Distributed Systems and Algorithms CSCI-4510/6510 – Fall 2017
Project 2

Assigned: Nov. 3, 2017
Teams formed by: Nov. 10, 2017 (10% penalty for missing this deadline)
Project due: Sunday, Dec. 3, 2017 at 11:00pm
Demos: Dec. 4 – Dec. 20, 2017

Overview

In Project 1, you implemented a distributed Twitter service using the Wuu-Bernstein algorithms for replicated logs and dictionaries. These algorithms have the benefit of simplicity, low message complexity, and tolerance of failures and message loss. However, these algorithms do not fully replicate every event on its occurrence, and so sites may not always have an up-to-date view of tweets and block and unblock events.

In this project, you will implement a similar distributed Twitter service using the Paxos algorithm to replicate tweet, block, and unblock events.

The distributed system consists of N sites, each corresponding to a single user. As in Project 1, 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. Your application should support the same operations as for Project 1, namely:

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 UTC time (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.

Implementation Details

You must implement a system with **five** sites. Each site must run on a different machine. 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 will maintain a write-ahead log; each log entry contains a single event: tweet, block, or unblock. The log should be fully replicated at all sites using the Paxos algorithm. A site's log copy should be stored in stable storage so it survives crashes. You should also use stable storage for the other state variables needed by the Paxos algorithm. A site's log should be viewable, either through a command in the UI or by examining a file.

~~In the full Paxos algorithm, to guarantee liveness, a single node acts as the leader. When your application starts, there is no leader site, so the full Synod algorithm will be used. Once the first log~~

~~entry is created, the leader site will be the site that proposed the winning value. This site will remain the leader until it crashes, at which point the system reverts back to the full Synod algorithm until a new leader is chosen by the same procedure. As specified in the Paxos algorithm, if a site does not know which site is the leader, it will use the full Synod algorithm to propose values.~~
See Paxos slides for updated requirement on leader-based Paxos.

Each site should also maintain several in-memory data structures: the timeline of tweets and the information about blocked users. When a follower is blocked or unblocked, the timeline on its site should be updated to remove or add the corresponding tweet events so that the timeline always contains only those tweets that are viewable by the user. These in-memory data structures do not survive crashes. When a site recovers, it must recreate these data structures by replaying the log.

Implementation Details for Teams of 1 and 2

- You may use TCP sockets for messaging. Because TCP is "reliable", there should not be any holes in a site's log. However, when a site is failed, it may miss some log entries. When a site recovers, it must learn the log entries it missed by executing the Paxos algorithm for each log entry.

Implementation Details for Teams of 3

- You must use UDP sockets for messaging, and you must keep the socket between a pair of sites open as long as the sites are both alive (i.e., you should not use socket creation as a failure detection mechanism). Because UDP is not reliable, you will need to implement timeouts in your code, as specified in the Paxos algorithm.
- In addition, a site may end up with holes in its log if commit messages are lost. You should implement a feature by which each site periodically (every 1 minute) checks whether there are holes in its log. If a hole is found, the site should execute the Paxos algorithm to learn the missing log entry.
- When a site recovers from failure, it should also use the Paxos algorithm to learn about any events it missed while it was down.
- Your application must include checkpointing of the timeline and block information. These data structures should be saved to stable storage every 5 events. When a site recovers from failure, it should recover these data structures from the most recent checkpoint, and then it should update these data structures by replaying log events that took place after the checkpoint was created.

You will demonstrate your project on Amazon EC2, on multiple virtual machines in at least 2 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).

Deliverables and Grading

Turn in your source code and a 2-page project report in pdf format. The report should provide details of your design and implementation. The code and report **must** be submitted through Submitty. **No late submissions will be accepted, and no submissions will be accepted over email. Any project not submitted via Submitty by the project deadline will receive a grade of 0.**

Project demonstrations will be scheduled closer to the project deadline. Your project grade will be based primarily on successful completion of test cases in your project demo.