Project 1 Design Doc

Lee Lee Choo (1032148; cll07@uw.edu)
Vaspol Ruamviboonsuk (1022906; vaspol@uw.edu)

Storage

One line per entry in file(s). Since it is only one client-server paradigm, we can have one file to store the tweets from one user.

To extend to later stages with multiple client, I think we should aim for multiple files. Each for one client. The file can be something like “user-tweet.txt”, and maybe we can extend such that each file represents each period of time.

Files:

User Management

Files	Usage
Users.txt	List of all users

Tweet Management

Files	Usage
[user].txt	Representing the tweets of the user

Follower-Followee Management

Files	Usage
[user]-following.txt	Representing who the user is following

Handling Failures

Files	Usage
.tmp	in the middle of an append

Keep track of who login

Files	Usage
login.txt	keeps track of who login

Keep track of who the server is connected to before it crashes

File	Usage
clients.txt	contains clients' node number so that server can send them a restarted message when it wakes up from a failure

Implementation

Client methods

Functions that extends class Function	Usage
Signup	Create new user
Login	Login
Logout	Logout
Tweet	Post tweet
Follow	Follow a user
Unfollow	Unfollow a user
Read	Read all the unread post

RPC (Interface provided by the server to the client)

- Create File
  - arguments: collection name, data in JSON
  - return: SUCCESS / FAILURE
- Read File (TwitterServer.READ = "read")
  - arguments: collection name
  - return: the content of the file
- Append File
  - arguments: collection name, data in JSON
  - return: SUCCESS / FAILURE
- Delete File (TwitterServer.DELETE = "delete")
  - arguments: collection name
  - return: the content being deleted
- Delete Lines (TwitterServer.DELETE_LINES
  - arguments: collection name, query in JSON
  - return: the entries that were deleted

Handling Failure

In the absence of node failure:

- Message drop, and message delay are not a problem in a absence of fault because ReliableInOrderMsgLayer provides reliable, in-order delivery in the absence of faults.

In the presence of node failure:

- Failure 1: server did not get the request
- Failure 2: server responded but the response got lost
- Failure 3: server crash

Assuming that there are no interleaving between create/delete in this assignment, we made create and delete idempotent. We ignore that fact that the file might be created twice or the delete file is not there.

Following Umar's logic of creating a temporary file

Solution to the presence of node failure:

```
1: String oldFile = read foo.txt
2: PSWriter temp = getWriter(.temp, false)
3: temp.write("foo.txt\n" + oldFile)
4: PSWriter newFile = getWriter(foo.txt, false)
5: newFile.write(contents)
6: delete temp
```

then on a server restart:

```
if .temp exists
 PSReader temp = getReader(.temp)
 if (!temp.ready())
   delete temp
 else
   filename = temp.readLine()
   oldContents = read rest of temp
   PSWriter revertFile = getWriter(filename, false)
   revertFile.write(oldContents)
   delete temp
```

so essentially there are 3 cases:
no temp file (state is consistent)
empty temp file, which means that the server failed between lines 2 and 3 (file has not been changed)
temp file with some content, which means that the server failed between lines 3 and 6 (possibly empty file)

In the presence of node(client) failure:
Since in our design, the client does not remember state, it does not need to recover.

Assumption 1: the file would not get too large, we will only worry about deleting the content in the file next time.

Assumption 2: In the middle of writing to file will not fail.