CS 677 Lab-2 :Design Document

Group 14: Lurdh Pradeep Reddy Ambati(lambati@umass.edu), Ravi Choudhary(ravichoudhar@umass.edu)

1. Design Considerations:

1.1 Peer Structure:

Each peer will have peer-id, host address, role which states whether it is buyer or seller, an inventory variable(dictionary, it is None for buyer) and a shopping list(a list of items to buy, which is None for seller). And each peer will maintain a list of the lookup requests that pass through them, this list is useful to eliminate the duplicate lookup requests.(Same request coming from different peers.)

A buyer-id or seller-id which is used for lookup or reply, consists of peer-id and host address.

1.2 Choice of Communication:

We have used XMLRPC library in python for RPC communication. It supports only single threaded operation only, so we had to use ThreadingMixIn library from SocketServer to make it multithreaded[1].

2. Implementation Details:

2.1 Leader Election Algorithm:

We have implemented bully algorithm. In case if the multiple elections are happening at the same, in order to prevent leader to send multiple “I won” messages to same peer and also to prevent a peer which already received “Ok” to forward a election message, we have used boolean flags(isElectionRunning, didReceiveOk, and didReceiveWon). isElectionRunning flag can be used by buyers to know whether a re-election has started and they don't do any transactions until a new leader is elected., in case it is set.Implementation details are:

1) Initially Peers are up, random peers start the election by forwarding the election message to higher peers. Each of these peers wait for the “Ok” reply within a time period, if they get then they drop out of election and wait for “I won” message, as soon as they get “I won”, they start trading. If they don’t get any reply from the higher peers within that time period, then they will broadcast the message that the respective peer is leader.

2) Re-election Criterion: There are two criterion. One is when a trader is down, buyer wants to do transaction, then the respective buyer will start the election.(also sends a re-election message to all the peers). Second is, a trader voluntarily gives up position after a fixed number of transactions, in this the last buyer who buys things will start the election, when the trader voluntarily gives up position, election message is not sent to him(he may trade, but will not be the new trader)

2.2 Lamport Clock:

We have implemented lamport clock. We have slightly modified the implementation, so that it suits to the logic that we designed. Events we consider are buying events only from buyers. Once buyers wants to buy a good, it increments its clock and broadcasts the clock to other peers(except trader). When other peers(other buyers and sellers) receive this message, they re-adjust there clocks such that:

max(clock_value, received_clock)

When a peer sends the lookup request to trader, trader checks whether it has served all the requests before that and only serves then or else it is put to wait, until all prior requests to it are served and adjusts its clock according to the above equation.

2.3 Trading Process:

Once the peers receive “I won” message after a election starts, they start trading process. Seller start registering there products with the trader. Buyers wait for a small time to allow the seller to register the goods and start the lookup process for the goods.

A Trader serves the lookup request, by searching for the vendors who sell the respective product and then decrements that product from respective seller and sends message to the buyer and seller that transaction has done(Request will be logged in a file). On the seller side, once it receives this message, if its inventory is empty then it picks up a random products and registers that with trader.

3. Design Trade-offs:

1) We have used few flags to check the status of the election at each peer. Each of the flags are shared resources among the threads, so we have used semaphores to restrict access to one thread at a time, this resulted in increased latency.But consistency in election was achieved.

2) For each transaction, two times we log the same request, one as incomplete and another as complete when the transaction is complete. Again this increased the delay, on the other hand when a trader is done before completing a transaction, new trader can see this and complete that request.

3) When a new leader is elected, he checks the transaction log only. He doesn’t check seller log, as there are few problems with it : one is that when a seller pickups a new good and if this information is not in that log, then the new trader will be unaware of it, rather we thought that when a new trader/leader is elected, all seller will register there goods with new trader.

4. Possible Improvements:

1) Vector Clocks can be improve the program efficiency in terms of the delay, as the vector clocks can result in better consistency/synchronization compared to the lamport clock.

2) We have observed some deadlocks in rare scenarios, which can be removed by better implementation of both bully algorithm and trading process.

5. References:

[1]http://stackoverflow.com/questions/1589150/python-xmlrpc-with-concurrent-requests