A Simpler Proof for Paxos and Fast Paxos

Keith Marzullo
University of California
San Diego, USA

Alessandro Mei*
Sapienza University
Rome, Italy

Hein Meling
University of Stavanger
Norway

ABSTRACT

The Fast Paxos algorithm, when presented in plain English, is still quite hard to understand for those like us—people who don't have the brain of Leslie Lamport. Here we propose a simpler proof that can help understand it.

1. INTRODUCTION

Reaching agreement, or consensus, among a set of processes in a distributed system is a fundamental problem that has been studied extensively in the research literature []. A distributed algorithm is said to solve consensus if a set of processes, each starting with some initial value, can reach agreement on a common value chosen among the initial values.

The Paxos [3, 4] and Fast Paxos [5] protocols for solving distributed consensus are acclaimed for be both simple and efficient, yet truly understanding these protocols and why they work still demands a significant effort on those interested in learning, and perhaps implementing them. This is still holds true after all these years, despite many efforts to explain the protocol in simple terms, e.g. [4].

Very few textbooks include a description of the protocols, despite their importance for implementing fault tolerant services.

In this paper, we aim to describe the Paxos protocols in an easy to understand manner, using simple realistic examples rather than a general description. We also provide a new and simpler proof of the protocols.

2. PAXOS

In its original incarnation [3], Paxos was described in terms of a parliament in which part-time legislators needed to keep consistent records of their passing of laws. This description was hard to understand and related to practical computer science, and Lamport later provided a simpler description of the protocol in [4]. In this description, the Paxos [3, 4] was described in terms of three separate agent roles: *proposers* that can propose values for consensus, *acceptors* that accept a value among those proposed, and *learners* that learn the chosen value. A process may take on multiple roles: in a typical configuration, all processes play all roles. Paxos is safe for any number of crash failures, and can make progress with up to t_a crash failures, given $2t_a + 1$ acceptors. The number of learners and proposers is unrestricted.

1. *Prepare:* A proposer chooses one of the rounds associated to itself, say round i, and starts off the round by sending a ⟨PREPARE⟩ message to all the acceptors.

2. *Promise:* When an acceptor receives the ⟨PREPARE⟩ message for round i, it sends a ⟨PROMISE⟩ message back to the proposer (unless it has promised not to!). In this way, the acceptor promises that it will not participate in any round smaller than i and it will stick to this promise. Along with the promise, the acceptor sends the last value it has voted and the associated round.

3. *Accept:* After collecting a quorum of $n - f$ promises for round i from the acceptors, the proposer sends an ⟨ACCEPT⟩ message to all the acceptors asking to vote for a value selected as follows:

   - A value v proposed by the proposer, if no acceptor in the quorum has ever voted;
   - The value *vval* in the promises that is associated with the highest round, otherwise (note that there can exists at most one such value, since the only value that can be voted in a round is the one proposed by the proposer that is responsible of that round).

4. *Learn:* If an acceptor receives an ⟨ACCEPT⟩ message, and if it has not promised otherwise, it *votes* for the value in the message and sends a ⟨LEARN⟩ message to all the learners to let them know about the vote. Acceptors votes only once in each round.

*This work was performed while Alessandro Mei was a Marie Curie Fellow at the Computer Science and Engineering Department, University of California San Diego, USA. The fellowship is funded by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n. 253461.

5. *The value is chosen*: If a learner receives $n-f$ ⟨Learn⟩ messages for the same round and the same value from a quorum of $n-f$ acceptors, then the value is *chosen*.

We will see later how, and under what hypothesis, we can guarantee progress. Now, we focus on safety.

Algorithm 1 Paxos — Proposer p

1: **Constants:**
2: A, n, and f. {A is the set of acceptors. $n = |A|$ and $f = \lfloor (n-1)/2 \rfloor$.}
3: **Init:**
4: $crnd \leftarrow -1$ {Current round number}
5: **on** ⟨Propose, val⟩
6: $crnd \leftarrow$ pickNextRound($crnd$)
7: $cval \leftarrow val$
8: $P \leftarrow \emptyset$
9: **send** ⟨Prepare, $crnd$⟩ **to** A
10: **on** ⟨Promise, $rnd, vrnd, vval$⟩ with $rnd = crnd$ **from** acceptor a
11: $P \leftarrow P \cup (vrnd, vval)$
12: **on event** $|P| \geq n - f$
13: $j = \max\{vrnd : (vrnd, vval) \in P\}$
14: **if** $j \geq 0$ **then**
15: $V = \{vval : (j, vval) \in P\}$
16: $cval \leftarrow$ pick(V) {Pick proposed value $vval$ with largest $vrnd$}
17: **send** ⟨Accept, $crnd, cval$⟩ **to** A

Algorithm 2 Paxos — Acceptor a

1: **Constants:**
2: L {Set of learners}
3: **Init:**
4: $rnd \leftarrow -1$
5: $vrnd \leftarrow -1$
6: $vval \leftarrow -1$
7: **on** ⟨Prepare, $prnd$⟩ with $prnd > rnd$ **from** proposer p
8: $rnd \leftarrow prnd$
9: **send** ⟨Promise, $rnd, vrnd, vval$⟩ **to** proposer p
10: **on** ⟨Accept, i, v⟩ with $i \geq rnd$ **from** proposer p
11: $rnd \leftarrow i$
12: $vrnd \leftarrow i$
13: $vval \leftarrow v$
14: **send** ⟨Learn, i, v⟩ **to** L

2.1 A simple proof of safety for Paxos

To prove safety, we need to prove the following three properties:

CS1 Only a proposed value may be chosen.

CS2 Only a single value is chosen.

CS3 Only a chosen value may be learned by a correct learner.

Property CS1 is very easy to check, acceptors only vote for values that have been proposed by the proposers. Property CS3 is very easy to check as well, learners learn a value only if it has been voted by a quorum of acceptors, the same quorum needed to choose the value. Therefore, we can concentrate on Property CS2. In order to do that, it is more convenient to prove the following safety property:

Algorithm 3 Paxos — Learner l

1: **Init:**
2: $V \leftarrow \emptyset$
3: **on** ⟨Learn, (i, v)⟩ **from** acceptor a
4: $V \leftarrow V \uplus (i, v)$
5: **on event** $\exists i, v : |\{(i, v) : (i, v) \in V\}| \geq n - f$
6: v **is chosen**

CS If acceptor a has *voted* for value v at round i, then no value $v' \neq v$ can be *chosen* in any previous round.

Property CS is easier to handle than Property CS2 and it implies it in a straightforward way (note that it is impossible that two different values are chosen in the same round since acceptors vote at most once in each round, therefore there cannot be two different quorums of ⟨Learn⟩ messages).

Let's assume that acceptor a has *voted* for value v at round i. Acceptor a must have received an ⟨Accept⟩ message from a proposer. In turn, the proposer must have collected a quorum of $n-f$ ⟨Promise⟩ messages for round i from the acceptors. Let $Q \subseteq A$ be the set of acceptors who sent the ⟨Promise⟩ message for round i. For each acceptor $a \in Q$, the promise has the form ⟨promise, $i, vrnd_a, vval_a$⟩, and the meaning is that acceptor a has last voted in round $vrnd_a$, its last vote was for value $vval_a$, and it has promised not to vote in rounds $vrnd+1, \ldots, i-1$. Let j be the largest $vrnd$ in the promises collected from the acceptors in Q. Now we have all the preliminaries to understand the proof. Before coming to the theorem, however, it is useful to look at Figure 1, where we show a graphical representation of the promises collected by the proposer executing round 9 in a possible execution of Paxos with 7 acceptors.

Theorem 1. *In Paxos, if acceptor a has voted for value v at round i, then no value $v' \neq v$ can be chosen in any previous round.*

Proof. We prove Property CS by induction on round i. The base case, when $i = 0$, is trivial. Let's move to the inductive step: We assume that the property is true for rounds $0, \ldots, i-1$ (the *inductive hypothesis*) and we prove that the property is true for round i. Recall that $Q \subseteq A$, $|Q| \geq n - f$, is the set of acceptors who sent the ⟨Promise⟩ for round i and that $j < i$ is the largest $vrnd$ in the promises.

First, no value can be chosen in rounds $j+1, \ldots, i-1$. Indeed, the acceptors in Q have promised not to vote in these rounds and the remaining $|A \setminus Q| \leq n - (n-f) = f < n - f$ acceptors are not enough to form a quorum. If $j = -1$, then we are done with the inductive step and with the proof. So, let's assume that $j \geq 0$ and proceed.

Assume acceptor a has voted for value v at round i. This is possible only if some acceptor in Q has voted value v

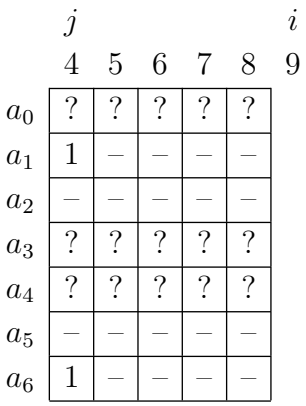


Figure 1: On the rows we have the acceptors; on the columns the rounds. The quorum Q, of $f = 4$ acceptors, consists of $\{a_1, a_2, a_5, a_6\}$. Acceptors a_1 and $a6$ have sent $\langle \text{PROMISE}, 9, 4, 1\rangle$ (that means they last voted 1 in round 4 and they promised not to vote in rounds $5, \ldots, 8$—hence the dashes). Acceptors a_2 and $a5$ have sent a promise message with some *vrnd* strictly smaller than 4 (they have last voted some round before and promised not to vote before round 9). The largest round j in which an acceptor in Q has voted is round 4. We don't know the votes of the f acceptors $\{a_0, a_3, a_4\}$ that are not in the quorum—hence the question marks.

in round j (recall that $j \geq 0$ and see Rows 15–16 in Algorithm 1). We can deduce two consequences: No value $v' \neq v$ can be chosen in round j (round j is associated to a single proposer and thus only a single value is proposed); and, no value $v' \neq v$ can be chosen in rounds $0, \ldots, j-1$ (by using the inductive hypothesis on round j). This concludes the inductive step and the proof. □

Therefore, we know that Paxos is safe. Note that safety does not depend on the number of failures—no assumption on the number of failures is used in the proof of the safety theorem. If more than f acceptors fail, then Paxos does not choose any value (no quorum can be formed) but it is still safe. To get liveness, the number of failures has to be at most f and also do we need other assumptions.

2.2 Liveness of Paxos

In Paxos, progress is not guaranteed even if the number of failures is at most $f = \lfloor (n-1)/2 \rfloor$. Indeed, if more than one proposer starts off new rounds concurrently, then there is no guarantee that any round completes and a value is chosen. As an example, rounds may interweave in the following way: Proposer p sends $\langle \text{PREPARE}\rangle$ message for round i, acceptors send the corresponding $\langle \text{PROMISE}\rangle$ message, then proposer p sends the accept message; unfortunately, right before receiving the $\langle \text{ACCEPT}\rangle$ message the acceptors receive a $\langle \text{PREPARE}\rangle$ message for round $i' > i$ from another proposer; the acceptors have to promise not to vote for any round before i', so preventing round i to complete successfully. In turn, round i' can then be frustrated by a further round and this can go on indefinitely. It is important to realize that we cannot do much about it—we cannot get both safety and liveness—since it is impossible to solve consensus in the presence of faults (see the well-known FLP result [2]).

To get progress we can however use a leader election protocol. One of the proposer is elected as the "distinguished" proposer—the only one allowed to start off new rounds. In this way, no conflict occurs and, if the number of failures is at most f, the round completes nicely with a chosen value. In the following, we will refer to the distinguish proposer as the "coordinator". Of course, we are not circumventing the FLP result. In the presence of faults leader election is impossible as well. However, with Paxos we can accept that the election fails and that two or more leaders are chosen. In that case we cannot guarantee liveness but Paxos is there to guarantee safety whatsoever.

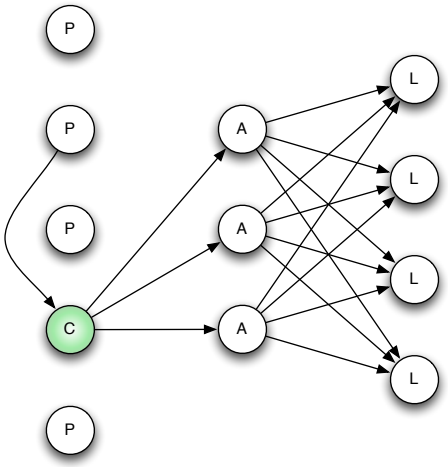
More formally, we can assume that the processes have access to failure detector Ω that indicates who is the leader. At each process p, the Ω module outputs a single process that is currently considered the leader by process p. The Ω abstraction has the following property: There is a time after which all the correct processes always trust the same correct process. For a more formal definition, see [1]. In Paxos, Ω is used by the proposers to agree on who is the coordinator. Proposer p considers itself the coordinator if its Ω module outputs p. Eventually, all the proposers will agree and therefore the system is guaranteed to progress.

3. FAST PAXOS

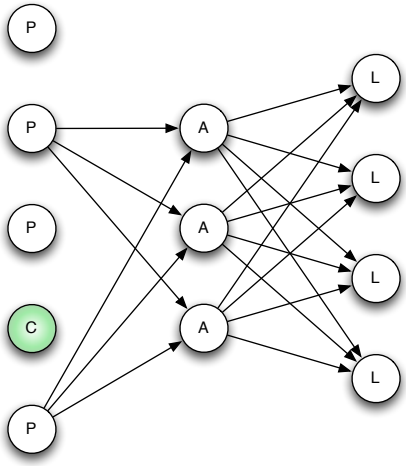
Consensus protocols like Paxos are often used to get consensus on a *sequence* of values. For example, Paxos can be used to implement a replicated state machine [], where a large number of consistently ordered commands should be agreed upon by the replicated servers. A sequence of Paxos *instances* executes and instance s is used to agree on the s-th command to the replicated state machine.

A key observation in such a system is that a single $\langle \text{PREPARE}\rangle$ message can be sent to initiate a sequence of Paxos instances. Similarly, a single $\langle \text{PROMISE}\rangle$ message can be sent to respond to the aggregate $\langle \text{PREPARE}\rangle$ message. In this way, the overhead of these two messages of Paxos is amortized over a large number of instances. To complete each instance, a proposer p that has a value to propose sends the value to the coordinator c. The coordinator then completes the instance by sending the proper $\langle \text{ACCEPT}\rangle$ message to the acceptors that will be followed by the $\langle \text{LEARN}\rangle$ message to the learners. The pattern is shown in Figure 2(a). You can clearly see that the delay between proposing and learning in each instance consists of 3 messages.

Fast Paxos [5] is based on the following idea: We can save one message and reduce the delay between proposing and learning (and thus the delay of executing a command by a service based on the replicated state machine approach) by allowing the proposer to send its value directly to the acceptors. To achieve this result, the coordinator can start off a so called *fast round* by sending an $\langle \text{ACCEPT}, crnd, \perp\rangle$ message (a so called *accept any* message) to the acceptors as a response to the $\langle \text{PROMISE}\rangle$ messages. The meaning of the $\langle \text{ACCEPT}, crnd, \perp\rangle$ message is that, in the same round, the acceptors can accept any value that they receive from any of the proposers. Since the accept any message does



(a) Proposer → coordinator → acceptor → learner.



(b) Proposer → acceptor → learner.

Figure 2: Delay in Paxos (a) and in Fast Paxos (b).

not carry any value, the coordinator can send an aggregate $\langle \text{ACCEPT}, crnd, \perp \rangle$ for the whole sequence of instances just like the $\langle$PREPARE$\rangle$ and $\langle$PROMISE$\rangle$ messages, and so the overhead of this message does not thus influence significantly the delay of a single instance. As a result, the pattern of each instance consists of two messages, a message with the proposed value from any of the proposer to the acceptors, and a $\langle$LEARN$\rangle$ message from the acceptors to the learners, as shown in Figure 2(b).

The improvement of Fast Paxos in delay comes at a price. In the same fast round multiple proposers can send a value to the acceptors. Therefore, in the same round many different values can be voted by the acceptors. Note that this is different from Paxos, where a key ingredient in the proof of the safety theorem is exactly that in every round only one value, the one proposed by the proposer responsible for that round, can be voted. To solve the problem, in Fast Paxos we require a larger quorum of $n - f'$ acceptors, where $f' = \lfloor (n-1)/3 \rfloor$. Of course, the downside is that the number of failures we can tolerate is now only f', which is smaller, or, in other words, the replication required to tolerate failures is larger. Why we need a larger quorum and why exactly that quorum is enough to guarantee safety will be clearer in the next sections.

3.1 The Fast Paxos Protocol

First, it is useful to understand why the quorum requirement of $n - f$, where $f = \lfloor (n-1)/2 \rfloor$, does not work any more. To make a practical example, assume that $n = 7$ and $f = 3$. During round i, the proposer responsible for round i collects a quorum of exactly $n - f = 4$ promises from the acceptors. Some of the promises have the form $\langle \text{PROMISE}, i, j, 1 \rangle$ (the last vote was cast in round j and the value was 1) and some have the form $\langle \text{PROMISE}, i, j, 2 \rangle$ (the last vote was cast in round j and the value was 2). This is possible in Fast Paxos since multiple values can be proposed and voted in a fast round. The problem is that the coordinator still does not know the last vote of $|A \backslash Q| = n-(n-f) = f = n-f-1 = 3$ acceptors and, unfortunately, these may be enough to form a quorum of $n - f = 4$ votes in round j either on value 1 or on value 2. The proposer just does not know and therefore it cannot make any safe choice in round i. To be able to progress we have ask for a larger quorum of $n - f'$ acceptors, where $f' = \lfloor (n-1)/3 \rfloor$.

Therefore, let's assume that the proposer that is executing round i has collected $n - f'$ promises from a set Q of exactly $n - f'$ acceptors. Again, let j be the highest *vrnd* in the promises. Moreover, let $Q_j \subseteq Q$ be the set of acceptors that last voted in round j and let $Q_j[v] \subseteq Q_j$ be the set of acceptors that last voted value v in round j. As we know, in Fast Paxos there is no guarantee that the acceptors in Q_j have last voted for the same value. Clearly, that means that we need to change the rule that Paxos uses to select the value to be sent in the $\langle$ACCEPT$\rangle$ message. We change the rule in the following way:

1. if $j = -1$ (no acceptor in Q has voted yet), select $\perp$ (start a fast round).
2. if $j \geq 0$ and there exists v such that $|Q_j[v]| \geq n - 2f'$, then select v (note that there can be at most one such value);
3. if $j \geq 0$ and for all v' we get $|Q_j[v']| < n - 2f'$, then select any value that has been last voted in round j.

In the following section we will see that this simple rule is enough to guarantee safety.

3.2 A simple proof of safety for Fast Paxos

Like in Paxos, our goal is to prove the three safety Properties CS1, CS2, and CS3. Again, Property CS1 and CS3 are very easy to check and we prove Property CS as a way to prove Property CS2.

THEOREM 2. *In Fast Paxos, if acceptor a has voted for value v at round i, then no value $v' \neq v$ can be chosen in any previous round.*

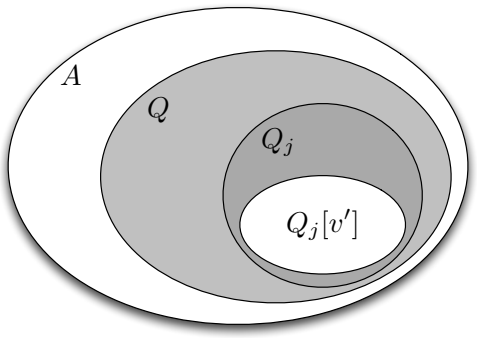


Figure 3: Fast Paxos: The acceptors in the gray areas have not voted for value v' in round j. The acceptors in $Q \setminus Q_j$ have promised not to vote any value in round j, and the acceptors in $Q_j \setminus Q_j[v']$ have voted in round j for some value different from v'.

PROOF. We prove Property CS by induction on round i. The base case, when $i = 0$, is trivially true. We now assume that the property is true for rounds $0, \ldots, i-1$ (the *inductive hypothesis*) and we prove the property for round i. Let $Q \subseteq A$, $|Q| = n - f'$, be the quorum of acceptors that sent the $\langle$PROMISE$\rangle$ message for round i, $j < i$ be the largest *vrnd* in the promises, $Q_j \subseteq Q$ be the set of acceptors who last voted in round j, and $Q_j[v'] \subseteq Q_j$ be the set of acceptors that have last voted value v' in round j. Just like in Paxos, we can easily see that no value can be chosen in rounds $j+1, \ldots, i-1$ (the acceptors in $A \setminus Q$ are not enough to form a quorum). If $j = -1$, then we are done with the inductive step and with the proof. So, let's assume that $j \geq 0$ and proceed.

Assume that acceptor a has voted for value v at round i. Then, for all $v' \neq v$ we know that $|Q_j[v']| < n - 2f'$. As a consequence, no value $v' \neq v$ can be chosen in round j since the acceptors in $Q \setminus Q_j[v']$, which are *strictly more* than $n - f' - (n - 2f') = f'$, have *not* voted for value v' in round j. Indeed, $Q \setminus Q_j[v'] = (Q \setminus Q_j) \cup (Q_j \setminus Q_j[v'])$, the acceptors in $Q \setminus Q_j$ have promised not to vote any value in round j, and the acceptors in $Q_j \setminus Q_j[v']$ have not voted for value v' by definition. (See Figure 3.)

Lastly, since at least one acceptor has voted for v in round j, no value $v' \neq v$ can be chosen in rounds $0, \ldots, j-1$ by using the inductive hypothesis on round j. This concludes the inductive step and the proof. □

Property CS2 easily follows from Property CS like in Paxos. Therefore, we are done and we can claim that Fast Paxos is safe.

3.3 Liveness of Fast Paxos

4. REFERENCES

[1] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43:685–722, July 1996.

[2] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.

[3] Leslie Lamport. The part-time parliment. *ACM Trans. on Comp. Syst.*, 16(2):133–169, 1998.

[4] Leslie Lamport. Paxos made simple. *ACM SIGACT News*, 32(4):18–25, December 2001.

[5] Leslie Lamport. Fast paxos. *Distributed Computing*, 19(2):79–103, 2006.