Distributed Systems Final Project Final Design
Distributed Systems CS437

Qianyu Luo - qluo3

Huabin Liu – hliu76

I. Group Strategy

There are four types of groups in our design.

1. Public groups of clients:

Each client will create its own group when it login as new user. These groups are used for servers to send reply to the connected clients. The groups' names are in the format client_<user name>.

2. Public groups of servers:

Each server will create a public group that only contains itself. These groups are used for clients to send updates to the servers they connect with. The groups' names are server<server index>.

3. Public group for all servers:

We create a public group for all servers. This group is used for multicasting updates and synchronization messages among servers. And servers in this group will receive the membership messages from spread, thus servers could know the memberships changes and process their updates. The groups' name is "servers_all".

4. Public group of servers and connected clients:

Each server has a public group that contains itself and the clients connected to it. So when the partition happens or the server crashes, the connected clients could know what happened by the membership change messages from this group. The groups' names are client_server<server index>.

II.   Data Structures

All the operations and commands are considered as updates in our projects. The structure of update is as following:

```
int    type;
char source;
lamport stamp;
int    update_index;
int    read;
int    membership[5];
char sender[MAX_NAME_LEN];
char receiver[MAX_NAME_LEN];
char subject[MAX_SUB_LEN];
char msg[MAX_MSG_LEN];
```

For lamport timestamp structure, we have two attributes:

```
int server_id
int time_stamp
```

With these two attributes, we can uniquely confirm an update or sort updates.

When it comes to reconciliation, the servers exchange their local vectors and the message type is status:

```
int type;
int server_id;
int vector[5];
```

For each server, we use linked list to manage all the updates that contribute data. And then, we also use another linked list to keep all the users. So generally, we use a linked list of linked list to keep each user's mails. The mails of each user are sorted in the order new to old.

III. Server

Our server is tolerant to crash and network partition. When receiving updates from clients, assign local lamport stamp to it, increment local time stamp index by 1, and update local vector. When receiving updates from servers, if its time stamp is smaller than local vector's entry for that server, no need to deliver, but if all the five servers are together at that time, delete relative update nodes or update files if exist. If the time stamp is larger than local vector's entry for that server, update local vector, deliver the update, and adopt that time stamp if it's larger than local time stamp.

General cases:

1. Updates from client
    a. A new mail: write it to disk, insert it to the receiver's mail list, then multicast the update to other users.
    b. Read a mail: use the receiver's name to find the user's mail list. According to the update index, find the exact mail node in the mail list and set read field to 1. Then overwrite the file on disk. Our way of naming a backup file is like <server>_<time_stamp>. Assign the file name to message field and then multicast to other servers.
    c. Delete a mail: similarly, use the receiver's name to find the user's mail list. According to the update index, find the exact mail node in the mail list and delete that node as well as the backup file on disk. Assign the target file name to message field and then multicast this update to other servers.
    d. Get list: use the receiver's name to find the user's mail list. Send all the nodes in the mail list back to user, followed by the final type message in the

end to indicate termination.

e. Get view: servers always keep an array with all the servers' index in the same partition. Upon request, send this array to user.

2. From server
    a. New mail: write the mail to disk and insert it to the receiver's mail list.
    b. Read a mail: use the receiver's name to find the user's mail list, then use the file name in the message field to get lamport time stamp information. If the mail exists, set it read and overwrite the file on disk.
    c. Delete a mail: use the receiver's name to find the user's mail list, then use the file name in the message field to get lamport time stamp information. If the mail exists, delete the file and the mail node.

Partition

During partition, the updates will be delivered directly. But meanwhile, a mail node will be inserted into update list and an update file will also be written to disk. The way we name an update file is like up_<server>_<time_stamp>. The update list is sorted in the order old to new.

Merge or come back after crash

When the network change or when a new server join in the servers_all group, every server multicast its local vector and all the servers keep other servers' vector in a matrix. This kind of message is SYN_TYPE. There is a counter for how many SYN_TYPE messages are received and an array to keep track of whose vector is received and who's not. Only when receiving vector from a new server, the counter will increment. As soon as merge or join happens, the counter and array will be reset. In the case of cascading merge, the former one should be discarded and the newest one will be tracked. When the number of received SYN_TYPE message equals to the number of servers in current partition, servers start scan the matrix, looking for the one with least information, the one with most information and set it to be updater. If tie, the server with smaller index will be updater. The updater should multicast everything in its update mail list that happens later than min information. Only if merge happens among all the five servers, update list nodes and relative update files will be deleted.

IV. Client

Update filed assignment:

1. List command:
   type: LIST_TYPE;
   source: FROM_CLIENT;
   receiver []: username of current client;
   subject []: group name for the current client;

2. Mail command:
   type: MAIL_TYPE;
   source: FROM_CLIENT;
   sender []: username of sender;
   receiver []: username of receiver;
   subject []: subject of the mail;
   msg []: message of the mail;

3. Delete command:
   type: DELETE_TYPE;
   source: FROM_CLIENT;
   update_index: The index of the mail that want to delete;
   receiver []: username of current client;
   subject []: group name for the current client;

4. Read command:
   type: READ_TYPE;
   source: FROM_CLIENT;
   update_index: The index of the mail that want to read;
   receiver []: username of current client;
   subject []: group name for the current client;

5. View command:
   type: VIEW_TYPE;
   source: FROM_CLIENT;
   receiver []: username of current client;
   subject []: group name for the current client;

Algorithms for command in client:

1. [-u <username>]: login with a username:
   Client leaves the original client_<username> group, and join the new client_<username> group. Don't need to send any updates. The user has to login with a username before any other commands.

2. [-c <server>]: connect to a specific server:
   Client leaves the original client_server<server index> group, and then join the new group. Don't need to send any updates. The user has to connect to the server after login with a new username or before doing the following commands. Also, when partition happens or the connected server crashes, the client will be notified.

3. [-l]: list the received mails:
   Compose and send the list update to the connected server. The server replies mails in the user's inbox in order. The latest one should be the first mail in the inbox. After

received all the mails, the server would reply a FINAL_TYPE reply to the client.

4. [-m]: mail a massage:
   Compose and send the mail update to the server.

5. [d <index>]: delete that mail in the current list:
   Compose and send the delete update to the server. The update index contains the index of the mail in the current user's inbox.

6. [r <index>]: read that mail in the current list:
   Similar to the delete command.

7. [v]: print the membership of servers in the current network:
   Compose and send the view update to the server. The server is supposed to reply an array that contains all the servers' indices in the current partition. The client would print all the servers in the current partition.