Università
della
Svizzera
italiana

**Facoltà
di scienze
informatiche**

Distributed Algorithms **Autumn 2025**

Prof. Fernando Pedone, Lorenzo Martignetti

Project

Description

Paxos is a protocol used to solve consensus in asynchronous systems. Simply put, consensus can be used by a set of processes that need to agree on a single value. More commonly though, processes need to agree on a sequence of totally ordered values - a problem known as *atomic broadcast*. In this project, you'll use the Paxos protocol to implement atomic broadcast.

In your implementation, four *roles* need to be provided:

- *clients:* submit values to proposers
- *proposers:* coordinate Paxos rounds to propose values to be decided
- *acceptors:* Paxos acceptors
- *learners:* learn about the sequence of values as they are decided

Your protocol should always guarantee *safety*. To guarantee liveness, in a complete Paxos implementation, one of the proposers should be elected as the leader. For simplicity, in this project, you will not be asked to implement a leader election oracle. However, you should not make any assumptions about which proposer is the leader, and you should support more than one proposer. Therefore, if two proposers are proposing in parallel and preventing each other from executing Phase 2 of the protocol, your implementation should ensure that they will keep trying until one of them hopefully succeeds.

Assumptions

You may assume crash failures. That is, processes fail by halting and do not recover. This allows the following simplifications:

- no need to implement a recovery procedure for acceptors or learners
- all state can be kept in memory - no need to use stable storage

Implementation

You should implement your solution respecting the following constraints:

- Your Paxos implementation must be based on *IP multicast* only. You will support four different multicast groups, one for each role of the protocol.
- Your solution should be implemented inside the `src` directory (containing the source code). The other scripts are used to run the implementation, perform checks on the output and cleanup, generate the plots from the collected data. In case you want to modify them for testing purposes, you should make sure that your final implementation works with the original scripts.
- You are not allowed to introduce synchrony in the system, i.e., you are not allowed to use `time.sleep()` or similar procedures.

There will be 3 *milestones* for the implementation:

1. *Synod algorithm*: the basic version of Paxos seen in class. It supports the decision of a single value.
2. *MultiPaxos*: an extension of the Synod algorithm to support atomic broadcast. It supports the decision of multiple values in the same total order.
3. *Optimizations*: we will focus on 3 of them:
   - One communication step is saved by allowing acceptors to send the `PH2B` messages directly to the learners.
   - Two more communication steps are saved on the critical path by the proposers performing `PHASE1` before receiving the value from the clients.
   - With batching, more values can be decided in a single instance of the Synod algorithm.

Your implementation should guarantee the following:

- message loss or processes crashing should never violate safety (total order, agreement or integrity)
- if a majority of acceptors are killed, no progress should be made (asynchronous consensus assumption)
- learning values must be possible if there are a majority of acceptors and 1 of each other role (no crashes or message loss)

Here are some tests you may perform on your solution:

1. Proposing 100 values per client (2 clients, 2 proposers, 3 acceptors, 2 learners). Check that learners learn values in total order. Check that values that were proposed were learned. Repeat with 1000 and 10000 values per client.
2. Repeat test 1 with only 2 acceptors.
3. Repeat test 1 with only 1 acceptor.
4. Repeat test 1 with some % of message loss.
5. Learners catch up. Start 3 acceptors, 1 proposer, and 1 learner. Clients start proposing values. After some time, start an additional learner. Check that learners learn values in total order. Specifically, the newer learner needs to learn previous values.

Please check the `README` of the project for further information.

Deliverables

Your submissions should include the content of the `src` folder only, as we assume all other scripts are left unchanged. In case you needed to modify the scripts, please add a report to the submission that explains why and how you modified them. In this case, include the modified scripts in the submission too.

You will have to submit the three milestones separately:

- **29th October 2025 @ 23:59**: Synod algorithm
- **12th November 2025 @ 23:59**: MultiPaxos
- **10th December 2025 @ 23:59**: Final submission with optimizations

At the end of the project we expect a working system, that can be easily tested on a cluster of machines.

Presentations

Each group will have 5 minutes to present the implementation. The implementation should include:

- A general overview of the system design
- The main difficulties you encountered, and how you addressed them
- The final takeaways from the project

Each presentation will be followed by a few questions. We expect all people in the group to be able to answer them. The grade of your project will also consider the source code correctness, completeness, readability and performance.