Distributed Auction System - Report

rono, ghej, luvr @ ITU, Group "G.L.R"

November 25, 2024

Contents

1 Link to Project on GitHub

https://github.com/NewNorthStar/BSDISYS1KU-Assignment-Five

2 Introduction

We have implemented an auction system consisting of one or more server nodes conducting a single auction. Clients may connect to any node to bid on the auction. The solution is implemented in Go and communicates using gRPC.

- The server nodes act as a single leader system. Followers will forward new bids to the leader. The leader then updates any followers before the method returns. This ensures linearizable state changes as long as the auction is live with atleast one node.
- When a follower discovers that the leader is unreachable, it will call for an election. This is implemented using the Bully election algorithm, with priority in particular given to the most up-to-date node.
- If a client loses connection to the auction, it will try reconnecting to another known node. Clients know the nodes available their registration, or at their last bid.

3 Architecture

Clients place bids at the auction using the `PutBid(...)` rpc. This may be called on any node. If a follower node receives this call, it will forward the call to the leader using a client instance. This is also when a follower may detect the failure of the leader.

The `GetLot(...)`, `GetAuctionStatus(...)` and `GetDiscovery(...)` rpc's are used to pull information from the auction. These are for use by clients, but follower nodes also use them when registering with the leader.

`Ping(...)` is used by clients to find a new connection to the auction, should the first one fail.

`Register(...)` is used by follower nodes to register with the leader. The successful connection is confirmed by the leader by sending the first `UpdateNode` call.

`UpdateNode(...)` is used by the leader to push auction state changes to to followers. This is called as a side-effect of the `PutBid` and `Register` rpc calls.

Should the leader node fail, then the `Election(...)` and `Coordinator(...)` calls are used to elevate a follower to lead the auction.

4 Correctness

4.1 Argument 1

Argue whether your implementation satisfies linearisability or sequential consistency. In order to do so, first, you must state precisely what such property is.

We have implemented single-leader synchronous replication. In order to have a fair auction, we decided that the placement of bids should satisfy linearisability. That is:

1. The leader is at all times the single correct copy of the auction. Followers only provide standby redundancy and bid forwarding.
2. The interleaving of bids is consistent with who is first able to secure the critical section with the leader in real time.

However:

1. Other calls such as `GetLot`, `GetAuctionStatus` and `GetDiscovery` are more leniently implemented. Followers are allowed to resolve these calls with their current information.
2. We have not locked `PutBid` for other followers during the election. This potentially allows another bid to sneak in front of the election triggering bid. This could perhaps be resolved with a mutual exclusion algorithm as seen in hand-in 4 with additional timestamps.

4.2 Argument 2

An argument that your protocol is correct in the absence and the presence of failures.

Given reliable message transport, the system will work consistently through the failure of any replica but one.

The leader maintains a timestamp that increments with each bid processed. This is part of the `UpdateNode` call made to follower nodes. When `PutBid` returns for a client, the new bid will have been replicated across all reachable non-failed followers. We know of one current limitation in our solution: Should the leader fail during a synchronous replication, the bid may persist after an election at other nodes. In this case the client will retry placing the bid with a follower node, and will then receive the correct reply after an election.

A Follower will trigger an election if the leader fails to process a forwarded bid. Elections are held using the Bully algorithm, with followers multi-casting their current stats. The node with the highest bidding timestamp takes priority. The election outcome elevates one of the followers to be leader, unseating any existing leader.

Assuming that leaders fail cleanly, it is not possible to reconnect to the auction service as a leader. Nodes can only register with the auction leader as followers. Without clean failures we would have to implement additional measures to avoid split-brain scenarios.