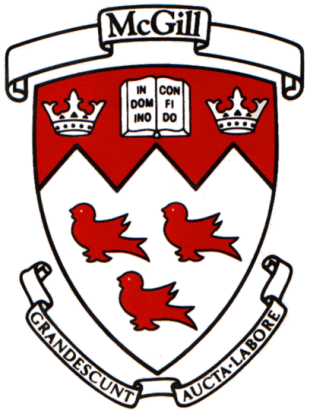

Multi-Paxos Consensus Algorithm

COMP 512 - Distributed Systems
Programming Assignment II

Sinan Can Gülan
Ege Güney
November 6, 2023

Contents

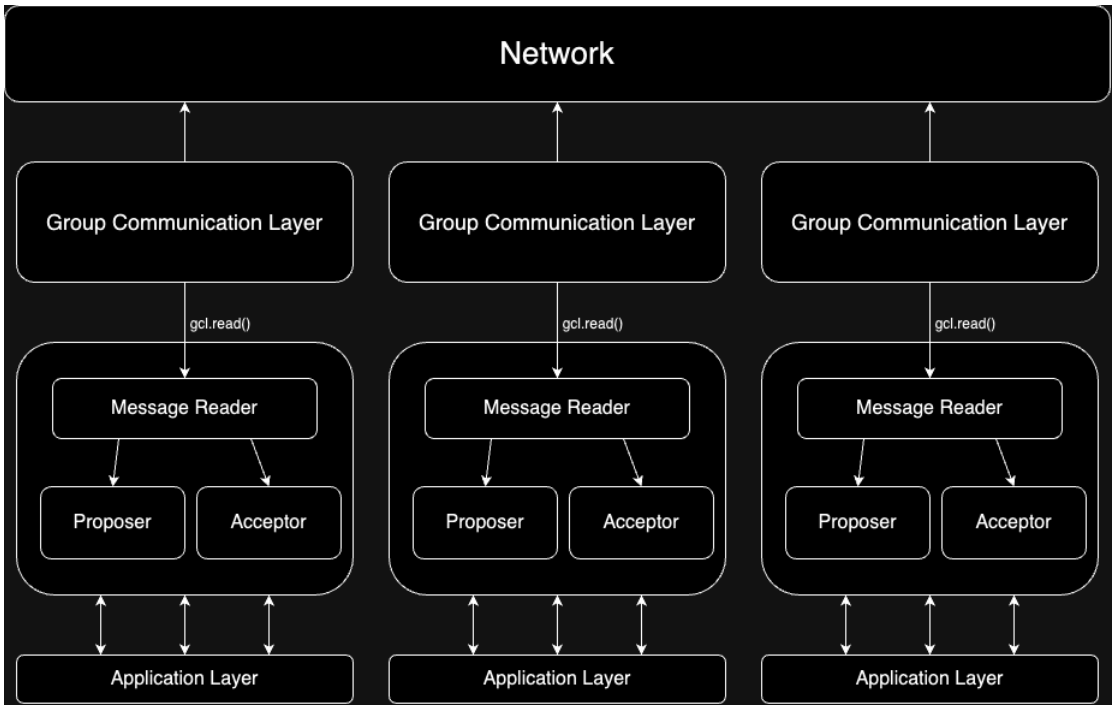

Figure 1: Implementation architecture for Paxos with $n = 3$.

1 Algorithm Objective

Given n client processes $\{p_1, ..., p_n\}$, each process has a list of values $X = \{x_1, x_2, ..., x_m\}$, which they continually update by proposing and accepting values. The Multi-Paxos algorithm guarantees that each of these lists are identical. In the context of a game, this means each client sees the same sequence of actions and consequently the same map.

2 Algorithm & Implementation Architecture

For each process, the Paxos Middleware sits between the Application Layer (AL) and the Group Communication Layer (GCL). Each Paxos instance has three threads: Acceptor, Proposer and Message Reader. The message reader continually reads from the GCL and redirects the messages to either the Acceptor or Proposer, depending on the type of message. The Acceptor and Proposer algorithms follow closely their formal definitions, details are omitted for brevity.

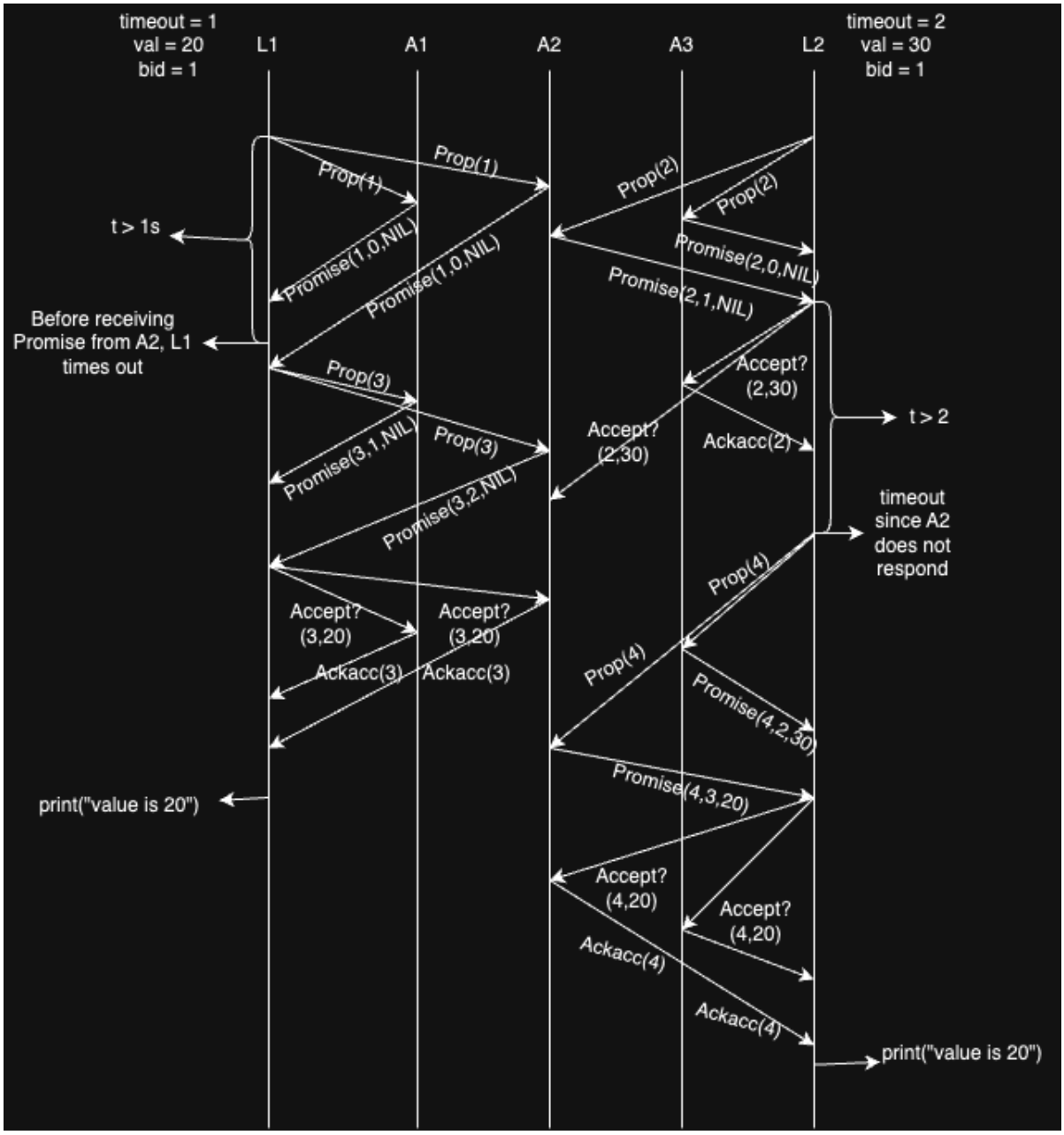

Figure 2: Example Paxos execution.

3 Implementation Guarantees

3.1 Integrity

Our implementation guarantees integrity in the sense that: values submitted by the AL are never modified or lost. Non-modification of course is guaranteed by default; we will describe briefly how we guarantee the loss of values.

When simultaneous values are proposed by different processes, one of the proposers won't be able to get a majority of promises. In order to ensure that the value is not lost, the proposer adds the value to an internal queue, let's say V. The next time the proposer is to suggest a value, it prioritizes values in V over new values submitted by the AL.

3.2 Uniformity

Our implementation also guarantees uniformity, which we define as follows: uniformity is guaranteed when no subset of processes dominates the confirmation of values. This means that, over a sufficiently long time-frame, the distribution of player id's of the moves delivered to the AL is uniform.

This is achieved through an "insistance mechanism". If a proposer is unable to receive a majority in a given round, in the next round it will increase its ballot id by the number of processes $bid_p + = n$. By not re-setting these values, we ensure that proposers that didn't get to push their values in previous rounds will have a higher likelihood to be accepted in future rounds.

3.3 Inclusion

In the Paxos algorithm, only a majority is required for the confirmation of a given value. In order to ensure inclusion of processes outside the majority, confirmation messages are sent to everyone, and the values are piggybacked onto the messages.

3.4 Fault-Tolerance

Our set-up allows for seamless fault-tolerance. At any given point, if a process fails, as long as there's still a majority of processes running, the rest of the processes will continue execution. The acceptors continue responding to other proposers even if they've promised to one, thus a different proposer can take over whenever the current leader fails.

4 Contributions

Both participants have been equally involved in the delivery of this programming assignment. Most of the implementation has been done from a single computer with both participants present in the room.

Only some specific parts of the project has been solely performed by one person:

- Ege Güney: Failchecks, Performance Analysis Experiments
- Sinan Can Gülan: Architecture & Performance Analysis Report

Discussions and meetings have been carried out in person, with a total of 6 meetings for the implementation.