ECE428 MP2 Report

Herman Pineda - hpineda2
Jacob(Chenhao) Wu - cwu57

Code Organization

General Structure

For our implementation, each replica consists of a server and at least 7 clients that attempt to connect to the all the other servers in the system as specified in the configuration file. Depending on the consistency models, we have designed some structs that aid in the thread-spawning process nature of this multi-threaded program. That is, each consistency model was implemented in such a manner as to require a multitude of threads to execute the requested operation of the user via the terminal.

Linearizability

The sequencer by default is set to process 0 and cannot be changed. For each process, an dynamic array is used as the holdback queue, which would hold the incoming non-order messages from other processes. Once the clients receive an order message from the sequencer, it will find the corresponding message to deliver in the holdback queue.

Eventual Consistency

The code for this model is characterized by local key and store arrays at each replica that are modified depending on the timestamp of the received messages at a replica, and per the requested operation. Essentially, when a given operation is given via the command terminal, a thread is responsible for multicasting the operation to all replicas in the system to support the overall functionality of the consistency model. Upon receiving a message, a client is responsible for responding appropriately to complete the operation sent out by the particular replica of the system.

Functionality Description:

For the Key-Value Store Consistency programming assignment, the linearizability model was implemented by Jacob Wu, and the eventual consistency model was implemented by Herman Pineda.

An execution is said to be linearizable, if there exists a permutation of the operations in the execution that is per-process order-preserving, valid and real-time order-preserving. All executions resulting from shared memory implemented using Algorithm 3(which uses a sequencer) satisfy linearizability. Specifically, the algorithm 3 for shared memory basically is saying that among n nodes, we pick a sequencer, whose job is to multicast the sequence of the messages to be delivered. Aa a result, every nodes in the group should deliver in the same order specified by the sequencer.

The eventual consistency model implemented guarantees that eventually all replicas in the system will contain the same key-value stores. Each replica implements the “last writer wins” rule through the use of timestamps. All executions of operations are implemented using Algorithm 1 along with the “last writer wins” rule to achieve the given functionality. This particular implementation, however, requires that for each read operation issued, the value of the specified key must be retrieved from at least R replicas and the value with the latest timestamp is returned, and upon each write operation, the new value of the specified key must be written to at least W replicas while abiding by the “last writer wins” rule at each replica. This implementation supports a consistency model in which each replica in the system has server/client functionality in order to communicate with other replicas. Different threads are utilized at each replica for receiving and sending messages, carrying out the requested get/put operations and responding correspondingly, and to make use of the shared variables/buffers across a specific process that determine its behavior.