CS5232 Project Report

Formalisation and Specification Design of Fast Paxos Algorithm using TLA+

By

Lim Ngian Xin Terry (A0218430N)

Gaurav Gandhi (A0254394X)

(Team 1)

Abstract

The problem of Distributed Consensus has always been troubling the computing community, is the building block of all Distributed Systems and is the key problem to be solved for them to function properly. There have been many algorithms like Transaction Commit, Two Phase Commit, Paxos Commit, etc., which attempts to solve the problem of Distributed Consensus. After getting published, Paxos algorithm has been modified many times into its different variants like Multi Paxos,Fast Paxos, Vertical Paxos, etc. This report discusses theory and implementation of Paxos and Fast Paxos algorithms using TLA+. These concepts are explained in easy to understand language and accompanied with a detailed description of TLA+ specifications.

Content

1 Introduction 4

1.1 Fast Paxos 4

2 Methodology 5

2.1 Constraints and Assumptions. 5

3 Implementation 6

3.1 Constants 6

3.2 Variables 6

3.3 Phase 2a (Fast) 7

3.4 Phase 2b (Fast) 7

3.5 Fast Decide (No Collision) 8

3.6 Phase 2a (Classic) 8

3.7 Phase 2b (Classic) 9

3.8 Decide (Collision) 9

3.9 Safety Properties 9

3.10 Liveness Properties 9

4 Future Work 10

5 References 10

6 Appendices 11

6.1 Github Repository Link 11

1 Introduction

The systems distributed across different locations, communicating and interacting amongst themselves through message passing forms the backbone of the modern cloud infrastructure. It is really important for those systems to agree upon a particular value in order to maintain consistency across the systems. In order to achieve this consistency many different consensus algorithms have been proposed in past which handle certain particular scenarios really well but fail to maintain consistency under systems failure. Algorithms like Transaction Commit and Two-Phase Commit are examples of such algorithms. The Two-Phase Commit protocol uses a single coordinator to achieve consensus in a distributed systems environment, which fails to reach a consensus if the coordinator process is crashed or gets disconnected from the rest of the system [1].

This is where Paxos Algorithm and its different variants come into the picture. The original Paxos algorithm, often referred to as Classic Paxos or Vanilla Paxos was proposed by Leslie Lamport in the year 2001 [1] followed by its variant Fast Paxos in the year 2006 [4]. The Classic Paxos algorithm assigns different roles (i.e., Proposer, Acceptor and Learner) to the different systems in the distributed environment, where each role has a specific task to perform. The Classic Paxos algorithm usually has one proposer. There is also a variant of Classic Paxos in which the system can have more than one proposer, which is known as Multi Paxos [2].

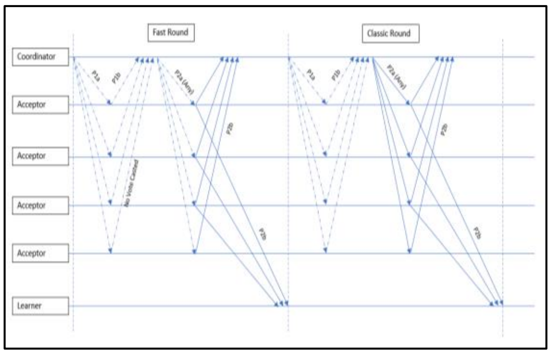
1.1 Fast Paxos

Fast Paxos is an optimization of the original Paxos algorithm (referred to as Classic Paxos), which is further optimized to reduce the latency for arriving at a consensus in a distributed systems environment with the help of larger quorum sizes. A variety of fault-tolerant systems are implemented in real-world scenarios with the help of the Classic Paxos algorithm but a further reduction of latency is implemented in the Fast Paxos. Just like the Classic Paxos, Fast Paxos also operates in rounds and there are two phases in each round [4].

It has 2 types of rounds, fast and classic. Each round consists of multiple phases: preparation, acceptance and learning [5]. When a sole coordinator is present, the preparation phase can be omitted.

Assume that a sole coordinator is present. As can be seen in Figure 1, in the fast round, acceptors are allowed to propose values directly to the coordinator in the acceptance phase. The coordinator will then select a value if it has been proposed by a majority of a quorum of acceptors. This reduces latency compared to Classic Paxos as the coordinator does not need to propose a value.

Figure 1. Each round in Fast Paxos comprises of two phases. There are two varieties of rounds in Fast Paxos: classic and fast. The coordinator selects the value to be voted in classic round whereas in the case of fast round each acceptor can propose its own value. [4]



If the coordinator receives the same value from a quorum of acceptors, a value is selected. If a collision exists and the coordinator is unable to select a value, it will initiate a classic round. Once a value has been selected, it is propagated to all acceptors and learners in the learning phase.

2 Problem Solved

The primary focus of Paxos and its different variants is to establish consensus between all the systems in a Distributed Systems environment under different scenarios so that the system can guarantee consistency. In order to guarantee consistency there is an exchange of messages within the system, Paxos requires 3 message delays. Fast Paxos improves on that by only requiring 2 message delays in cases where there are no collision collisions. If there is a collision it falls back to Classic Paxos and also takes 3 message delays.

3 Methodology

In order to understand Fast Paxos, we read Leslie Lamport's Fast Paxos research paper [2]. However, the implementation presented in the paper was very complex to both understand and implement. In that implementation, Leslie Lamport accounts for situations such as not all agents can communicate with one another, coordinators possibly changing every round, and retransmission of messages.

Zhao Wenbing's Fast Paxos Made Easy: Theory and Implementation [4] explained the protocol in a much simpler fashion, and simplified the rules of collision recovery, as well as showing that we can omit 2 of 4 messages in the system when the system has a unique coordinator.

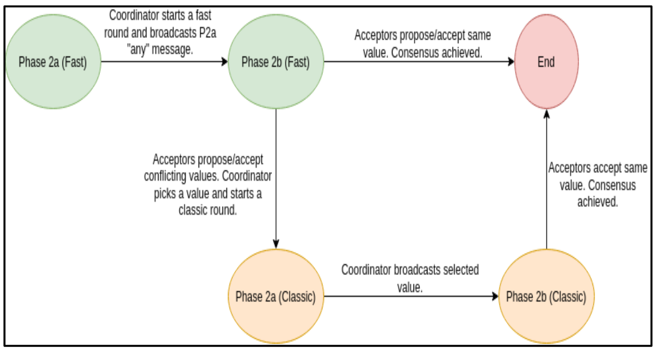
We therefore decided to implement a simplified version of the Fast Paxos specification which can be understood much more easily by adding constraints to the system.

3.1 Constraints and Assumptions.

There is a unique coordinator in the system. Therefore, phase 1a and 1b can be omitted. All agents in the

system can communicate with one another.
Agents must have some stable storage that survives failure and restarts. An agent restores its state from stable storage when it restarts, so the failure of an agent is indistinguishable from its simply pausing. There is thus no need to model failures explicitly.
Only coordinators and acceptors are explicitly modelled. In a fast round, the coordinator only has a single role, while the acceptors are also proposers and learners. In a classic round, the coordinator is also a proposer, and the acceptors are also learners.
Since Fast Paxos is an asynchronous system, our model must also reflect the fact that messages may be delayed, received out-of-order, or lost, and be able to handle such cases.

Figure 2. Graphical Representation for different phases of Fast Paxos implemention


Phase 2a (Fast)
Coordinator starts a fast round and broadcasts P2a "any" message.
Phase 2b (Fast)
Acceptors propose/accept same value. Consensus achieved.
End
Acceptors propose/accept conflicting values. Coordinator picks a value and starts a classic round.
Acceptors accept same value. Consensus achieved.
Phase 2a (Classic)
Coordinator broadcasts selected value.
Phase 2b (Classic)

4 Implementation

Our Fast Paxos specification can be simplified into 4 phases i.e., Phase 2a (Fast), Phase 2b (Fast), Phase 2a (Classic) and Phase 2b (Classic) as shown in the figure 2.

4.1 Constants

Table 1. Constants and their of description.

Constant	**Description**
any	A special value sent by the coordinator in a P2a message to ask the acceptors to propose a value.
none	A special value indicating none.
Replicas	The acceptors/learners in the system.
Values	Set of values that may be proposed.
Ballots	Set of integers indicating the round number.
Quorums	A simple majority of acceptors required for Classic Paxos protocol, defined to be $n/2 + 1$.
FastQuorums	A majority of acceptors required for Fast Paxos protocol, defined to be $3n/4$.
FastBallots	Set of integers representing fast rounds.
ClassicBallots	Set of integers representing classic rounds.

4.2 Variables

Table 2. Variables and their of description.

Variable	**Initial Value**	**Description**
messages	∅	Set of all messages sent.

decision	none	Decision of any acceptor.
maxBallot	∀ r ∈ Replicas, maxBallot[r] = 0	Highest ballot seen by an acceptor.
maxVBallot	∀ r ∈ Replicas, maxVBallot[r] = 0	Highest ballot accepted by an acceptor.
maxValue	∀ r ∈ Replicas, maxVBallot[r] = none	Value of the highest ballot accepted by an acceptor.
cValue	none	Chosen value of coordinator.

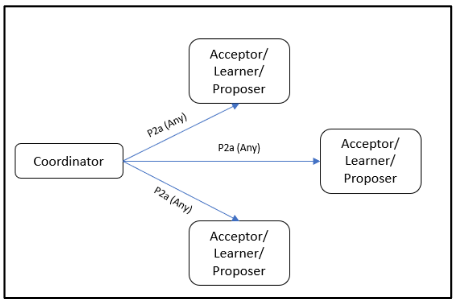
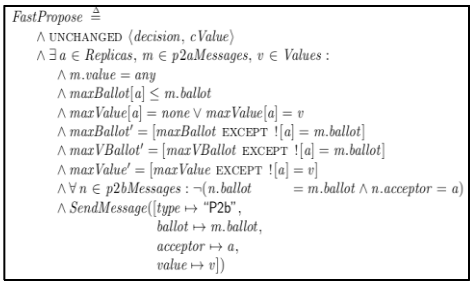

Coordinator
P2a (Any)
P2a (Any)
P2a (Any)
Acceptor/ Learner/ Proposer
Acceptor/ Learner/ Proposer
Acceptor/ Learner/ Proposer

Figure 3. Phase 2a of Fast Paxos

4.3 Phase 2a (Fast)

The coordinator can start a fast round by broadcasting a P2a "any" message to the acceptors, if no value has been proposed before. There is no need to explicitly model "if no value has been proposed before", as it is possible that in an asynchronous system, the acceptors may have already proposed a value, but the messages were lost. Alternatively, an acceptor might have already proposed a value previously, but received this P2a "any" message much later as it was delayed.



```
FastPropose ≜
    ∧ UNCHANGED ⟨decision, cValue⟩
    ∧ ∃ a ∈ Replicas, m ∈ p2aMessages, v ∈ Values :
        ∧ m.value = any
        ∧ maxBallot[a] ≤ m.ballot
        ∧ maxValue[a] = none ∨ maxValue[a] = v
        ∧ maxBallot' = [maxBallot EXCEPT ![a] = m.ballot]
        ∧ maxVBallot' = [maxVBallot EXCEPT ![a] = m.ballot]
        ∧ maxValue' = [maxValue EXCEPT ![a] = v]
        ∧ ∀ n ∈ p2bMessages : ¬(n.ballot        = m.ballot ∧ n.acceptor = a)
        ∧ SendMessage([type ↦ "P2b",
                       ballot ↦ m.ballot,
                       acceptor ↦ a,
                       value ↦ v])
```

Figure 4. TLA+ specification of Phase 2a of Fast Round.

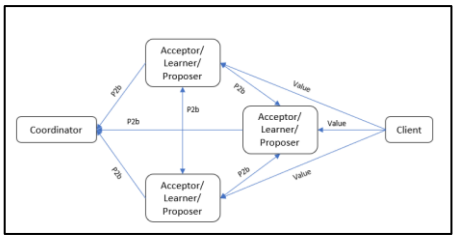
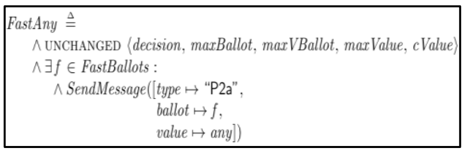

Coordinator
Acceptor/ Learner/ Proposer
Acceptor/ Learner/ Proposer
Acceptor/ Learner/ Proposer
Client
P2b
P2b
P2b
P2b
P2b
P2b
Value
Value
Value

Figure 5. Phase 2b of Fast Paxos

4.4 Phase 2b (Fast)

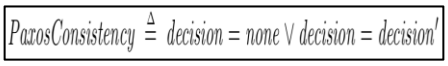


```
FastAny ≜
    ∧ UNCHANGED ⟨decision, maxBallot, maxVBallot, maxValue, cValue⟩
    ∧ ∃ f ∈ FastBallots :
        ∧ SendMessage([type ↦ "P2a",
                       ballot ↦ f,
                       value ↦ any])
```

Figure 6. TLA+ specification of Phase 2b of Fast Round.

Upon receiving a P2a "any" message with a ballot higher or equal to any it has seen before, an acceptor is allowed to propose a value. If it has already accepted a value, it will propose that. Otherwise, it will propose a new value and accept that value. Then, it will broadcast the

value to the coordinator and acceptors in a P2b message.
There will be 2 possible cases from here:
Every acceptor proposes the same value. In this case, there is no collision, consensus is reached and no further action needs to be taken.
Multiple different values were proposed by the acceptors. In this case, there is a collision and the coordinator needs to start a classic round to resolve this collision.

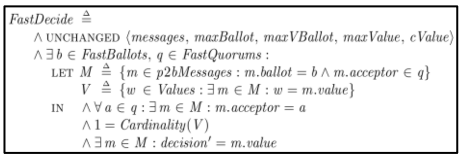


```
PaxosConsistency ≜ decision = none ∨ decision = decision'
```

In a real world scenario, the new value proposed by the acceptor will likely come from a client. For example, a user initiates a bank transaction, and the value of their new account balance is sent to multiple acceptors in the system.

4.5 Fast Decide (No Collision)

Figure 7. TLA+ specification of Decide phase of Fast Round.



```
FastDecide ≜
    ∧ UNCHANGED ⟨messages, maxBallot, maxVBallot, maxValue, cValue⟩
    ∧ ∃ b ∈ FastBallots, q ∈ FastQuorums :
        LET M ≜ {m ∈ p2bMessages : m.ballot = b ∧ m.acceptor ∈ q}
            V ≜ {w ∈ Values : ∃ m ∈ M : w = m.value}
        IN  ∧ ∀ a ∈ q : ∃ m ∈ M : m.acceptor = a
            ∧ 1 = Cardinality(V)
            ∧ ∃ m ∈ M : decision' = m.value
```

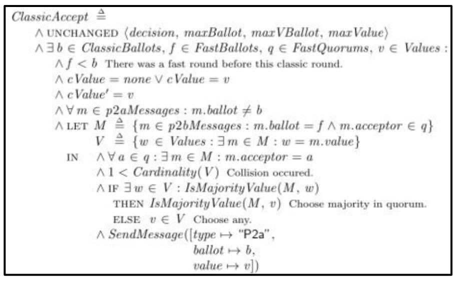
In the first case, an acceptor receives a fast quorum of P2b messages for this fast round with the same value in every message. It decides on that value, and the value is learnt.

Because the quorum size of a fast round and classic round is different, we assume that the acceptor distinguishes a fast round and classic round based on the P2a message it receives. If the P2a message contains the special value “any”, it is a fast round. Else it is a classic round.

Rather than have the variable decision be a function which maps every acceptor to its decision, it represents the decision of any acceptor. FastDecide can run multiple times, and decision can be set multiple times. We then use the PaxosConsistency safety property to check if decision is ever set to a different value. By doing this, we reduce the number of distinct states of the model, which decreases the verification time taken while maintaining correctness.

4.6 Phase 2a (Classic)

Figure 8. TLA+ specification of Phase 2a of Calssic Round.

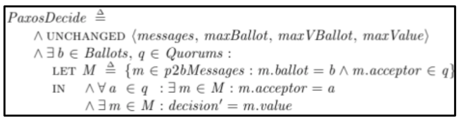


```
ClassicAccept ≜
    ∧ UNCHANGED ⟨decision, maxBallot, maxVBallot, maxValue⟩
    ∧ ∃ b ∈ ClassicBallots, f ∈ FastBallots, q ∈ FastQuorums, v ∈ Values :
        ∧ f < b  There was a fast round before this classic round.
        ∧ cValue = none ∨ cValue = v
        ∧ cValue' = v
        ∧ ∀ m ∈ p2aMessages : m.ballot ≠ b
        ∧ LET M ≜ {m ∈ p2bMessages : m.ballot = f ∧ m.acceptor ∈ q}
              V ≜ {w ∈ Values : ∃ m ∈ M : w = m.value}
          IN  ∧ ∀ a ∈ q : ∃ m ∈ M : m.acceptor = a
              ∧ 1 < Cardinality(V)  Collision occured.
              ∧ IF ∃ w ∈ V : IsMajorityValue(M, w)
                  THEN IsMajorityValue(M, v)  Choose majority in quorum.
                  ELSE v ∈ V  Choose any.
              ∧ SendMessage([type ↦ "P2a",
                             ballot ↦ b,
                             value ↦ v])
```

Upon detecting a collision in a fast round, the coordinator will resolve the collision by falling back to Classic Paxos. If the coordinator has already picked a value in a previous fast round, it will pick that again.

Otherwise it will pick a value from the P2b messages using the following rules:

Figure 9. TLA+ specification of Decide Phase of Classic Round.



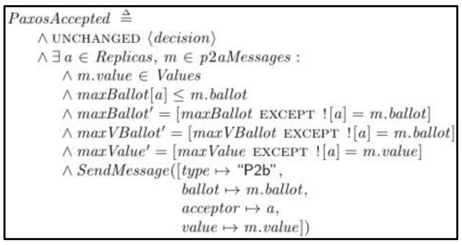
```
PaxosDecide ≜
    ∧ UNCHANGED ⟨messages, maxBallot, maxVBallot, maxValue⟩
    ∧ ∃ b ∈ Ballots, q ∈ Quorums :
        LET M ≜ {m ∈ p2bMessages : m.ballot = b ∧ m.acceptor ∈ q}
        IN   ∧ ∀ a ∈ q  : ∃ m ∈ M : m.acceptor = a
             ∧ ∃ m ∈ M : decision′ = m.value
```

"A value is chosen if the majority of acceptors in a fast quorum proposed that value.
Else, it can pick any proposed value from the P2b messages."

The coordinator will then start a classic round by sending a P2a message with a higher ballot number, and the picked value. Since a classic round only starts when a collision occurs in a fast round, there is guaranteed to be a fast round that happened before a classic round, but because this is an asynchronous system, there is no guarantee that all the acceptors received a P2a "any" message.

4.7 Phase 2b (Classic)

Figure 10. TLA+ specification of Phase 2b of Classic Round.



```
PaxosAccepted ≜
    ∧ UNCHANGED ⟨decision⟩
    ∧ ∃ a ∈ Replicas, m ∈ p2aMessages :
        ∧ m.value ∈ Values
        ∧ maxBallot[a] ≤ m.ballot
        ∧ maxBallot′ = [maxBallot EXCEPT ![a] = m.ballot]
        ∧ maxVBallot′ = [maxVBallot EXCEPT ![a] = m.ballot]
        ∧ maxValue′ = [maxValue EXCEPT ![a] = m.value]
        ∧ SendMessage([type ↦ "P2b",
                       ballot ↦ m.ballot,
                       acceptor ↦ a,
                       value ↦ m.value])
```

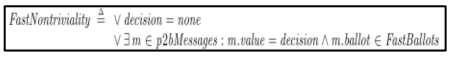
Upon receiving a P2a message with a ballot higher or equal to any it has seen before, an acceptor will accept the value in the P2a message, and it will then broadcast a P2b message with the value to the coordinator and other acceptors. Since there is only 1 coordinator, all the acceptors are guaranteed to accept the same value.

4.8 Decide (Collision)

In a classic round, if an acceptor receives a classic quorum of P2b messages for that round, it choses a value from one of the P2b messages. Since all the acceptors are broadcasting the same value, all of the P2b messages will contain the same value and consensus will be reached, thus this value is learnt.

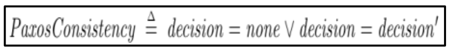
4.9 Safety Properties

We have to satisfy 2 safety properties, non-triviality and consistency.
Non-triviality: Only proposed values can be learnt.



```
FastNontriviality ≜ ∨ decision = none
                    ∨ ∃ m ∈ p2bMessages : m.value = decision ∧ m.ballot ∈ FastBallots
```

Consistency: All acceptors must learn the same value.



```
PaxosConsistency ≜ decision = none ∨ decision = decision′
```

4.10 Liveness Properties

We did not include any liveness properties as Fast Paxos is not guaranteed to terminate.

5 Usefulness

Fast Paxos is useful in systems which have a low rate of collision, where the all proposers will propose the same value most of the time. In systems with a high rate of collision, where proposers tend to propose different values, Fast Paxos is slower than Classic Paxos, as Fast Paxos falls back onto Classic Paxos to resolve collisions.

An example of a system with a low rate collision may be a player's inventory in an online game. Since players are usually only allowed to log in onto one device at a time, there is only one client proposing changes to the player's inventory. As such, all the proposers will receive the same value from the client to propose.

6 Future Work

- In our current implementation, we assume that all agents can communicate with one another. Perhaps we can investigate cases where certain agents can only communicate one way, or not all agents can communicate with one another.
- Our current implementation also works because we assume that faults are benign, and that the agents have stable storage which allows them to recover into their state before failure. Maybe we can investigate how the system can possibly handle byzantine faults, and what is the fault tolerance the system can handle.
- With only 1 unique coordinator in the system, Zhao Wenbing's Fast Paxos Made Easy: Theory and Implementation [4] suggests that we can also omit phase P2a in a fast round. Perhaps we can model that.

7 Conclusion

In conclusion, this project not only helped us understand how Fast Paxos worked, but also how to take a complex system and break it down into simpler terms, and decide on what reasonable constraints we can add while preserving correctness.

We hope that our specification of Fast Paxos can help others understand how to implement and verify Fast Paxos in the simplest terms, and from there build upon it for more complex scenarios.

8 References

[1]. Gray, J., & Lamport, L. (2006). Consensus on transaction commit. ACM Transactions on Database Systems, 31(1), 133-160. https://doi.org/10.1145/1132863.1132867

[2]. Lamport, L. (2006). Fast Paxos. Distributed Computing, 19(2), 79-103. https://doi.org/10.1007/s00446-006-0005-x

[3]. Luis Quesada Torres. (2018, March 1). The Paxos Algorithm [Video]. YouTube. https://www.youtube.com/watch?v=d7nAGI_NZPk

[4]. Zhao, W. (2015). Fast Paxos made easy. International Journal of Distributed Systems and Technologies, 6(1), 15-33. https://doi.org/10.4018/ijdst.2015010102

[5]. Lamport, L. (2001). Paxos Made Simple. Distributed Computing

9 Appendices

9.1 Github Repository Link

https://github.com/sudogandhi/CS5232-Fast-Paxos-Specification