Distributed Systems Final Project Design Document

Minqi Ma (JHED: mma17), Jerry Chen (JHED: gchen41)

General Idea

- Each server maintains 5 log files of updates from each server, and one state file for easy recovery from crash.
- Client sends update to the server's public group. Server first saves the update to its log file, and sends the update to other servers.
- Each update has a lamport timestamp of counter, server_index and index. Server maintains a 5*5 matrix of the indices.
- Execute an update
    - Join
        - Do not save in log file
        - Find if the client is in any room previously
        - Inform other servers to remove this client from the old room, and add to the new room
    - Append
        - Create a message with id = counter and server_index of this update.
        - Messages in a room is sorted in the canonical order of the id.
        - Client saves the message id in memory and use it to create like/unlike updates.
    - Like/unlike:
        - If the user has never liked/unliked this message, create a new like node with the username, counter and server_index of this update.
        - If the user has liked/unliked this message, compare saved timestamp and new update's timestamp. Only update the liked status if this update is in a higher canonical order.
    - If any update is out of order, ignore the update.
- **Merge steps**
    - Server with the smallest server index sends a START signal, in case one merge happens in the middle of another.
    - Server sends current participant list for each room to servers group. After merge server informs server-room group of the reconciled participant list.
    - Servers exchange matrices. Server which knows the most about one server's updates will send missing updates for others to catch up with (with coarse-grain flow control).
    - Each server wait until receiving all the missing updates. Sort them in the canonical order and execute one by one.
    - If client sends any updates during merge, put in buffer and execute after merge.
- Server can clean logs in memory if all 5 servers know the updates up to some index.
- Server saves state of data structures for every FREQ_SAVE updates received.

Group Architecture

- Server's public group, named "server?". It is used for clients to send updates to the server.

  ```
  #server1#ugrad1
  ```

- Client's private group, named "#<username>#ugrad?". It is used for server to send messages specifically to this client.
- Server-client group, for each connected server and client pair, named "server?-<username>-ugrad?". It is used to detect client/server connectivity.

  ```
  #server1#ugrad1
  #<username>#ugrad?
  ```

- Server-room group, for all clients in this room connected to the same server, named "server?-<roomname>". It is used for the server to send room updates

  ```
  #user1#ugrad1
  #user2#ugrad1
  #user1#ugrad2
  …...
  ```

- Servers group with all 5 servers in it, named "servers". It is used to send messages among servers, and detect servers' connectivity.

  ```
  #server1#ugrad1
  #server2#ugrad2
  #server3#ugrad3
  #server4#ugrad4
  #server5#ugrad5
  ```

Messages

Chatroom Membership

- JOIN <room_name>: client to server, request to join a room
- ROOMCHANGE <client_name> <old_room> <new_room> <server_index>: server to servers, notify other servers if my client changes room
- PARTICIPANTS_ROOM <user1> <user2> ...: server to client, inform client of current participants for this room
- PARTICIPANTS_SERVER <room_name> <server_index> <client1> <client2> ...: server to servers, inform other servers of current participants of a room, used to reconcile on chatroom membership during merge

Updates

- <update> =
  - a <room_name> <username> <content>

        - l <room_name> <counter of the liked message> <server_index of the liked message> <username>
        - r <room_name> <counter of the unliked message> <server_index of the unliked message> <username>
- UPDATE_CLIENT <update>: client to server, send update to server
- UPDATE_NORMAL <counter> <server_index> <index> <update>: server to servers, stamp the client's update with a lamport timestamp, save it to log file, and send it to other servers
- UPDATE_MERGE <counter> <server_index> <index> <update>: server to servers, reconcile on missing updates during merge
- APPEND <counter of the message> <server_index of the message> <username> <content>: server to client, client will append a new message and save its message id
- LIKES <counter of the message> <server_index of the message> <num_likes>: server to client, client will update number of likes for this message

Others
- MATRIX <25 integers>: server to servers, exchange matrix during merge
- START <5 integers 0/1>: signal to start merging for the current network component
- CONNECT: client to server, request to connect with server
- MESSAGES (<counter> <server_index> <creator> <num_likes> <content>\n)*: server to client, send latest 25 messages of the room
- HISTORY
    - HISTORY <room_name>: client to server, request history of the room
    - HISTORY <creator> <num_likes> <content>: server to client, send the history
- VIEW
    - VIEW: client to server, request membership of each server
    - VIEW <5 integers 0/1>: server to client, 0/1 represents connectivity for each server in the current network component

Data Structures

Client
- struct message
    - *int counter*: message id
    - *int server_index*: message id
    - *char* creator*: username of the creator
    - *char* content*: content of the message
    - *int num_likes*
    - *struct message* next*
- *struct message* messages:* up to 25 messages in this room
- *struct participant*
    - *char* name*
    - *struct participant* next*
- *struct participant* participants: list of participants in this room*
- *char* username*

- *char* room_name*
- *int server_index*: connected server index

<u>Server</u>

- *struct room*
    - *char* room_name*
    - *struct participant* participants[5]:* array of 5 lists of participants from each server
    - *struct message* messages*: list of messages in this room
    - *struct room* next*
- *struct room *rooms*: list of all rooms
- *struct message*
    - *int counter*: message id
    - *int server_index*: message id
    - *char* content*: content of the message
    - *char* creator*: username of the creator
    - *struct like* likes*: list of likes and unlikes of different users
    - *struct message* next*
- *struct like*
    - *char* username*
    - *bool liked*: if this user likes/unlikes the message
    - *int counter*: counter of highest timestamp applied
    - *int server_index*: server_index of highest timestamp applied
    - *struct like *next*
- *struct participant*
    - *char* name*
    - *struct participant* next*
- *int my_counter*
- *int my_index*
- *struct log* logs[5]*: 5 lists of logs from each server
- *struct log*
    - *int counter*
    - *int server_index*
    - *int index*
    - *char* content*
    - *struct log* next*
- *int matrix[5][5]*
- *struct log* buffer*: buffered client updates received in the middle of merging
- *struct log* updates*: updates from log file OR received missing updates during merge
- *bool connected_servers[5]*: current network connectivity
- *int num_updates*: number of updates expected to receive during merge
- *bool received_matrix[5]*: if I have received the matrix from server (i+1) during merge
- *bool merging*: in merging state or not

- *bool received_start*: if received start merging signal, to prevent from receiving updates from previous network change
- *int sent_updates[5]*: highest index of sent missing logs

Algorithm

Client

User command

- u <username>
    - Leave previous server-client, server-room group if exist
    - Clear data structures, like *messages*, *participants* list
    - Create its new private group with the new username
- c <server_index>
    - Record server index; later the client will send updates to this public group
    - Leave previous server-client group, if exists
    - Clear data structures, like *messages*, *participants* list
    - Join new server-client group
    - Send “CONNECT” message to the server
    - Start a timer. If the server is not connected when timer expires, inform user that connection failed.
- j <room_name>
    - Leave previous server-room group, if it exists
    - Join new server-room group
    - Send “JOIN <room_name>” message to the server’s public group
- a <content>
    - Send “UPDATE_CLIENT a <room_name> <username> <content>” to the server’s public group
- l <line_number>
    - Find the message’s id (counter and server_index) in *messages* list
    - Send “UPDATE_CLIENT l <room_name> <counter of the liked message> <server_index of the liked message> <username>” to the server’s public group
- r <line_number>
    - Find the message’s id (counter and server_index) in *messages* list
    - Send “UPDATE_CLIENT r <room_name> <counter of the unliked message> <server_index of the unliked message> <username>” to the server’s public group
- h
    - Send “HISTORY <room_name>” to the server’s public group
- v
    - Send “VIEW” to the server’s public group

Regular messages

- Receive “MESSAGES ...” from server
    - Construct *messages* list accordingly

        - Display
- Receive “PARTICIPANTS_ROOM <user1> <user2> ...” from server-room group
    - Clear *participants* list and reconstruct the list accordingly
    - Display
- Receive “APPEND <counter of the message> <server_index of the message> <username> <content>” from server-room group
    - Append a new message to *messages* list and record its id
    - If list size is larger than 25, remove the first message
    - Display
- Receive “LIKES <counter of the message> <server_index of the message> <num_likes>” from server-room group
    - Search the message id in *messages* list
    - If the message exists, update its number of likes
- Receive “HISTORY <creator> <num_likes> <content>” from server
    - Display
- Receive “VIEW <5 numbers 0/1>” from server
    - Display

<u>Membership messages</u>

- Receive membership change in server-client group
    - JOIN (client itself or server joins the group)
        - If the server joins the group, mark as successfully connected to server
    - DISCONNECT/NETWORK CHANGE (server crashes/daemon crashes)
        - Leave previous server-room group, if exists
        - Clear data structures, like *messages*, *participants* list
        - Leave server-client group
- Receive membership change in server-room group
    - Do nothing; the server will send corresponding updates later

Server

Start: ./server <my_server_index>

- Upon the server starts
    - Join servers group
    - Join its public group “server<my_server_index>”
    - If there is state file
        - Reconstruct data structures from state file
        - Retrieve counter, index and matrix; matrix[my_server_index] vector represents indices of last executed logs to reach this state

      else
        - initialize empty data structures, *counter* = 0, index = 0, matrix is all 0’s
    - For every server, if log file exists
        - Check matrix to see if all 5 servers have logs up to some lowest index
        - Loop through each line, if log index <= lowest index, skip

        - If log index <= *matrix[my_server_index][server_index]* (i.e have executed this log to reach the state)
            - Append to *logs[server_index]* list
        - If log index > *matrix[my_server_index][server_index]* (i.e have not executed this log)
            - Insert it to *updates* list in canonical order
    - Traverse each log in *updates* list
        - Append the update to *logs[server_index]* list
        - Adopt the counter if higher
        - Update matrix accordingly
        - If the update is from myself, adopt the index
        - Execute the update
- Start receiving messages

<u>Regular messages</u>

- Receive "CONNECT" message in the public group
    - Join server-client group "server?#<client_name>"
- Receive "JOIN <room_name>" message from <client_name> in the public group
    - If in merging state, put it in *buffer* list
    - Search rooms and see if the client is previously in any room
    - Send "ROOMCHANGE <client_name> <old_room> <new_room> <server_index>" to servers group (<old_room> can be "null")
    - Send up to latest 25 messages of this room to the client's private group
        - Format: "MESSAGES" followed by messages
        - Each message: <counter> <server_index> <creator> <num_likes> <content>\n
- Receive "ROOMCHANGE <client_name> <old_room> <new_room> <server_index>" in servers group
    - If in merging state, put update it in *buffer*
    - If old_room is not null, remove client from *participants[server_index]* list in the old room
    - if new_room is not null
        - create the new room if the room does not exist
        - add client to *participants[server_index]* list in the new room
    - If I have clients in the affected rooms (i.e *participants[my_server_index]* is not empty)
        - Send new participant list to the affected server-room groups, "PARTICIPANTS_ROOM <user1> <user2> ..."
- Receive "UPDATE_CLIENT <update>" in the public group
    - If in merging state, put update in *buffer* list
    - Increment counter
    - Increment index
    - Stamp the message with counter + server_index + index
    - Write the message to log file

        - Append to *logs[my_server_index]* list
        - Update *matrix[my_server_index][my_server_index]* to new index
        - Send “UPDATE_NORMAL <counter> <my_server_index> <index> <update>” to servers group
- Receive “UPDATE_NORMAL <counter> <server_index> <index> <update>” in servers group
    - If the update is not sent by myself
        - If index is out of order from *matrix[my_server_index][server_index]*, return
        - Write it to log file “*server[my_server_index]-log[server_index].out*”
        - Append it in *logs[server_index]* list
        - Adopt the counter if it is higher
        - Update *matrix[my_server_index][server_index]* to the new index
    - If update is “a <room_name> <username> <content>”
        - Create a new message and insert it to *messages* list of the room in the canonical order of message id
        - If I have clients in the room (i.e *participants[my_server_index]* is not empty)
            - Send “APPEND <counter> <server_index> <username> <content>” to the server-room group
    - If update is “l <room_name> <counter of the liked message> <server_index of the liked message> <username>”
        - If user is the creator of the message, return
        - If there is already a like node in *likes* list with the same username
            - If this update’s timestamp is not higher than saved one, return
            - Adopt counter, server_index of the node to the higher value
            - If the message is previously unliked by this user
                - Update it to liked status
                - If I have clients in the room (i.e *participants[my_server_index]* is not empty)
                    - Send “LIKES <message’s counter> <message’s server_index> <num_likes>” to the server-room group
        - If there is no node in *likes* list with the same username
            - Create a new node and append it to *likes* list
            - If I have clients in the room (i.e *participants[my_server_index]* is not empty)
                - Send “LIKES <message’s counter> <message’s server_index> <num_likes>” to the server-room group
    - If update is “r <room_name> <counter of the unliked message> <server_index of the unliked message> <username>”
        - If user is the creator of the message, return
        - If there is already a like node in *likes* list with the same username
            - If this update’s timestamp is not higher than saved one, return

- Adopt counter, server_index of the node to the higher value
- If the message is previously liked by this user
  - Update it to unliked status
  - If I have clients in the room (i.e *participants[my_server_index]* is not empty)
    - Send “LIKES <message’s counter> <message’s server_index> <num_likes>” to the server-room group

- If there is no node in *likes* list with the same username
  - Create a new node and append it to *likes* list

- For every FREQ_SAVE updates received
  - Save data structures to state file

(<u>Reconciliation</u>)

- Receive “START <5 integers 0/1>” in servers group
  - If 5 integers do not match with current network connectivity, return
  - *received_start* = true
  - for every room
    - Clear *participants[server_index]* for servers not in the current network component
    - Send *participants[my_server_index]* to the servers group, “PARTICIPANTS_SERVER <room_name> <server_index> <client1> <client2> ...”
  - Send my matrix “MATRIX <25 integers>” to servers group
  - Initialize *received_matrix* array to false for all the connected servers
  - Initialize number of updates expected to receive *num_updates = 0*
- Receive “PARTICIPANTS_SERVER <room_name> <server_index> <client1> <client2> ...” in servers group
  - If *received_start* == false, return
  - Create the room if it does not exist in *rooms* list
  - Clear *participants[server_index]* list in this room and construct new list accordingly
- Receive “MATRIX <25 integers>” in servers group
  - If *received_start* == false, return
  - Adopt all integers if it is higher, except for my vector *matrix[my_server_index]*
  - Mark as received matrix from this server (*received_matrix[server_index]* = true)
  - If *received_matrix* array is all true, (i.e just received all matrices)
    - Clear *updates* list
    - For every room
      - If I have clients in the room (i.e *participants[my_server_index]* is not empty)
        - Send new participant list of this room to server-room group, in the format of “PARTICIPANTS_ROOM <client1> <client2> …”

- For each server column
    - Clear *logs[server_index]* list up to the lowest index among 5 servers
    - Calculate the number of updates expected to receive and add to *num_updates*
    - If I have the highest log index with the lowest server index
        - Get update in *logs[server_index]* from lowest index+1 to highest index
        - Send INIT_SEND_SIZE updates to servers group in format of “UPDATE_MERGE <counter> <server_index> <index> <update>”
        - Record index of highest sent updates in *sent_updates[server_index]*
- If *num_updates == 0* (i.e no updates to merge)
    - Mark as out of merging state
    - If *buffer* list is not empty
        - Execute UPDATE_CLIENT or JOIN or HISTORY or ROOMCHANGE as normal

- Receive “UPDATE_MERGE <counter> <server_index> <index> <update>” in servers group
    - If *received_start* == false, return
    - If *num_updates* == 0, return (i.e it is an update sent in the previous network change)
    - *num_updates*--
    - If update is sent by myself
        - If *sent_updates[server_index]* has not reached the highest log index I have
            - Send one more UPDATE_MERGE to servers group
            - *sent_updates[server_index]*++
    - If I do not have the update (i.e index > matrix[my_server_index][server_index])
        - Insert it in *updates* list in the canonical order
    - If *num_updates == 0*, (i.e just received all missing updates)
        - For every update in *updates* list
            - Write it to log file
            - Append it to *logs[server_index]* list
            - Adopt the counter if it is higher
            - Update *matrix[my_server_index][server_index]* to the new index
            - Execute the update
        - Mark as out of merging state
        - If *buffer* list is not empty
            - Execute UPDATE_CLIENT or JOIN or HISTORY or ROOMCHANGE as normal
- Receive “HISTORY <room_name>” from <client_name> in the public group

        - If in merging state, put it in *buffer* list
        - for each message in this room
            - Send “HISTORY <creator> <num_likes> <content>” to client’s private group
- Receive “VIEW” from <client_name> in the public group
    - Send “VIEW <5 numbers 0/1>” to the client’s private group

<u>Membership messages</u>

- Receive membership change in server-client group
    - LEAVE/NETWORK CHANGE/DISCONNECT (i.e client reconnects to another server or crashes)
        - Search all rooms and see if the client is previously in any room
        - If the client is previously in a room, send “ROOMCHANGE <client_name> <old_room> <null> <server_index>” to servers group
        - Leave server-client group

(<u>Reconciliation</u>)

- Receive membership change in servers group (i.e servers crash/network partition)
    - Mark as in merging state (i.e *merging* = true)
    - Record network connectivity in *connected_servers* array
    - *received_start* = false
    - If I have the lowest server_index in the current network component
        - Send “START <5 integers 0/1>” representing current network connectivity

File Format

<u>Log</u>

- Name: server<server_index>-log<server_index of logs>.out
  e.g server1-log1.out, server1-log2.out
- Line format: <counter> <server_index> <index> <content>
- <content> =
    - a <room_name> <username> <content>
    - l <room_name> <counter of the liked message> <server_index of the liked message> <username>
    - r <room_name> <counter of the unliked message> <server_index of the unliked message> <username>

<u>State</u>

- Name: server<server_index>-state.out
  E.g server1-state.out, server2-state.out
- Format (line by line)
    - counter
    - matrix: 5 lines, each line has 5 integers
    - <num_rooms>
    - <room_name> <num_messages>, followed by messages in this room
    - Message:

        <counter> <server_index> <creator> <length of *likes* list> <content>
        Followed by <like> lines
    - Like:
        <username> <counter> <server_index> <liked 0/1>
- e.g state1.txt
  10 ← counter
  1 2 3 4 5 ← matrix
  1 3 4 5 6
  6 3 5 6 3
  1 3 2 1 1
  0 0 0 0 0
  2 ← 2 rooms
  room1 2
  1 1 user1 1 hello everyone ← message line
  user2 1 2 1 ← 1 like node
  2 1 user2 0 hi there
  room2 1
  4 2 user3 2 what's up
  user1 1 2 0
  user2 1 3 1