CS425 Distributed System MP1 Report

Haitong Tian (htian3), Mengjia Yan (myan8)

Part 1:

Implementing the channel delays:

For each processor(or replica, we call it data center), we have three kinds of threads as following.

1) Server thread is used to listen to sockets connected to other data center and receive all messages from others. Each server thread listens to one specific socket, therefore, the number of server threads equals to the number of data centers it connects to. After receiving messages, it will parse the message, apply required memory read/write operations and create acknowledge message accordingly.
2) Client thread is used to read input either from the terminal or a formatted file. It will parse the command line and create messages, sets up message timestamp, delay. Afterwards the message will be inserted into a message queue which is implemented as a FIFO queue and shared by all threads.
3) Message thread which is used to keep reading from message queue and send out messages to its corresponding destination. It’s where message delay is implemented. Before sending a message, the sending time(message create time+delay) of the message will be compared with current system time. If the message is not ready to be sent, the thread will be put to sleep.

Part 2:

Implementing linearizability:

We build a leader server to implement linearizability. Every message will go through the four stages below.

1) The routine starts as a general data center needs to send a message. It first forwards the message to the leader server.
2) The leader server broadcasts the message to all data centers in the network.
3) Upon receiving a message from the leader server, the data center will do corresponding operations and send back an ACK message containing essential reply information, which indicates that it has successfully received the message.
4) The leader server waits for 4 ACK messages. After all ACK messages successfully delivered, it sends an ACK message to the original data center, indicating the required operation have been completed by all data centers in the network. Only after a data center receives an ACK message from the leader server will it continue to process the next message in the message queue.

Linearizability is implemented through leader server. There exists a FIFO message queue in the leader server, only after receiving 4 ACK messages will the leader server starts broadcasting next message. In this way, total order broadcast can be guaranteed.

Implementing sequential consistency:

The way we implement sequential consistency is similar to linearizability, except that when performing read operations, it is read from the local copy instead of going through the 4 stages message passing above.

Eventual consistency:

Eventual consistency is implemented the way below.

1) The routine starts as a data center tries to broadcast a read/write/update message. Instead of forwarding one message to leader server, it creates four messages with same content but different destinations. Message thread will read the four messages from message queue and send them at corresponding time, which varies due to different channel delays. After finishing sending message, this data center will enter “WAIT” state.
2) Upon receiving messages, all data centers take the same operations as described in linearizability. The ACK message will be forwarded to the original data center instead of leader server.
3) After receiving all required number of messages, the data center will exit “WAIT” state to “NORMAL” state, indicating the operation has been finished. The required number is determined by eventual consistency mode. In mode 3, 1 message is required; in mode 4, it has to wait for 2 messages. For the remaining ACK messages, we will ignore them if no inconsistency fixed is needed.

In order to avoid deadlock which may occur when data center is asked to respond with an ACK message during “WAIT” state, we add an extra thread, ACK thread, which is used to send ACK message.

Inconsistency fix:

Our inconsistency fix occurs upon write/read operations and only applies to local copy. No extra messages exchange is required.

- For write, when a data center receives a insert/update message, it compares the timestamp of existed key-value pairs. If it’s an older message, discard the message. Otherwise, apply write operation accordingly.
- For read, in our design, without inconsistency fix, the remaining ACK messages(the number of such message is 3 in mode 3, and 2 in mode 4) will be ignored. However, we find these messages are actually useful and convenient resource for inconsistency fix. Instead of ignoring them, the timestamp of key-value in these messages will be compared against to existed timestamp. If a more recent one is detected, an update will be applied.

todo:

- fix all timestamp, that printed out formatted timestamp instead of milliseconds
- when “get” finish, display the result in mode 3/4
- when write finish, display the completion message -> to distinguish 1,2 and 3, and 4
- for get, it may not existed, fix nullpointer bug
- inconsistency fix for get
- inconsistency fix for insert/update ⇒ print out successful write or ignore old write