Mandatory Activity 5

Markus Sværke Staael - msvs@itu.dk

Patrick Shen - pash@itu.dk

Nicky Chengde Ye - niye@itu.dk

Git repository:

https://github.com/BDIS2024/Distributed_Systems/tree/1526696ee9f073b699629f69c06301dc81df5565/Assignment_5

Introduction

This distributed auction implementation uses active replication to become crash tolerant. This means that the demonstration is to have 3 servers serving any clients at the same time. When any of the servers fail the other servers will be there to serve the client.

This implementation assumes that the connection is stable and no inconsistencies will occur between the servers, and the client will therefore not check for them nor correct them.

Architecture.

This implementation has two main components: Client and server, respectively they represent an entity trying to interact with the bidding system, and the bidding system itself. Clients can make two kinds of requests to the server, a bid request and a result request.

Bid requests are sent with a sender, value and physical timestamp. Somewhat obviously the server over time records whoever has the largest bid and lets that entity be the “winner” at the end of the auction.
Whenever a server receives its first bid it will begin the auction process. The timeframe in which the clients have to bid in is based on the physical timestamp given by the bid request, that way we easily ensure consistency between the servers without making them have to communicate with each other.
Every bid request is returned with an acknowledgement that communicates if the bid was a success or not.

Result requests are the way for the clients to know what state the auction is in. If the auction is over the client will end its process.

Clients keep track of what of the hard coded servers they can connect to. When a connected server cannot be accessed it will assume it has crashed. Whenever the client has no more servers to communicate with it will end its process.

An observation to make is that all servers are “synchronized” because we have no connectivity issues on a local machine. Therefore a client has no need to check for inconsistencies in the answers from the servers.

- Correctness 1.

The auction system satisfies sequential consistency by using active replication. Sequential consistency has the purpose of creating an illusion of there only being a single copy of a backend serving a client. Meaning that any sequence of requests to the auction system will result in the same result, regardless of the client.

Active replication means that all clients interact with a frontend that sends the requests to all replica managers. The physical time for these operations will therefore happen at different times because the operations are multicast to different servers that handle the operations locally. However, the sequence of events will be kept in the same order across the different servers.

- Correctness 2.

Active replication is more suitable for an auction system because it is a time sensitive procedure, which requires minimum latency and constant uptime. In the event of a server crash, the clients will continue to interact with the other servers normally without having to wait for a backup server to be promoted to a primary server. By having multiple servers actively running and receiving the same calls from the clients, the sequence of events are maintained and constantly updated. However, our protocol does not handle data corruption as no form of consensus is implemented. In the absence of failure, the protocol will proceed as expected and function normally with all replicas active.