IT UNIVERSITY OF CPH

Mandatory Activity 5: Distributed Auction System

Distributed Systems, BSc (Autumn 2025)

BSDISYS1KU

Group:

Anders Grangaard Jensen Agje@itu.dk

Mathias Vestergaard Djurhuus mavd@itu.dk

Theodor Monberg tmon@itu.dk

IT University of Copenhagen

Fall 2025

Table of contents -Mandatory-Activity-5

1. Introduction

This project implements a distributed auction service that tolerates one crash failure. The system supports two operations:

- `Bid(amount):` submit a new bid
- `Result():` retrieve current auction state

The auction runs with exactly one leader and multiple followers.
Clients may contact any node, but only the leader accepts Bid requests.
Replication is handled using a simple primary-backup protocol and a majority commit rule, ensuring that valid bids survive the crash of one follower.

The auction is implemented in Go using gRPC and Protocol Buffers for communication. Replication between nodes is explicit and observable in the logs provided. All valid bids must increase strictly compared to both the bidder's previous bid and the global highest bid. The auction closes automatically 100 seconds after startup, after which bids are rejected and only the final result is returned.

Repository:

https://github.com/Djurhuus02/-Mandatory-Activity-5.git

2. Architecture

The system follows a primary-backup design where a single leader serializes all bid requests. When the leader receives a bid, it verifies that the auction is open and that the bid amount satisfies the auction constraints. Once a bid is validated, the leader replicates it to all followers through an AppendBid RPC. Followers do not validate the bid; they

simply apply the update in the order in which they receive it.

A bid is considered committed when a majority of nodes specifically, the leader and at least one follower acknowledge the replicated update. Only at this point does the leader update its local state and return a success response to the client.
The behavior is clearly visible in the logs. (S1)
Leader:

```
[l1|leader] received Bid: auction=a1 bidder=A amount=50
[l1|leader] replication result: ok=true msg=majority reached (3/3)
[l1|leader] bid committed: highest_bid=50 highest_bidder=A
```

The followers apply the update accordingly:

```
[f1|follower] AppendBid from leader: auction=a1 bidder=A amount=50
[f1|follower] state updated: highest_bid=50 highest_bidder=A
```

This confirms that the replication procedure works correctly under normal operation.

Fault Model

- Fail-stop crashes only.
- Reliable, ordered network.
- At most **one follower** may crash.
- Leader election is out of scope.

If the leader crashes, no further bids can be submitted, but the replicated state remains safe on the followers. If a follower crashes, the leader retains liveness because the remaining follower still forms a majority with the leader.

3. Correctness 1 - Consistency Argument

The system satisfies sequential consistency, not linearizability. The reason is that followers momentarily lag behind the leader. A follower may respond to `Result()` with a slightly older state if it has not yet received the most recent AppendBid message. This small lag violates the real-time ordering requirement necessary for linearizability.

Despite this, sequential consistency holds. All writes pass through a single leader, which serializes them using a mutex, ensuring a total global order. Followers apply updates in exactly the same order as they are received, preserving this sequence.

Rejected bids are never inserted into the history, which is visible in Scenario 2:

```
[l1|leader] >>> Bid RPC reached: auction=a1 bidder=A amount=50
[l1|leader] received Bid: auction=a1 bidder=A amount=50
[l1|leader] replication result: ok=true msg=majority reached (3/3)
[l1|leader] bid committed: highest_bid=50 highest_bidder=A
[l1|leader] >>> Bid RPC reached: auction=a1 bidder=B amount=30
[l1|leader] received Bid: auction=a1 bidder=B amount=30
```

Since the second bid is invalid, it does not propagate to followers, which remain at:

```
[f1|follower] AppendBid from leader: auction=a1 bidder=A amount=50
[f1|follower] state updated: highest_bid=50 highest_bidder=A
```

This confirms that the global history is preserved across all nodes and that only valid, increasing bids form the shared state.

4. Correctness 2 - Correctness With and Without Failures:

All nodes apply the same updates.

Scenario 3, the system demonstrates correct behaviour under a single crash failure.

In this scenario, follower f2 is offline. The leader receives a new bid of 120 from bidder B and attempts replication. Since one follower is unavailable, the leader replicates only to f1, achieving majority (2 out of 3 nodes):

```
[l1|leader] replication result: ok=true msg=majority reached (2/3)
[l1|leader] bid committed: highest_bid=120 highest_bidder=B
```

Follower f1 applies the update:

```
[f1|follower] state updated: highest_bid=120 highest_bidder=B
```

Follower f2 remains stale but harmless. It cannot accept client operations, cannot become leader, and cannot break the global order established by the leader. The system therefore continues operating correctly despite the crash.

`Result()` calls on the leader and surviving follower return the same committed auction state, demonstrating both safety and liveness.

5. Appendix

The accompanying logs provide full evidence of system correctness:

- **Scenario 1:** Successful replication and majority commit under normal conditions.
- **Scenario 2:** Rejection of invalid bid, with followers receiving no updates.
- **Scenario 3:** Majority commit with one follower offline, demonstrating fault tolerance.

All referenced logs appear in the project under logs/scenario1, logs/scenario2(low bid), and logs/scenario3(crash).