Bayou KV Store

Yicong Wang | Xiaofan Lu
yw5567 | xl4326
yicong.wang@utexas.edu | xiaofan@cs.utexas.edu

Slip days used (this project): 0 Slip days used (total): 1

1. Implementation Overview

The code base is structured as follows:

a. framework: the socket framework provided in Project 1

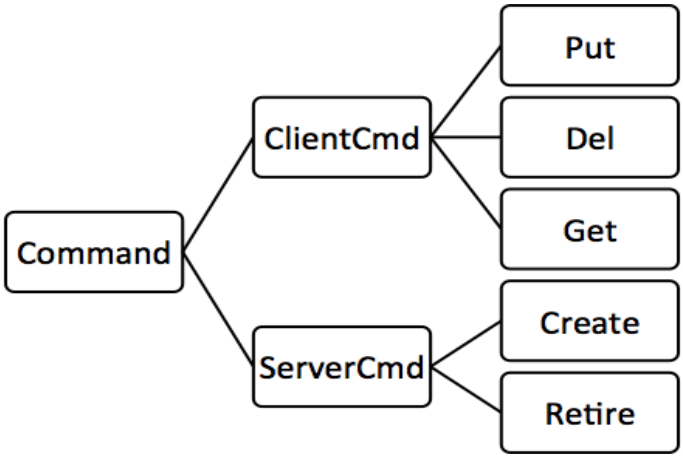
b. msg: different kinds of messages used in Bayou protocol:

   i. Message: superclass of all messages, serializable

Purpose	Message Type
Server to Client communication	ClientMsg, ClientReplyMsg
Creation of server	CreateMsg, CreateReplyMsg
Anti-entropy	AERqstMsg, AERplyMsg, AEAckMsg, AEMultiAckMsg
Primary Handoff	PrimaryHandOffMsg

c. command:

Command from Server/Client with the following class hierarchy:



d. util: common data structures used
    i. Constants: constants used, including debug mode, maximum nodes supported and thread sleep time
    ii. MsgQueue: Blocking Message Queue
        ** ref: http://tutorials.jenkov.com/java-concurrency/blocking-queues.html*
    iii. PlayList: the “database”, compare
    iv. ReplicaID: (acceptTime, parent)
    v. SessionManager: on client side, as specified in “Session guarantees for weakly consistent replicated data”. Maintains
        1. ReadVector: version vector of all the writes relevant to the latest read.
        2. WriteVector: version vector of all the writes performed in this session.
    vi. Version Vector: a.k.a Vector Clock
    vii. Write: an write entry
        (csn, acceptTime, ReplicaID) - Command
    viii. WriteLog: a log of all the writes received.
e. exec: Major components running in the protocol
    i. Client: Bayou Client
    ii. Server: Bayou Server/Replica
    iii. NetNode: Superclass for Client and Server, provides communication infrastructure.
    iv. Master: as required in the handout.

2. Major Design Decisions

a. The Master

As allowed in Piazza post @74, the Master program shares memory with Server/Clients to simplify the coding.

The Master program keeps reference to each node in the system in an array of NetNode (a superclass for Servers and Clients which provide the communication infrastructure). A maximum of 50 nodes can be supported (the maximum index number for node, either Server or Client, is 50. If more nodes are needed, you can simply change the value of constant MAX_NODE in 'util.Constants'.

b. The Client

Client can perform three major operations (PUT, DEL, GET), which is called by the Master directly. The only message it will receive is ClientReplyMsg from Server. For successful Write (PUT, DEL) operation, the Client will update the write_vector in Session Manager and for successful Read (GET) operation, read_vector is updated.

c. The Server

Server has two different constructors. The first one is used to build the very first Server with itself as the Primary. This constructor is used when no other Server is alive in the system. The second one is build for the other Servers with a given server ID (the very first node in the 'curServers' list in Master. Any active server in the system can be used here).

Server handles four categories of messages.

- ClientRequest:
  Upon receiving a request for client, we first check whether the session manager is dominated by the version vector of the Server or not. **If not, one of the session guarantee is violated and no further action is performed**. If the request is a get, the client will get a ERR_DEP. (ERR_DEP has higher priority than ERR_KEY)
  If the request is a write request, the server would generate the Write entry, update the writeLog, update database, update version vector. After sending ack to client, the Server would also gossip with neighbors to inform others of the update.
  If the request is a read request, we get the url from the database a.k.a playList. If the song name is not found in playList, "ERR_KEY" is set to the url value. Then we get the last relevant write of the read from writeLog (to update the read vector of the Session Manager on Client) and forward it to the client.
- Create Messages:
  If it is a CreateMsg (Create request from a new Server), the server generate the new replicaID following the Creation Protocol described in the paper and inform the new Server of its ID.

Upon receiving the CreateReplyMsg, the server setup its ID and timeStamp and starts normal operation. The server will drop any message other than CreateReplyMsg before it sets up.

- Anti-entropy Messages
  We implement a three-way anti-entropy protocol. The sender would first ask the receiver for his version vector with AERqstMsg. Upon receiving the request message, the receiver reply with its version vector in AERqlyMsg. The sender then execute the anti-entropy process as described in the paper and inform the receiver of the Writes it doesn't know about. The receiver update its writeLog and gossip with its neighbours if there is any change made to the writeLog.
- Primary Handoff Msg
  When the primary server retires, it will hand off its duty of the first process knows about it retirement. After writing retire to its own writeLog, the primary server will start gossip will all neighbors and wait for the first AERplyMsg. After receiving AERplyMsg, the primary server sends new writes in a AEMultiAckMsg to the sender of that AERplyMsg and send a PrimaryHandOffMsg. Once a server receives PrimaryHandOffMsg, it has already received all the writes of the primary server, thus it turns itself into the new server and commit all the remaining tentative writes in its writeLog.

3. Other instructions regarding grading tests

a. The source codes and the binaries are in "Bayou.jar".
b. We used "org.apache.commons.codec.binary.Base64" library for base64 encoding/decoding. The library is included in our jar file.