CSE-5306
DISTRIBUTED SYSTEMS

PROJECT – 3
Date: 11/30/2023

Submitted by:

Name: Mohammad Shahedur Rahman ID: 1002157980

Name: Mehzad Hossain Setu ID:1002138330

I have neither given nor received unauthorized assistance on this work.

Signed:

Signed: Mohammad Shahedur Rahman Date: 11/30/2023
Signed: Mehzad Hossain Setu Date: 11/30/2023

Table of Contents

Introduction 3
Assignment 1 4
Answer to the Question-01 4
Answer to the Question-02 7
References (for questions 1 and 2) 8
Assignment 2 9
Basic Paxos Protocol Implementation 9
Evaluation 10
Scenario (a): The previous value is already chosen. 10
Scenario (b): Previous value not chosen, but new proposer sees it 10
Scenario (c): Previous value not chosen, new proposer doesn't see it 10
Centralized Coordinator 11
Conclusion 11

Introduction

The folders contain all the source codes of the programs separated by Assignment name and title. The Readme files in each folder contain instructions on how to run the programs for ready reference.

In this programming project, we have practiced programming related to basic single decree (value) Paxos protocol, and evaluated our implementation of the Paxos protocol by studying whether it can correctly choose a value for the three different scenarios:

(a) Previous value is already chosen,
(b) Previous value not chosen, but new proposer sees it,
(c) Previous value not chosen, new proposer doesn't see it

Moreover, to evaluate our Paxos protocol in a controlled environment given the uncertainties in message passing, consider implementing a centralized coordinator to collect from individual proposers.

Although. we used the 5 proposers and acceptors configuration for evaluation, our Paxos implementation works for any number of proposers/acceptors.

We have used the below-mentioned tools and programming language for project 3.

1. Oracle VM Box 7.4.10
2. Ubuntu 22.04.3 LTS
3. Python 3.11
4. Other relevant packages.

Assignment 1

Answer to the Question-01

How does the Multi – Paxos protocol ensure multiple nodes to agree on a consistent ordering of a sequence of values and how is it different from running the Basic Paxos protocol for multiple rounds?

In distributed systems, which are more and more crucial in today's interconnected world, distributed consensus is a critical issue. When distributed services are replicated for fault tolerance, consensus is essential because non-faulty replicas must concur on the system's current state or the order in which activities have been carried out. Unfortunately, consensus is difficult when processes or communication channels may fail. Paxos, is a crucial algorithm for getting a group of distributed processes to agree on a single value or a sequence of values. Basic Paxos is for agreeing on a single value, such as whether to commit a database transaction. Multi – Paxos is for agreeing on a continuing sequence of values, for example, a stream of commands to execute.

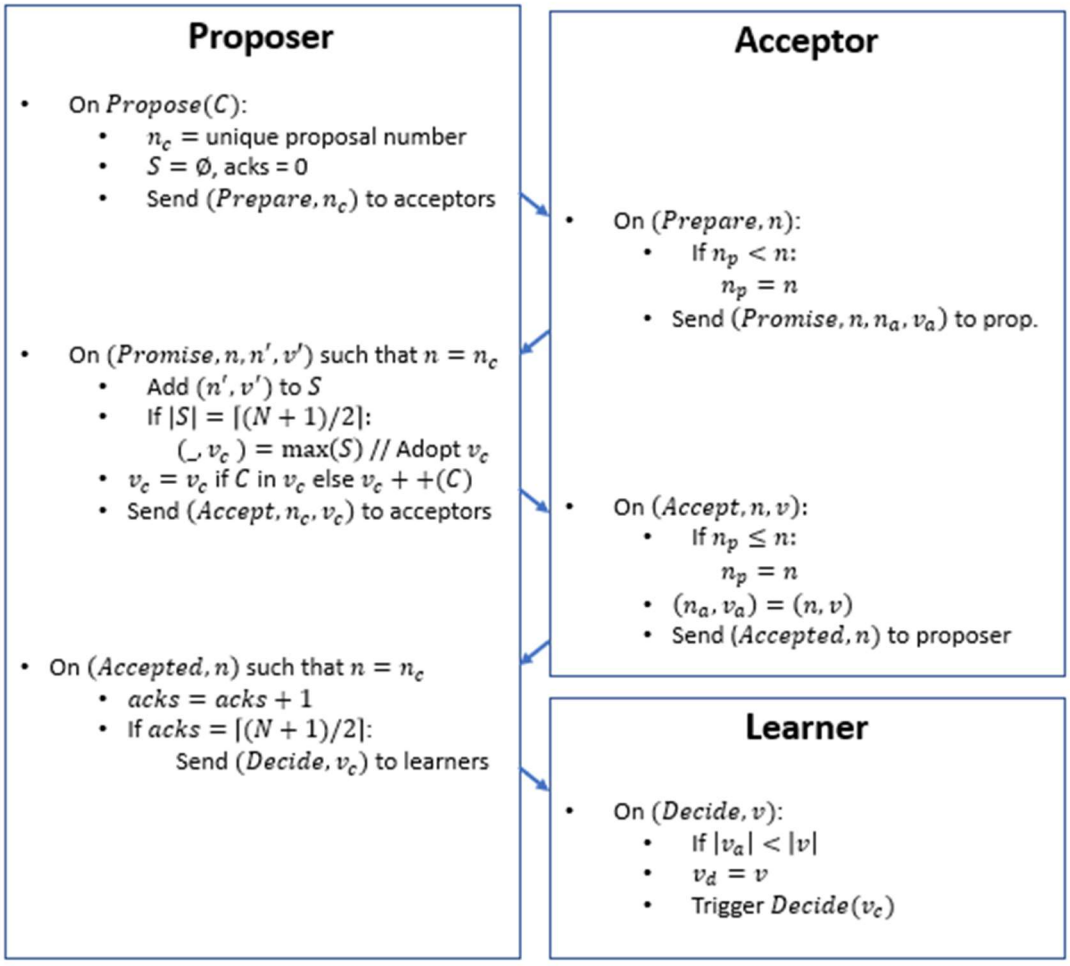
Basic Paxos defines several different roles which must cooperate to achieve consensus; they interact to collectively agree on a proposed value. The three roles are:

- Proposers, who receive requests (values) from clients and try to convince acceptors to accept the value they propose.
- Acceptors, who accept certain proposed values from proposers and let proposers know whether a different value was accepted. A response from an acceptor represents a vote for a particular proposal.
- Learners, who announce the outcome to all participating nodes.

In Basic Paxos, the Proposer sends a command C to propose a value for acceptance. It picks a unique number, it has a set that keeps track of the response of the acceptors and it sends a prepare message with its unique proposal number to all acceptors. On the Acceptor side, on receiving the prepare, the Acceptor looks and sees if the proposal number it got is higher than its current promise. If not, then it sets its promised value to n and sends a promise back to the Proposer not to accept any values with a proposal number which is less than what it has received from that Proposer. The Acceptor also sends the value and the proposal number which is last accepted, if any. Once the Proposer receives this promise for the current proposal number from the majority of Acceptors, if in the promises, there is a higher round number than the one it originally proposed then it adopts that value. Otherwise it proposes its own value and sends an accept message to all Acceptors. At the Acceptor side, if it didn't promise to accept a proposal higher than this number, it accepts the proposal, updates its round number, updates its most recent accepted number and the value it has accepted and sends the accept message to the Proposer. When the Proposer gets a majority of acknowledgement messages from the Acceptors, then it means that its value has been chosen and so it sends this to the Learners. The Learners update their value and triggers the decide event. If the Acceptors do not accept the proposed

value because a higher value was already selected, then the Proposer has to go back, pick a higher round number and try the whole process again.

We want to develop the $\mathrm{Multi-Paxos}$ algorithm, by making minimal modification to $\mathrm{Basic\ Paxos}$. The flow of the algorithm looks like this:



The main difference between $\mathrm{Basic\ Paxos}$ and $\mathrm{Multi-Paxos}$ are:

- In the adopted value at the proposer when it receives the promise message. When we have an adopted value v, so if the command C is in the sequence, we cannot change it, so the value we are going to send is the same as before. But if our command is not in that sequence that we got in the promise, then we can extend the sequence and then send that sequence to the acceptors.
- The second change is at the learner. When the learners get a sequence, it looks at the lengths of the sequence. If the sequence is longer than the previously decided sequence, then it updates the decided sequence and then triggers the decide.

In $\mathrm{Multi-Paxos}$, values are not single values any more, values are sequences here. Initially every acceptor accepts an empty sequence. After adopting a value with the highest proposal number, the proposer is allowed to extend the sequence with non-duplicate command(s) and

propose that extended sequence. When a learner receives a decide request with the newly extended sequence value, it will choose the new value, if that sequence is longer than the previously decided sequence.

Paxos is an algorithm that can choose the sequences of values among multiple ones and ensures the following two properties:

- Safety
    - Only a single value could be chosen because the machines all decide on the value to be chosen
    - Only the chosen value would be learned
- Liveness
    - Some proposed value is eventually chosen
    - Learners would eventually learn the chosen value

Thus, it can ensure a consistent ordering of the sequences of values.

Answer to the Question-02

What are the performance and scalability issues of the Multi-Paxos protocol? List all the references you used in your answer.

The procedure for $\text{Multi} - \text{Paxos described}$ above is not efficient. This is because of several issues.

- With multiple concurrent proposers, conflicts and restarts are likely
- Every proposal takes two round trips – a prepare round and an accept round.
- We cannot take another proposal until we finish the preparation round which means proposals are not pipelined.
- Entire sequences are sent back and forth.

These issues must be optimized to improve the efficiency of the procedure to improve performance and scalability.
The first thing to be done to reduce conflicts is to have one proposer at a time. This is done by picking a leader which after the first Prepare, only performs Accepts until it is aborted. A proposer will send a Prepare message and when it gets a Promise, it will send multiple Accepts and will get back multiple Accepted and Decide messages. The Proposer issues multiple proposals guaranteeing that each time it sends a proposal, it sends a longer sequence than the one before. It does not need to wait for one proposal to be chosen before sending the next one. The benefits are:

- Proposer does $\text{Prepare}(n)$ before the first $\text{Accept}(n, v)$. After that only one round-trip is needed to decide on an extension of value v, as long as the round is not aborted.
- Allows multiple outstanding accept requests which adds pipelining to the process. Therefore, it lowers propose to decide latency for multiple proposals.

The next thing we want to do is to avoid sequences. So far, the messages contain entire sequences, which can be very large. We want to eliminate sending redundant information. The proposer needs to include the length of its decided sequence in the prepare message it sends out at first. When the Acceptor sends the promise message back to the Proposer, it can just send the difference between what the Acceptor already knows and the decided sequence in the proposal and thus it can be optimized. Secondly, if we look at the accept message which is sent by the Proposer after it has received the promises, the accept message now only needs to carry the delta between the new extended sequence and what each acceptor already knows. We don't need to send the whole sequence again. Also, in the case of the Decide message, we need to carry the extended part only. Making these changes helps avoid sending the whole sequence back and forth. For this to work, Proposers and Acceptors needs to know what is decided.

References (for questions 1 and 2)

1. Lamport, L. (1998). "The Part-Time Parliament." ACM Transactions on Computer Systems (TOCS), 16(2), 133-169.
2. Ongaro, D., & Ousterhout, J. (2014). "In Search of an Understandable Consensus Algorithm." USENIX Annual Technical Conference.
3. Chand, Saksham, Yanhong A. Liu, and Scott D. Stoller. "Formal Verification of Multi-Paxos for Distributed Consensus," 9995:119–36, 2016. https://doi.org/10.1007/978-3-319-48989-6_8.
4. Alastair. "Paxos Consensus Algorithm." ScyllaDB. Accessed December 6, 2022. https://www.scylladb.com/glossary/paxos-consensus-algorithm/.
5. "Understanding Paxos." Accessed December 7, 2022. https://people.cs.rutgers.edu/~pxk/417/notes/paxos.html.
6. Howard, H., & Malkhi, D. (2003). "Scalable Agreement: Toward Ordering as a Service." Proceedings of the 6th ACM/IFIP/USENIX International Conference on Middleware.
7. Lecture 8. Distributed Systems (Dr. Jia Rao) https://ranger.uta.edu/~jrao/CSE5306/fall2023/index.html
8. Lecture 11. Unit 2 From Paxos to Multi-Paxos, 2014. https://www.youtube.com/watch?v=-Bl5GleEN5s.
9. Lecture 11. Unit 4 Mutli-Paxos Prepare Once Pipeline Accepts, 2014. https://www.youtube.com/watch?v=iOblL-SdVMI.
10. Lecture 11. Unit 5. Multi-Paxos Trim Sequences, 2014. https://www.youtube.com/watch?v=Lpb2hM0K-dc.

Assignment 2

Basic Paxos Protocol Implementation

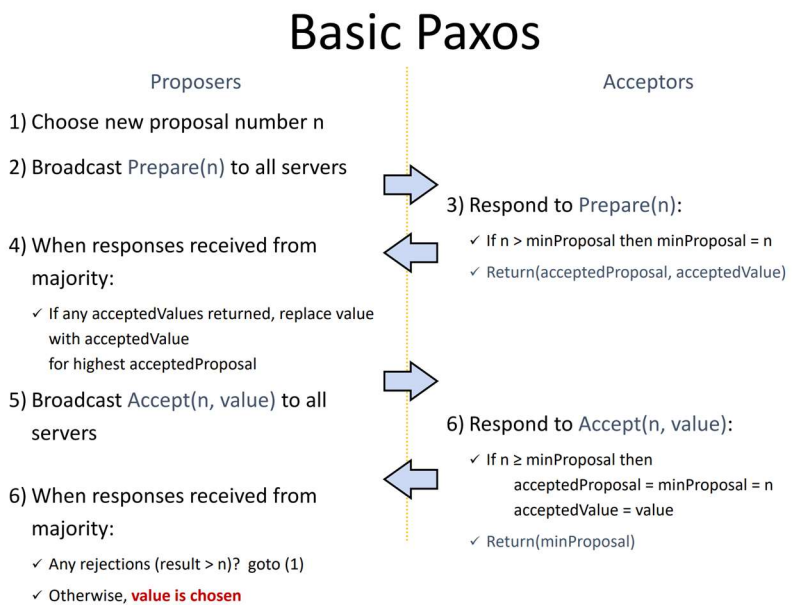
The Paxos protocol consists of two phases:

Phase 1: broadcast Prepare RPCs

- The proposer generates a unique proposal number (n) and sends a prepared request to all acceptors.
- Each acceptor responds with a prepared response containing the highest proposal number (n') it has seen and the value (v') associated with that proposal number.
- The proposer collects and prepares responses from a majority of acceptors. If a majority of acceptors have not responded, the proposer aborts the preparation phase and restarts with a new proposal number.

Phase 2: broadcast Accept RPCs

- If the proposer has received prepared responses from a majority of acceptors, and all responses have the same value (v'), then the proposer sends an accept request to all acceptors with the value (v').
- Each acceptor responds with an accept response if it has not already accepted a proposal with a higher proposal number.
- The proposer collects accept responses from a majority of acceptors. If a majority of acceptors have not responded, the proposer aborts the accept phase and restarts with a new proposal number.



Once the proposer has received accepted responses from most acceptors, it broadcasts a learn message containing the chosen value (v') to all nodes.

Evaluation

To evaluate the Paxos protocol, we can simulate the three different scenarios mentioned in the problem statement:

Scenario (a): The previous value is already chosen.

- In this scenario, a previous proposer has already chosen a value and all acceptors have accepted that value.
- When a new proposer enters the protocol, it will receive prepared responses from all acceptors with the previously chosen value.
- The new proposer will then send an accept request with the previously chosen value, and all acceptors will accept it.
- Therefore, the protocol will correctly choose the previously chosen value.

Scenario (b): Previous value not chosen, but new proposer sees it

- In this scenario, a previous proposer has not yet chosen a value, but the new proposer sees the previously proposed value from some acceptors.
- The new proposer will still send out prepare requests to all acceptors, but only the acceptors that have seen the previously proposed value will respond with it.
- Since the new proposer needs a majority of acceptors to agree on a value, it will not be able to choose the previously proposed value.
- The new proposer will then generate a new proposal number and send out another round of preparation requests.
- Eventually, a majority of acceptors will agree on a value, and the protocol will correctly choose that value.

Scenario (c): Previous value not chosen, new proposer doesn’t see it

- In this scenario, a previous proposer has not yet chosen a value, and the new proposer does not see any previously proposed values from any acceptors.
- The new proposer will still send out prepare requests to all acceptors, and all acceptors will respond with their highest proposal number and value.
- Since the new proposer has generated a unique proposal number, it will be able to choose its own proposed value.
- The new proposer will then send out accept requests with its proposed value, and all acceptors will accept it.
- Therefore, the protocol will correctly choose the new proposer's value.

Centralized Coordinator

To further evaluate the Paxos protocol in a controlled environment, we can implement a centralized coordinator to collect votes from individual proposers and send them to acceptors in a specific order. This allows us to test the protocol in various scenarios and observe its behavior under different conditions.

Conclusion

The Paxos protocol is a powerful tool for achieving consensus in distributed systems. It is a complex protocol, but it has been proven to be correct and reliable. The implementation and evaluation presented here demonstrate the effectiveness of Paxos in handling various scenarios and ensuring data consistency in distributed environments.