CSCE 662: Design Document for MP 2.2 SyncService
By Tanmai Harish (UIN: 434007349)

1. Introduction

The SyncService synchronizes data between different server clusters in a distributed system. A synchronizer initially doesn't know about the other synchronizers in the system. Hence, an initial RegisterSyncServer RPC call is made to the Coordinator. Then, regularly, the synchronizer calls the GetSyncServers RPC, which polls for data from the clusters. Then, based on whether data on following, followers and timeline has changed, it appends the required information to its cluster's corresponding users. Synchronizer neither communicates with any server nor with the clients. The communication is only with the coordinator and each other.

Commands to Run Each Test Case

1. **Test Case 1 (Sanity Check)**

Coordinator and Server

```
./coordinator -p 9090
./tsd -c 1 -s 1 -h localhost -k 9090 -p 10000
./tsd -c 1 -s 2 -h localhost -k 9090 -p 10001
```

Wait 10 seconds

Client

```
./tsc -k 9090 -u 1
./tsc -k 9090 -u 4
```

2. **Test Case 2**

```
make reset
./coordinator -p 9090
```

Server

```
./tsd -c 1 -s 1 -k 9090 -p 10000
./tsd -c 1 -s 2 -k 9090 -p 10001

./tsd -c 2 -s 1 -k 9090 -p 20000
./tsd -c 2 -s 2 -k 9090 -p 20001

./tsd -c 3 -s 1 -k 9090 -p 30000
./tsd -c 3 -s 2 -k 9090 -p 30001
```

Synchronizers

```
./synchronizer -j 9090 -n 1 -p 1234
./synchronizer -j 9090 -n 2 -p 1235
./synchronizer -j 9090 -n 3 -p 1236
```

Client

```
./tsc -k 9090 -u 1
./tsc -k 9090 -u 2
./tsc -k 9090 -u 3
```

Note: After sending the messages p31, p32 from user3 in the timeline mode (Last command in this test), **wait for 60s** to stream the messages to the other two users 1 and 2.

Wait 60s after follow commands and after sending messages in the timeline mode. This much time is needed to synchronize all clusters with the latest data.

3. **Test Case 3**

   Ctrl+C to kill the users 1,2,3
   Ctrl+C to kill the master server running on port 20000

   Wait for 60 seconds

   Now start 4 clients with this command

```
./tsc -k 9090 -u 1
./tsc -k 9090 -u 2
./tsc -k 9090 -u 3
./tsc -k 9090 -u 5
```

   Wait for 60 seconds

   Now go ahead with the commands for the test case

Testing test case with a startup script

1. cd into the folder
2. **make**
3. **bash tsn-service_start.sh**
4. Wait 60s
5. Start the clients for the test cases
6. After test case 1, in another terminal run “**make reset**”
7. Start the clients for test case 2

8. To kill the master server in cluster 2, **“ps aux | grep “-c 2 -s 1”** see the process id, and kill it by typing **“kill -9 processId”**
9. **Ctrl+c** to kill the clients when required.

RPC Definitions

1. RegisterSyncServer

   Synchronizer Side

   This is needed to let the coordinator become aware of the sync server. At any point, only the coordinator has a global view of all the synchronizers in the distributed system.

   a. Create a ServerInfo object with the synchronizer server hostname, port, and synchronizer ID.
   b. Use the created coordinator stub and send these details via the RPC
   c. Wait for Confirmation. If GRPC status is not ok, then exit

   Coordinator Side

   a. When an RPC call is received, extract server info details
   b. Add this synchronizer server information to the synchronizer list object.
   c. Set true in the Confirmation status, and return grpc::Status::OK

2. GetSyncServers

   A synch server uses the GetSyncServers RPC call to request information about all synchronizer servers registered in the system.

   Coordinator Side:

   a. Upon receiving a GetSyncServers request, the coordinator iterates through its list of registered synchronization servers.
      For each synchronization server, it adds the server's information (ServerInfo) to the AllSyncServers Response.

   Synchronizer Side:

   a. The synchronizer periodically sends a GetSyncServers request to the coordinator. It collects the response containing the ServerInfo of all registered synchronizer servers.
   b. The synchronizer updates its local list of synchronization servers with the received information.

3, GetUserTLFL (SynchService)

   a. Get the current users registered in the cluster.
   b. For every current user, obtain the userId, their follower list, their following list, and their sent messages.
   c. Do this for both the master and slave files, use the data of the longer file.
   d. Insert this data into the UserTLFL message object.
   e. Add this message object to the AllData message. When done, return Status::OK

Design Document for SyncService

4. Heartbeat

Server Side

a. Initially, set the server type as "new".
b. Send heartbeat rpc every 10s to the coordinator. If grpc Status is OK, check the confirmation for the assigned type "master/slave", use this type for further heartbeat messages.
c. For a slave, if the type returned by the coordinator is now "master", then perform a copy of all the "master" data to the "slave" data files. The slave is now ready to start serving client requests as the master.

Coordinator Side

a. Receive the server info object from the RPC from the client.
b. Get the cluster id from the server info, then from the *routingTable* map, use the cluster as the key, and see how many servers are active in this cluster.
    i. If no servers are present, mark this server as "master" and return this confirmation.
    ii. If a server is present and its type is master and active, mark this current server as "slave" and return this confirmation.
    iii. Otherwise, mark this current sever as "master" and return this in the confirmation.

Threads

1. run_synchronizer (Synchronizer)

   This thread in the synchronizer is responsible for contacting the other synchronizers in the system for the data in the other clusters, checking if the files have been updated, and then appending the required data to the corresponding files in its cluster.
    a. Get all synchronizers using the GetSyncServers RPC, then create the stubs for the other synchronizers.
    b. In a while loop, make a GetTLFL RPC for each stub to get users' data in the other cluster.
        i. For each user present in the result, check their following list, if a new user has been added and they present in the current sync cluster, add the other user to the follower list of the current user.
        ii. Similarly, check the user's sent messages in the other cluster, for each user. If they have written a new message, then check their follower list. If the user in their follower list is present in the current cluster, then add this message to the timeline of the current user.
        iii. Sleep for 20s, go back to (i)

2. updateTimelineThread (Server)

   The updateTimelineThread is a separate thread responsible for continuously updating users' timeline streams based on messages received from other servers. It ensures that each user's stream shows the most updated information.
    a. In a while loop, retrieve the list of current users from the local file.

b. For each user, read their timeline file and identify new messages.
c. For each new message check, see if the message has come from a use outside their cluster. If from the same cluster, then skip streaming this message to the client (since the server timeline RPC stream would handle communication between users in the same cluster)
d. If the message from a user from another cluster, check if the current user is online and has a valid stream, if yes, then use the grpc steam write to write this message to their stream.
e. Sleep for 15s, go back to (i)

Data Consistency

The design ensures data consistency by synchronizing timelines to follow data across clusters and synchronizing necessary data between the master and slaves.