Comparison of Different Implementations of Paxos for Distributed Database Usage

Proposal By

Hanzi Li
Shuang Su
Xiaoyu Li

12/08/2015
COEN 283
Santa Clara University

Abstract

This study aims to simulate the basic Paxos algorithm database usage by implementing all the three roles of the Paxos algorithm: Proposer, Acceptor and Learner. In our implementation, we have an interface for read and write operations to simulate client request. To simplify the implementation, we store only one value in the database. To read to is get the value stored in the database, while to write is to overwrite the value stored in the database. This study also compares two different implementations of Paxos algorithm. In the first algorithm, an acceptor will send acknowledge response to all the learners when it accepts a proposal, thus the total message flow in the learning phase equals to the product of number of acceptors and number of learners. In the second algorithm, an acceptor will only send acknowledge response to a distinguished learner. When the distinguished learner has learned the chosen value, then it will send a broadcast to all the other learners in the system. Thus the total message flow in the learning phase equals to the sum of number of acceptors and number of learners. We assume the second algorithm will be more efficient. We also run a series of test cases to compare the running time of read/write operations in these two implementations. Furthermore, we run a series of failure test cases to simulate proposer/leader fail, acceptor/voter fail. Also, the Paxos protocol also assumes the processes could fail in any state in the processing of reaching consensus. Therefore, we also simulate failures in different state, for instance, the prepare request state, the prepare response state, accept request state, and accept response state.

The results of our study all confirm with our hypothesis. Implementation 2 is more efficient than implementation 1. For recommendations for future study, we would recommend using the distinguished learning in the learning phase to reduced the number

of messages needed to be sent in the learning phase. Also, future study may try to implement this Paxos algorithm in another order. When we do read first, then write operations, the read takes more time then write. That's contradictory to our expectations. We infer this abnormal result maybe due to the optimization of Java when it runs operations sequentially.

1. Introduction

1.1. Objective

There are two objectives of this study. First, it intends to do a simulation of the basic Paxos algorithm for general application by simulating how the roles of proposer, acceptor and learner coordinate to reach consensus to client request in an unreliable network and unreliable server environment. Second, this study aims to compare two different implementations of Paxos algorithm in the learning phase. The first algorithm of learning a chosen value is to have each acceptor respond to all learners when it accepts an proposal. The advantage of this algorithm allows each learner to learn which value has been accepted as soon as possible. The disadvantage of this algorithm is that it requires each acceptor to send message to each learner, therefore the total of responses equals to the product of the number of proposer and the number of learners. In the second algorithm of learning, we have all the acceptors sending response message to a distinguished learner only. Then when the distinguished learner learned this value then it can inform other learners about the chosen value. In the second algorithm, the total number of message response delivered in the learning phase equals to the sum of the number of proposer and the number of learners. Thus the advantage of the second algorithm is that the message traffic will be less in the learning phase. However, if the distinguished learner fails, then it makes the system more reliable and cause extra work to elect a new distinguished learner. We will do a comparison between the first algorithm and the second algorithm in performance, calculating the time needed from the request of client to sending response to the client for each algorithm.

1.2. What Is The Problem

The paxos algorithm is developed to reach consensus among a group of servers so that it can tolerate failures and continue respond to client as long as more than half of the servers are working. This algorithm ensures that the correct value/state is recorded on all the servers in the correct order. Also, all the servers agree on a single value that has been committed. Paxos algorithm has many variants, and most of them are difficult to understand. Though Lamport[1]

has rewritten the basic Paxos in his paper "Paxos Made Simple", it is still considered complicated. Furthermore, it doesn't provide much details in how to implement the Paxos algorithm for general application. This paper will provide a simulation of the basic Paxos and demonstrate how this models works to achieve consensus in a failure-prone network and server environment. Also, this paper will implement another way of learning by choosing a distinguished learner in the learning phase.

1.3. Why This is a Project Related to This Class

Paxos is a family of protocols for reaching consensus in a network of unreliable processors. Paxos is a very important protocol in distributed systems for reaching consensus. Also it is widely used by many companies, such as Google's Megastore[2] and chubby[3], IBM's spinnaker[4], and Apache's Zookeeper. Therefore, Paxos can give us some hands on experience in working with distributed systems.

1.4. Drawbacks of other approaches

There are many variants of Paxos applications, such as IBM's spinnaker data store[4], and google's Megastore[2]. The drawback of those research lies in that those Paxos applications are very specific to a certain datastore application, and they are not easily adapted for more general usage. Moreover, those research don't provide details in how to implement the Paxos algorithm for general usage.

1.5. Advantages of our approaches

This study focuses on simulating the basic Paxos algorithm and implementing three major agents in this algorithm: Proposer, Acceptor and Learner. Second, most of the literature is complex and difficult to understand. This study intends to offer a brief simulation of the basic Paxos algorithm and illustrate the beauty of the simple algorithm in reaching consensus. Finally, as mentioned earlier in the introduction, we will implement an alternative method of the learning phase by appointing a distinguished learner, and all the acceptors when they accept a certain proposal send

response to the distinguished learner instead of all the learners. Then the distinguished learner can notify all the other learners once it learns the accepted proposal and value.

1.6. Statement of the Problem

Our project focuses on simulating the basic Paxos algorithm and provides a demo on how it works in an unreliable network and server environment. Also, this project will compare two different implementations of Paxos by varying the learning phase. Then this study will do a performance comparison of the two implementations of the Paxos algorithm. We hypothesize that a distinguished learner will help to reduce the traffic of communication and reduce the time for response to client.

1.7. Area or Scope of Investigation

This study aims to simulate the basic Paxos algorithm and compare two different ways of implementations of Paxos algorithm. First, we will simulate the basic Paxos algorithm by implementing the roles of proposer, acceptor and learner among a group of servers. The process time for responding to client read and write request will be measured and recorded. Then we will compare two different methods of implementing Paxos algorithm in the learning phase.
Then we will measure the processing time for both client read and write request in the two different implementations. Our hypothesis is that the processing time will be shorter for the implementation with a distinguished learner.

2. Theoretical Bases and Literature Review

2.1. Definition of The Problem

Due to unreliable network and possible server failures, it is important to use a replica of servers to store information in order to give reliable response to client requests. Then the replica of servers will face a problem to reach consensus to make sure all the servers agree on a single value. Paxos is a commonly used algorithm to solve this consensus problem. This study will first do a simulation of the basic Paxos algorithm. Then this study will implement a model with a distinguished learner and test whether this variation will help to reduce response time to client.

2.2. Theoretical Background of the Problem

There are many variants of Paxos algorithm. Lamport described both the basic Paxos and also the fast Paxos[5] which allows 2 message delays and requires the client to send its request to multiple destinations while the basic Paxos has 3 messages delays. Cheap Paxos[6] extends Basic Paxos to be able to tolerate F failures with F+1 main processors and F auxiliary processors by dynamically reconfiguring after each failure. In this project, we will mainly explain the Basic Paxos.

2.3. Basic Paxos

In the basic paxos, there are three roles in the consensus algorithm: proposers, acceptors and learners (Paxos made simple).

2.3.1. Client

The client will issue a request to the system and wait for a response, this request can be either read or write to the files in a distributed file server.

2.3.2. Proposer

A proposer selects a proposal number N which is unique and increases and sends a prepare request with number N to majority of acceptors.

2.3.3. Acceptor

The acceptor assumes the role of fault-tolerant memory of the protocol. Acceptors are usually organized into groups of Quorums, which refers the majority of acceptors in the distributed system. The message must be sent to a Quorum of Acceptors in order to be processed. Any messaged will be ignored unless more than half of the acceptors has agreed on it.

2.3.4. Learner

Learners assume the replication factor for the protocol. Whenever the distributed system has agreed on a client request, the Learner will respond to the client by sending the agreed message to the client. Extra Learners may help to enhance the availability of the processing system.

2.3.5. Two phases in this algorithm:

Phase 1 (prepare):
A proposer selects a proposal number N which is unique and increases and sends a prepare request with number N to majority of acceptors. If an acceptor receives a prepare request with number N greater than any of prepare requests it has received before, it responses YES to that request and promises not to accept any more proposals with a number less than N. It will also send the highest-numbered proposal (if any) that it has accepted and the corresponding V
Promise: (N, N', V').

Phase 2 (accept)
If the proposer has received a response YES for its prepare request from a majority of acceptors, then it will send an accept request to all the acceptors for a proposal numbered N with a value V which is the value of the highest-numbered proposal among the responses.
If an acceptor receives an accept request for a proposal numbered N, it will accept the proposal unless it has already responded to a prepare request which has a number greater than N.

A value is chosen at proposal number N if and only if majority of acceptors accept that value in the phase 2 of the proposal number.

This algorithm ensures safety (only a single value will be sent back to client) and liveness (all the replicas in the system will eventually reach consensus and store the same value). Also, this

algorithm is failure-tolerant as long as F + 1 out of 2F servers are alive, the system can respond to client requests.

2.4. Related Research To Solve The Problem

There are extensive research on the Paxos algorithm, and there are many variant of Paxos algorithm in applications. In IBM's spinnaker[4] datastore application, a leader-based Paxos algorithm is used. The leader takes charge of sending prepare request and accept request to acceptors and also notifying the chosen value to all learners. Google's chubby also uses this leader-based Paxos algorithm in building the datastore.

2.5. Disadvantage of Those Research

These research paper focus on a very specific implementation of Paxos on a certain datastore, thus the application or adaptability is limited. Those research don't provide details in how to implement the the Paxos algorithm for general database usage.

2.6. Proposed Solution

Our project provides a general simulation of the basic Paxos algorithm, therefore, it is a good demo for educational purposes. Furthermore, we will implement two different Paxos algorithms as mentioned in the objectives. We hypothesize with a distinguished learner, the learning phase can be more efficient and make the whole Paxos algorithm more efficient.

2.7. Where Your Solution Different From Others

Our solution is mainly different in implementing another version of learning phase by electing a distinguished learner. We will measure the performance of two implementations of the Paxos algorithm. We will compare the performance of those two implementations in responding to clients' requests. We assume that the the second implementation of appointing a distinguished learner will help to reduce the time delay in sending feedback to client's request.

2.8. Advantages of proposed solution

We implement another version of Paxos algorithm by appointing a distinguished learner, thus reduced the amount of messages to be sent during the learning phase. Therefore, we hypothesize this variation will help to improve the efficiency of the basic Paxos algorithm.

3. Hypothesis

3.1. Multiple Hypothesis

The architecture of our implementation design is shown in Figure 1.
Firstly, our implementation of Paxos is expected to deal with process (or replica, node, server) failure cases by using failure simulators. If one process fails in Paxos progresses, either as proposer or acceptor, our system is still able to send a response to client.

Secondly, Our paxos system should guarantee safety (consistency). This means, when the client sends a request, by using our Paxos system, at most one value would be responded. There will never be the case that two or more values are replied to the client.

Thirdly, by utilization of learner interface, liveness should be addressed. Using Paxos system, the learner will eventually acknowledge some value in final phase, if some value V has been proposed in phase one.

Fourthly, our implementation is expected to show better performance using our second implementation of the Paxos algorithm. In the basic Paxos algorithm, each acceptor sends a message to each learning when it has accepted a proposal. With the adjusted implementation, each acceptor only sends acknowledgement to the distinguished leader. When the distinguished leader has learned the proposed value, it will broadcast to all the other learners. We expect this change would decrease running time. However, there might be negative impact of this implementation, since it increases more complexity when the distinguished learner happens to fail and a new distinguished leader must be elected.

Fifthly, our API allows the rebuild (recovery) of process. If one process fails when Paxos progresses, after certain time, our API would recover it, so that there are always certain number of working processes which would guarantee a response to the client.

3.2. Assumptions

Our simulation is based on the following assumptions:

1) Processes use messengers to deliver messages.
2) Processes and messengers operate at arbitrary speed.
3) Processes and messengers may crash at any time.
4) Processes and messengers never collide or lie.
5) Given N processes in the system. No more than N/2 - 1 processes can crash at the same time.
6) To make it simple, there is only one single value stored in our database.

4. Methodology

4.1. How to generate/collect input data

There will be two kinds of input data. One is the requests from client handled by the API of the distributed system. The client request may contain two types of operations, read and write. To make it simple, we only store one value in each database replica (process). Read operation is to read that value. Write operation is to overwrite that value. The other input is from the failure simulator of network and process. In an asynchronous network, messengers and processes are both unreliable. They may crash at any time. There might be different cases of failures, e.g. leader fail/proposer fail, voter/acceptor fail, distinguished learner fail, random fail of either a proposer or an acceptor. Both input data will be generated by Random class in Java within a proper range.

4.2. How to solve the problem

The original Paxos algorithm proposed by Lamport is difficult to understand and gives limited guidance in how to implement it in different use cases. We want to implement two different versions of Paxos algorithm with variation in the implementation of the learning phase.
Our implementation of Paxos is more practical and easy to implement for distributed datastore. In order to prove the performance improvement and correctness, we will compare the two different implementations of Paxos and test with the same client input data to see whether the second implementation of electing a distinguished learner will help to reduce the response time to client.

4.3. Algorithm design

4.3.1. Roles (as shown in Figure 2):

a. There will be three roles for the processes in the consensus protocol: proposer, acceptor and learner. Each process can be any role at a time.
b. There will be a messenger object used to simulate network, which delivers messages between processes. Each messenger is created dedicated for a single message. The messenger object will be freed after the message is delivered. it also may crash.

4.3.2. Phases (as shown in Figure 3 and Figure 4):

Prepare request:

a. Prepare:

The proposer initiates a proposal identified with a number N. N is bigger than any proposal number the proposer has used before. It sends a prepare request with the proposal to a Quorum of acceptors (prepare: N).

b. Promise:

When acceptor receives a proposal, if the proposal's number N is bigger than any previous proposal number promised or accepted by the acceptor, then the acceptor must return a promise

to proposal saying that it will ignore all future proposals identified with a number less than N. If the acceptor has previously accepted a proposal, it must include the previous proposal number and previous value in its response to the proposer (Promise: (N, N', V')).

Accept Request:

a. Accept Request

When a Proposer receives promises from the majority of acceptors, it needs to set the value of the proposal N as the value associated with the highest proposal number accepted by the acceptors before. If none of the acceptors has accepted a proposal so far (they returned null for number and value), then the proposer may choose any value for its proposal. It then sends an Accept Request to acceptors with the chosen value V (Accept: (N, V)).

b. Accepted

When an acceptor receives an Accept Request message for a proposal N, it must accept it if and only if it has not promised to any prepare requests having an identifier greater than N. In this case, it should store the corresponding value V with proposal number N, and broadcast an Accepted message to the proposer and every Learner (Accepted: (N, V)). Otherwise, it can ignore the Accept Request.

Learning Phase:

1) The first implementation of the learning phase: when an acceptor has accepted a proposal, it will send an acknowledge message to all the other learners. In this implementation, the total message flow in the learning phase is number of acceptors X number of learners.
2) The second implementation of the learning phase: when an acceptor has accepted a proposal, it will send a response to the distinguished learner only. Then the distinguished learner can broadcast the chosen value to all the other learners. In this implementation, the total message flow in the learning phase is the sum of number of acceptors plus number of learners.

In summary, this study focuses on comparing the two variants of Paxos implementation. We intends to test which implementation will be more efficient. We hypothesis that the second implementation with a distinguished learner will be more efficient. However, this implementation may cause more complexity if the distinguished learner dies.

4.4. Programming Languages

The Paxos algorithm will be implemented in Java.

4.5. Tools used

We will mainly use Java multithread to simulate processes in the cluster.

4.6. Output Generation

When reaching consensus by a system of processes, the chosen value will be returned to client by printed to stdout. The runtime of the voting process will also be printed to stdout.

4.7. Methodology to test against hypothesis

We will simulate the real world environment by randomly crash a process in a random interval. There will be a dedicated thread to simulate the process and network failure. When reaching consensus, the chosen value will be returned to client by printing out to stdout. We will also measure the runtime of each implementation by averaging the runtime of 10 independent tests. Then we compare the result and time in the two implementation

5. Implementation

5.1 Code

We have three interface in this implementation, proposer, accepter and learner interfaces. Then we have messenger as a tool to send messages among different nodes in this system. First I will list the interface code and also the messenger code.

```
public interface Proposer {
    public void sendPrepareRequest(String v, long timeStamp);
    public void receivePromise(PromiseMessage m);
}
public interface Acceptor {
    public void receivePrepareRequest(PrepareRequestMessage m);
    public void receiveAcceptRequest(AcceptRequestMessage m);
}
public interface Learner {
    public void receiveAccepted(AcceptedMessage m);
}
```

Messenger code

```
import java.util.Map;
public class Messenger {
    Map<NodeLocationData, Node> nodeLocation;
    public Messenger(Map<NodeLocationData, Node> map){
        nodeLocation = map;
    }
    public void send(Message m){
        new Thread() {
            public void run(){
                NodeLocationData receiver = m.getReceiver();
                nodeLocation.get(receiver).receive(m);
            }
```

```
            }.start();
        }
    }

public class Message {
    private NodeLocationData sender;
    private NodeLocationData receiver;
    public Message(NodeLocationData sender, NodeLocationData receiver){
        this.sender = sender;
        this.receiver = receiver;
    }
    public NodeLocationData getSender() {
        return sender;
    }
    public void setSender(NodeLocationData sender) {
        this.sender = sender;
    }
    public NodeLocationData getReceiver() {
        return receiver;
    }
    public void setReceiver(NodeLocationData receiver) {
        this.receiver = receiver;
    }
}
```

There are five message inherited from message. There are PrepareRequestMessage, PromiseMessage, AcceptedRequestMessage, AcceptedMessage, and DecisionMessage.

```
public class AcceptRequestMessage extends Message {
    private Proposal proposal;
    public AcceptRequestMessage(NodeLocationData sender, NodeLocationData receiver,
Proposal proposal) {
```

```
        super(sender, receiver);
        this.proposal = proposal;
    }
    public Proposal getProposal() {
        return proposal;
    }

public class PrepareRequestMessage extends Message {
    private int sn;
    public PrepareRequestMessage(NodeLocationData sender,
            NodeLocationData receiver, int sn) {
        super(sender, receiver);
        this.sn = sn;
    }
    public int getSn() {
        return sn;
    }
}
public class PromiseMessage extends Message {
    private int currentSn;
    private Proposal acceptedProposal; // proposal accepted before
    public PromiseMessage(NodeLocationData sender, NodeLocationData receiver, int csn,
Proposal accepted) {
        super(sender, receiver);
        this.currentSn = csn;
        this.acceptedProposal = accepted;
    }
    public int getSn() {
        return currentSn;
    }
```

```
    public Proposal getPrevProposal() {
        return acceptedProposal;
    }
}
```

Please refer to our github source files for more code (https://github.com/ShellyLL/OSPaxos)

5.2 Design document and discussion

We applied OOP design principles in design the implementation code. First, we have three interfaces: proposer, accepter and learner interfaces. Then we designed a messenger object to deliver messages among the nodes. Also, we implemented a multiple threads for the messenger to simulate the random time for different nodes to process the request of read or write. We also designed a node object and defined it with different fields so that it can switch between the three different roles of being proposer, acceptor or learner.

6. Data Analysis and Discussion

6.1 Test case

Since we compared two different implementations of Paxos algorithms, we will compare the performance of those two algorithm in read and write request. We will input 5 nodes, 30 nodes and 50 nodes in both of the two Paxos algorithms. The following is our test cases table with the running time measured in milliseconds. We measure the time from the client sending request to the time when the client receiving a feedback in each case.

Table 1 Comparison of Implementation 1 with Implementation 2

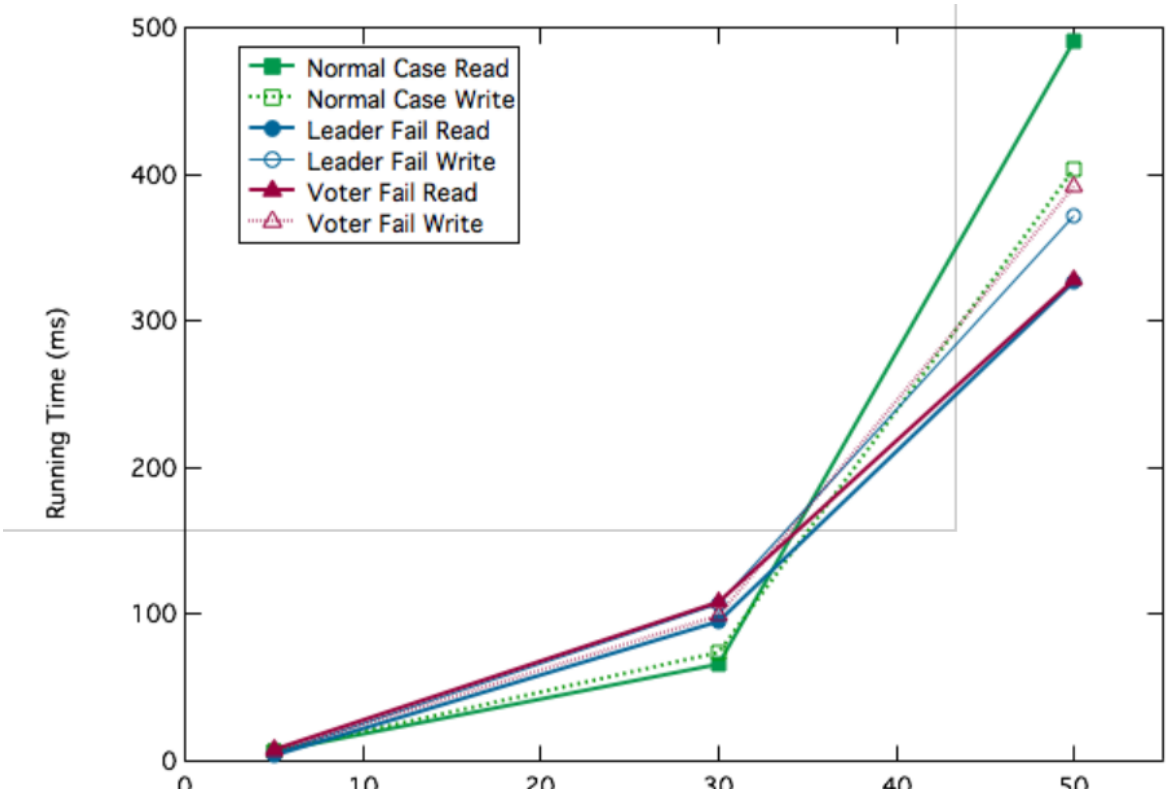
unit: ms

	nodes	5	30	50
implementation 1	read	7	66.6	491
	write	6.3	74.6	403.6
implementation 2	read	4.3	20	22.6
	write	3.6	17	19.6

Table 2 Test Failure Cases in the Implementation 1 of Paxos Algorithm

unit:ms	implementation 1		
nodes	5	30	50
normal read	7	66.6	491
normal write	6.3	74.6	403.6
leader fail read	4	95	327
leader fail write	5	107	372
voter fail read	8	108	328
voter fail write	7	99	392

Graph 1 Comparison of Implementation 1 and Implementation 2



6.2 Output analysis

From the results of Table 1, we can see the implementation 2 (with a distinguished learner) is much efficient than implementation 1 in both read and write operations. Inside each implementation, the read and write time are of the same magnitude. Contrary to our expectation, read takes more time than write operation in both implementation 1 and implementation 2.

From Table 2,we compared the three cases read and write operation in normal case, leader fail case, and voter fail case. We used 5 nodes, 30 nodes and 50 nodes to run in scale to compare different time required. As we can see, with more nodes, it takes more time to respond to read and write operations in all three scenarios (normal, leader fail and voter fail).

6.3 Compare output against hypothesis

For our first hypothesis, our Paxos implementation has fault tolerance. When there is leader fail, voter fail in our system, this system can still provide response to our client. Therefore, it is consistent with our hypothesis.

For our second hypothesis regarding safety of this system, only one value is returned to client in our implementation. Therefore, it is consistent with our hypothesis.

For our third hypothesis regarding liveliness, since our implementations provides feedback to client and reaches consensus within the timeframe. Therefore, it is consistent with our hypothesis.

For our Forth hypothesis regarding the comparison of the two different implementations, we hypothesize that the second algorithm will be more efficient. This hypothesis is also confirmed in our results. With a distinguished learner in the learning phase, the message traffic is reduced from the product of number of acceptors and number of learners to the sum of number of acceptors and number of learners. For our Fifth hypothesis regarding recovery, our implementation include this property and can recover from leaderfail, randomfail, and voterfail. Therefore, all of our hypothesis are confirmed.

6.4 Discussion

Our comparison of two implementations of Paxos algorithm indicates that the second implementation with a distinguished learner is more efficient than the first implementation. This pattern is the same for different test cases, in either normal case, leader fail case and voter fail case. One thing contrary to our expectation is that read takes more time than write in our test. We assume that write would take more time than read. One explanation is that we run read first and write second in our test cases, Java may have optimizations with the sequences operation.

7. Conclusions and Recommendations

7.1 Summary and conclusions

In summary, our two implementations of Paxos algorithms confirms with our hypothesis regarding the liveness, safety, and failure tolerance. Also, with the implementation 2, the time to respond to client read and write is much shorter. This result also confirms with our hypothesis 4. The distinguished learner has to reduced the amount of message traffic during the learning phase. And this result is very significant.

7.2.2 Recommendations for future studies

For recommendations for future study, we would recommend using the distinguished learning in the learning phase to reduced the number of messages needed to be sent in the learning phase. Also, future study may try to implement this Paxos algorithm in another order. When we do read first, then write operations, the read takes more time then write. That's contradictory to our expectations. We infer this abnormal result maybe due to the optimization of Java when it runs operations sequentially.

8. Bibliography

[1] Lamport, Leslie. "Paxos made simple." *ACM Sigact News* 32.4 (2001): 18-25.

[2] Baker, Jason, Chris Bond, James Corbett, J. J. Furman, Andrey Khorlin, James Larson, Jean-Michel Léon, Yawei Li, Alexander Lloyd, and Vadim Yushprakh. "Megastore: Providing Scalable, Highly Available Storage for Interactive Services." In *CIDR*, vol. 11, pp. 223-234. 2011.

[3] Chandra, Tushar D., Robert Griesemer, and Joshua Redstone. "Paxos made live: an engineering perspective." *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*. ACM, 2007.

[4] Rao, Jun, Eugene J. Shekita, and Sandeep Tata. "Using Paxos to build a scalable, consistent, and highly available datastore." *Proceedings of the VLDB Endowment* 4.4 (2011): 243-254.

[5] Lamport, Leslie. "Fast paxos." *Distributed Computing* 19.2 (2006): 79-103.

[6] Lamport, Leslie, and Mike Massa. "Cheap paxos." *Dependable Systems and Networks, 2004 International Conference on*. IEEE, 2004.

9. List of Figures and Graphs

Figure 1. Architecture of Implementation Design

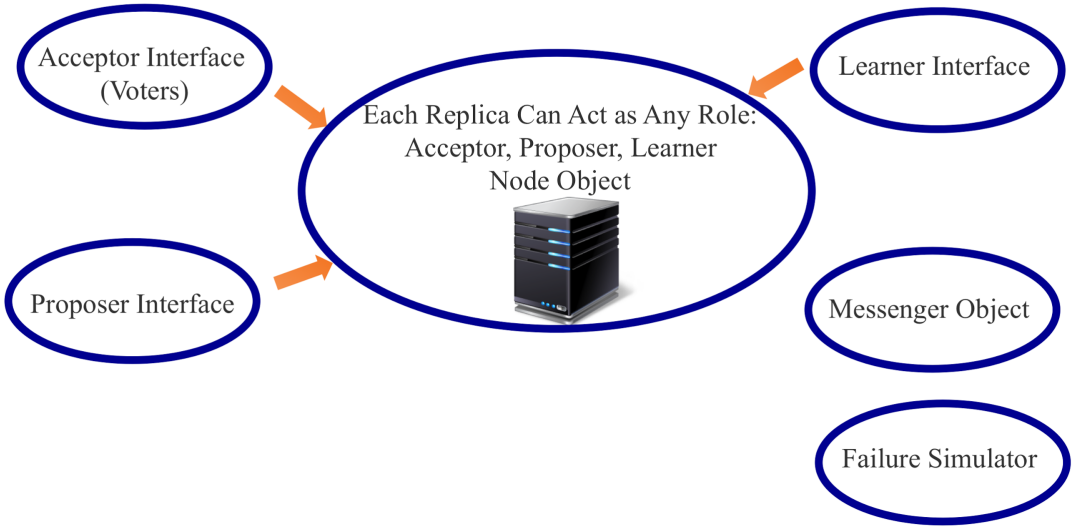


Figure 2. Roles in Paxos Algorithm Implementation

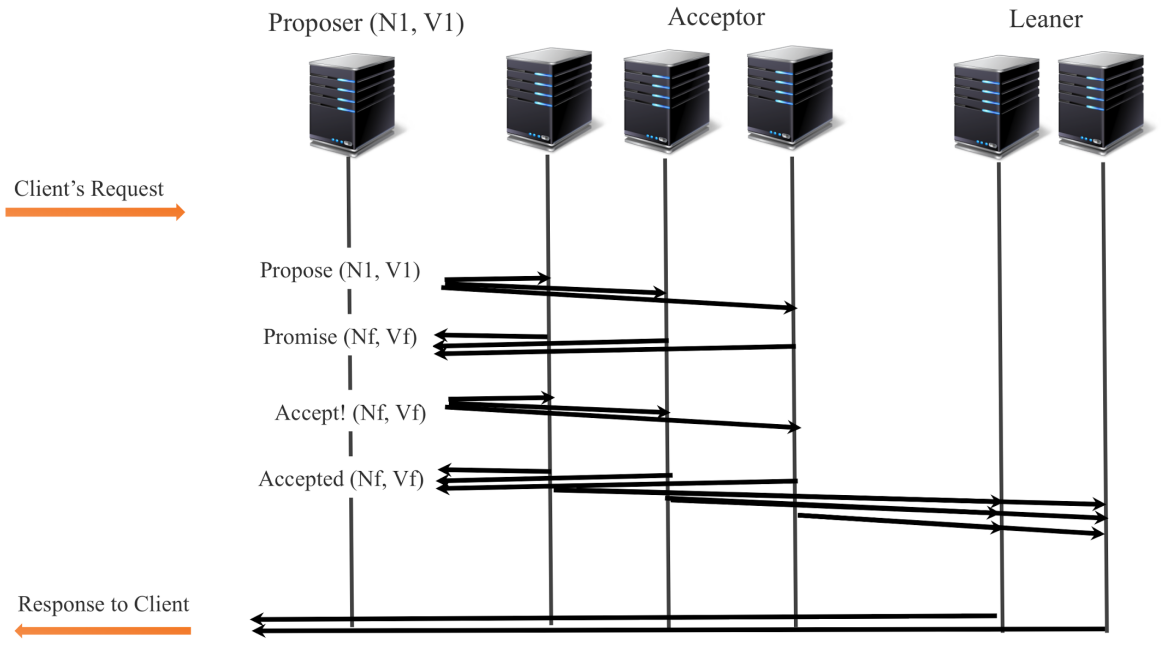


Figure 3. Failure-Free Execution of One Paxos Progress

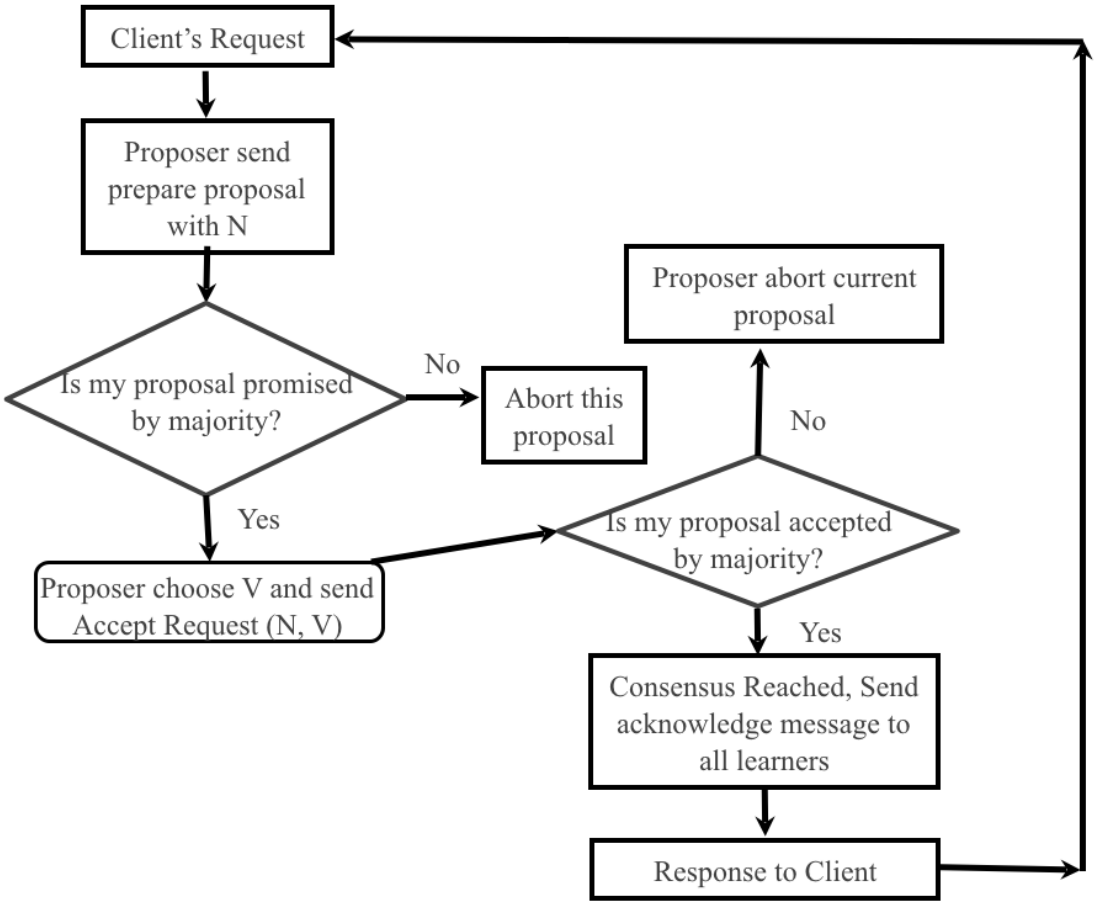


Figure 4. Flow Chart of Paxos Implementation (Algorithm for Proposer)

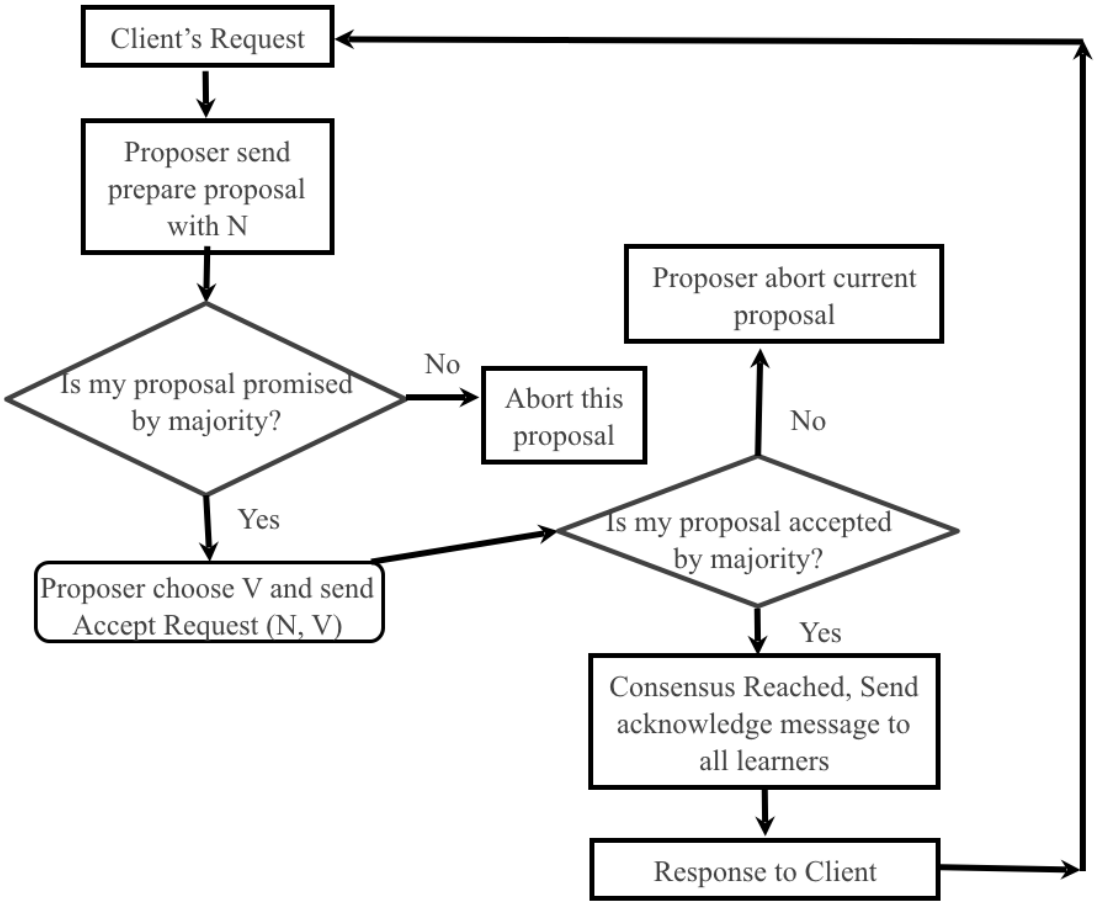


Figure 5. Flow Chart of Paxos Implementation (Optimized Case)

10 Input and Output Listing

10.1 Input Listing

10.1.1 To test Failure type 1 (fault tolerance)

Leader fail: Since it is the proposer who send the prepare request at the first step, when the proposer fails, we elect a new proposer whose ID is one larger than the failed proposer.
Input: init 5, read, write s, leaderFail, read, leaderRecover, read
Voter fail: Any voter can fail. In our simulation, we randomly choose a voter and set it fail. In our implementation, as long as we receive majority votes, we are still able to send valid reply, despite that one or some voters can fail.
Input: init 5, read, write s, voterFail, read, voterRecover, read

10.1.2 To Test Failure type 2 (State fail)

State Fail (During the Paxos process, any state can fail, we use boolean isRunning variables to keep track of the states)
Proposer (Leader) State Fail: it fails after sending the request, so it cannot evaluate voters' promises, not saying to send Accepted! and so no.
Our solution: The other nodes would catch the failure and elect a new proposer
Input: init 5, read, leaderStateFail, read