Lab 4: Consensus via Paxos

Ankit Goyal
ankit@cs.utexas.edu
CS380L

November 5, 2014

1 Setup

1.1 Hardware

Host Processor: 64 bit 4 core Intel(R) Xeon(R) CPU E3-1270 V2 @ 3.50GHz
Host Memory: 16GB
HyperThreading: Yes
Logical CPUs after Hyperthreading: 8
CPU frequency scaling: Disabled in BIOS (turned off Intel SpeedStep and C-states)

1.2 Software

Host Operating System: Ubuntu with 3.13.0-34-generic 64 bit kernel.

2 Methods Implemented

2.1 `void paxserver::execute_arg(const struct execute_arg& ex_arg)`

This is sent by clients to primary server (to the best of their knowledge). In normal operation, logs the request, update the viewstamp and latest accept in paxos log. Following are some non-trivial cases:

2.1.1 Client sends the message to non-primary server

If the server at the receiving end is not the current primary server in the current view, it drops (`net->drop`) the `execute_arg` message and sends back the `execute_fail` message.

2.1.2 Check if the request (`rid`) has already been received from the given client (`nid`) but not executed

The client may time out and send the request again to the primary server. But primary server should not log the request twice, so it checks in it's log using `Paxlof::find_rid` method. If it's present it simply drops the message.

2.1.3 Check if the request has already been executed and the log truncated

It may happened that the primary has already executed the request and trimmed the log. Since the log has been trimmed the case above will not be able to see that primary has already processed this request. So we search for the given request id(`rid`) and client id(`nid`) in `exec_rid_cache` maintained by paxos server to detect duplicates in different scenarios.

2.2 `void paxserver::replicate_arg(const struct replicate_arg& repl_arg)`

This is sent to replicas by primary to replicate the log. This method checks for duplicates from primary and performs similar checks as performed in the previous method (`paxserver::execute_arg`)

This method also executes the log enteries based on the `latest_seen` field received in the message. It updates `latest_accept` in *paxlog* and `latest_seen` in `vc_state` for the current replica.

At the end it trims the log and sends `replicate_res` to the primary server, acknowledging the receipt of `replicate_arg`.

2.3 `void paxserver::replicate_res(const struct replicate_res& repl_res)`

Primary server records the receipt by incrementing the counter in paxlog in the corresponding tuple (`tup`). Primary now executes the log entries in order if the majority of the servers have acknowledged the logging of the request according to the viewstamp using `next_to_exec` method. Client here updates the `latest_seen` field in `vc_state` so that replicas can be informed of it in the next message.

Primary aggressively trims the log here. Trim function simply checks the entry has been executed and then`trim_front` is called to trim log from front.

Primary checks if the paxlog is empty, and if it is empty, primary sends an `accept_arg` message to all replicas with the updated `latest_seen` value.

2.4 `void paxserver::accept_arg(const struct accept_arg& acc_arg)`

Replicas execute the remaining entries in this method, update their `latest_accept` and `latest_seen`. We also trim the log here.

3 Interesting bug

Timestamp(`ts`) starts from 1 and I was updating the timestamp first and then assigning it to the viewstamp of new message, which basically started my `ts` from 2 and no record was therefore satisfying the `next_to_execute` condition. It lead to no entry being executed, client timeouts and eventually initiated view change.

Time Spent on the lab ≈ 10 hours

4 References

1. Paxos Made Practical - David Mazieres