Tribbler Report

Siyu Gao
Zhejin Huang
GroupName: debug

1. TribServer

(1) communication with tribclient

One important function for tribserver is to get the request from tribclient by RPC and fill in the reply parameters. Tribserver does this using go RPC functions.

(2) communication with libstore

The other important function for tribserver is to forward the request from client to libstore. This communication is implemented by defining libstore as an element of tribclient structure. Since both tribserver and libstore are on the same physical machine, there is no need to use RPC from one to the other. After tribserver has got a request from client, it translates this request to database management and call the database manipulation functions on libstore using interface functions. Noted that the parameters are passed using value or pointer directly, for tribserver and libstore share the same memory space.

2. Libstore

(1) data caching

a. cached data type
    i. we use a map structure mapping from key to its value list (**listCache**) to store list contents that are frequently requested by libstore
    ii. we use a map structure mapping from key to its value (**itemCache**) to store item contents that are frequently requested by libstore
    iii. mutex is used to prevent race when accessing these structures
b. lease request and timeout
    i. to store the count of request corresponding to each key and data type. We use a map from key to count for list data type (**listCount**), and a map from key to count for item data type (**itemCount**). We add the count value by one corresponding to list or item when performing request of a key
    ii. we use a epoch timer in another routine to clear the count of each key-count map. All the values are set to 0 at the beginning of each epoch
    iii. when the count reaches the given threshold, request the lease
    iv. when going to request a lease corresponding to a key and a data type, set its count to 0
c. cached data management
    i. when a lease is granted, we start a routine to manage lease timeout, it is implemented by a one-time timer with timeout limit set to the given threshold. Then we store the cache using the map type above
    ii. when a lease is timed out, the manage routine will remove the cache and return

(2) connection caching

a. Connection to storage servers are cached using two structures: a map from key to hostport (**keyHostPortMap**) and a map from hostport to connection

(**hostPortClientMap**). Noted that in the former one, different keys could share the same hostport value, in the latter one, one hostport is corresponding to one connection

b. For convenience, we use a map from virtual ID to hostport (**virtualIDRing**) for each storage server
c. When requesting, we first lookup keyHostPortMap and then lookup hostPortClientMap. If we cannot find it in keyHostPortMap, we cache this map. However, this design makes it easier when a key is not cached but the connection corresponding to the key is already cached. Noted that in this situation, we do not need to setup a new connection, we can just use the cached connection.

(3) request routing

Libstore just uses the given hash function to find the nearest node in virtualIDRing, which is built on the creation of libstore, as mentioned above.

3. Storage Servers

(1) data storage

a. listMap - We use a map structure mapping from key to a list of data including clientID, friends and tribblerID.
b. itemMap - We use a map structure mapping from key to a list of tribblers.
c. revokingMap - We use a map structure as dictionary for storing revoking request in case that there are multiple revoking requests for the same key.
d. leaseMap - We use a map structure mapping from key to a list of leases (leaseContent), which includes the hostport and expected expired time.
e. For all the data structures above, we uses mutex to prevent race conditions when accessing them.

(2) lease management

a. Grant
    i. Requests for granting leases only happens in Get() and Getlist().
    ii. If the lease with the same key is being revoked, storage server cannot grant the lease. Otherwise, storage server will grant the lease and record the lease with hostport to revoke it.
b. Revoke
    i. Requests for revoking leases happens when there is modification in storage server including Delete(), Put(), RemoveFromList() and AppenfToList().
    ii. We firstly insert the revoking request into revokingMap. If there are multiple revoking request for the same key, revokingMapCond and revokingMapLock can prevent race conditions and maintain dealing with all the requests.
    iii. For revoking leases, we wait the return from LeaseCallBack.RevokeLease through rpc or the lease has been expired.

4. Division of work

Checkpoint: Siyu Gao implemented tribserver, Zhejin Huang implemented libstore and storage server

Final: Siyu Gao implemented storage server, Zhejin Huang implemented libstore