view any tweets made by U after the block event (until F learns about an unblock event from U). These unviewable tweets may still be added to F's log as part of the Wuu-Bernstein algorithm.

Note that the Wuu-Bernstein dictionary algorithm assumes that for any possible dictionary entry D, insert(D) is executed at most once. In this application, a user may be blocked and unblocked multiple times. So, you will need to either adapt the Wuu-Bernstein algorithm to (correctly) handle the possibility of repeated inserts, or you will need to come up with some other workaround.

Your system should only send messages for tweet operations. Do not use any additional messages to sync on the logs on recovery, for example.

Implementation Details

You must implement a system that supports N sites, where N can range from 2 to 5. Your system can use a configuration file that gives the IP addresses and ports that will be used for communication between all sites. The application should tolerate crash-failures and recoveries, meaning that at any time, you should be able to terminate one or more sites (using Ctrl-C), and then later restart the failed site(s), and the application should continue to operate as specified.

Each site's log and dictionary should be stored on disk so that it will survive crashes. You need to provide, at least, a minimal UI to enter the four commands and to view the twitter timeline. A text-based UI is fine. The contents of the log and dictionary must also be viewable. This can be done by reading files or by issuing commands in the UI.

You will demonstrate your project on Amazon EC2, on multiple virtual machines in at least 3 different regions. You will use micro-instances. You can use any programming language supported by this platform that uses the message-passing model and does not mask failures (e.g., C, C++, Java, Python). Details about setting up accounts on EC2 will be provided soon.

Deliverables

Turn in your source code and a 2-page project report. The report should provide details of your design and implementation. The code and report will be submitted through Submitty. Details will be provided in the next few weeks. **No late submissions will be accepted.**

Project demonstrations will be scheduled closer to the project deadline.