Stoppable Paxos

Dahlia Malkhi Leslie Lamport Lidong Zhou

April 28, 2008

Abstract

A stoppable state machine is one whose execution can be terminated by a special stopping command. Stoppable state machines can be used to implement reconfiguration in a replicated state machine; a reconfigurable state machine is implemented by a sequence of stoppable state machines, each running in a fixed configuration. Stoppable Paxos, a variant of the ordinary Paxos algorithm, implements a replicated stoppable state machine.

Contents

1 Introduction

State machine replication is a well-known method of implementing a fault-tolerant service [11, 14]. The service is described as a deterministic state machine that accepts client commands and produces outputs, and multiple replicas of the state machine are implemented. The different replicas operate independently and asynchronously. However, they all have the same initial state and execute the same sequence of commands, so they all produce the same sequence of outputs. Since each replica can respond to any client request, using $f + 1$ replicas allows the system to tolerate the failure of f processes.

Implementing a replicated state machine requires a fault-tolerant algorithm for choosing the sequence of state machine commands executed by the replicas. Such an algorithm must guarantee that, for each i, if a replica executes c as the i^{th} command in the command sequence, then (i) c was issued by a client, and (ii) no replica executes a different command as the i^{th} command in the sequence.

Different replicas operate asynchronously, so they may execute the same command at different times. Moreover, if the output produced by executing command number i does not depend on what commands are executed as numbers 1 through $i - 1$, then a replica may generate the output for command i before it generates the output for command $i - 1$. Hence, although the replicas all produce the same sequence of outputs, the outputs in that sequence could be generated in different orders.

An asynchronous algorithm for choosing the sequence of state machine commands requires at least $2f + 1$ processes to tolerate the (non-malicious) failure of f of them [4]. Hence, we need more processes to choose the commands than to execute the replicas. The processes that choose the sequence of commands are called *acceptors*, and the ones that execute the replicas are called *learners*.

We can choose a sequence of commands by using a separate consensus protocol to choose each one, where the i^{th} consensus protocol chooses the i^{th} command. The protocol used to choose the i^{th} command will be called the i^{th} consensus *instance*. Separate consensus instances need not have disjoint implementations—for example, messages belonging to separate instances may be batched in a single physical message. However, the separate instances are logically independent, which makes reasoning about their correctness easier.

Different commands in the command sequence can be chosen concurrently. Processes can begin the i^{th} consensus instance without waiting for instances 1 to $i - 1$ to terminate. This concurrent processing is vital to the efficiency of an asynchronous distributed system. For example, in a typical leader-based protocol, the current leader can send proposals for commands one after another, without waiting for acknowledgements of previous proposals.

In a static system, all consensus instances are instances of the same algorithm. In particular, they all use the same sets of acceptors and learners—sets we call the *configuration*. However, achieving long-term resilience requires a reconfigurable system, in which the configuration can change. A reconfigurable system requires that, for each i, there be agreement on the configuration that is to execute consensus instance i.

In state machine replication, reconfiguration has traditionally been done by using the state machine itself to perform special reconfiguration commands [10, 14, 15]. The obvious method is to have a reconfiguration command change the configuration for all subsequent instances until the next reconfiguration command. However, since a consensus instance cannot be executed until the configuration executing it is known, this prevents concurrent execution of different instances. We can permit concurrent execution of up to α consensus instances by instead letting a reconfiguration command executed as command number i determine the configuration starting from instance $i + \alpha$, where α is a system parameter [10]. Instances $i + 1$ through $i + \alpha$ can begin execution once commands 1 through i have been chosen. In practice, α can be made large enough so the system never has to wait to learn the current configuration, if no reconfiguration command has been issued. However, this approach has two somewhat awkward properties:

- To force a reconfiguration to happen quickly, a reconfiguration command must be followed by $\alpha - 1$ *no-op* commands that have no effect.

- If $\alpha > 1$, then several reconfiguration command could appear among commands i through $i+\alpha-1$, meaning that one configuration is choosing the next several configurations. This can happen when using a leader-based consensus algorithm if a failure causes multiple processes to each think it is the leader.

In this paper, we propose an alternative reconfiguration procedure based on *stoppable* state machines. A stoppable state machine has a special class of *stopping* commands that terminate the state machine. If a stopping command is chosen as the i^{th} command, then the complete sequence of chosen state machine commands has length i. That is, if a stopping command is chosen as command i, then no command can ever be chosen as command j

for $j > i$. We implement the system state machine by executing a sequence of stopping state machines. Each consensus instance of a single stopping state machine is executed by the same configuration. Reconfiguration is performed by stopping the current state machine and starting a new one with a new configuration. The stopping command specifies the configuration used to execute the new stopping state machine. The system's complete sequence of state machine commands is the concatenation of the command sequences of the individual stopping state machines. If one stopping state machine is terminated by executing a stopping command as command number i, then we can number the commands of the next stopping state machine starting with $i + 1$. This provides consecutive numbers for the commands in the system state machine.

This method of reconfiguring with stoppable state machines seems to correspond more closely to the way engineers have traditionally approached reconfiguration. It is similar to view changing in group communication [1, 2, 3, 5, 6, 7, 8, 12]. However, the purpose of this paper is to present Stoppable Paxos, an algorithm for implementing a stoppable state machine. We have discussed reconfiguration only to indicate why stoppable state machines may be useful. A detailed description of how stoppable state machines are used for reconfiguration and how they relate to group communication is beyond the scope of this paper.

Stoppable Paxos is a variant of Paxos [10]. Our goal was to devise an algorithm that is as efficient as Paxos in the absence of a stopping command. This is not easy to do because Paxos allows the choosing of the i^{th} command to be performed concurrently for different values of i. We must avoid the possibility that the i^{th} command is chosen and a stopping command is concurrently chosen as the j^{th} command for $j < i$. The obvious method is to delay the choice of the i^{th} command until all previous commands are chosen, but this would considerably degrade the performance. Stoppable Paxos adds no messages or delays to ordinary Paxos, except that a leader cannot propose an i^{th} command if, in the normal course of execution, it learns that a stopping command has been chosen or was proposed and may have been chosen as the j^{th} command for some $j < i$. Although the basic idea of the algorithm is not complicated, getting the details right was not easy.

The following section reviews ordinary Paxos. The Stoppable Paxos algorithm is described in Section 3, and its correctness properties are stated in Section 4. A proof of correctness appears in the appendix for reading at the program committee's discretion.

2 Paxos Revisited

Ordinary Paxos assumes a distributed system of processes communicating by messages. Processes can fail only by stopping, and messages can be lost or duplicated but not corrupted. Timely actions by non-failed processes and timely delivery of messages among them is required for progress; safety is maintained despite arbitrary delays and any number of failures.

The core of Paxos is a consensus algorithm (originally called the Synod algorithm). In a consensus algorithm, client processes can propose values, and learner processes each learn a value. We use the term *command* for a value that may be proposed. A consensus algorithm must satisfy two safety properties:

Consistency No two learners can learn different commands.

Nontriviality Any command learned must have been proposed.

For almost all consensus algorithms, including Paxos, nontriviality is easily seen to hold. We therefore ignore it and consider only consistency. A consensus algorithm must also ensure that, under some suitable hypothesis, a command is learned.

A replicated state machine is implemented with a sequence of instances of a consensus algorithm, the i^{th} instance choosing the i^{th} state-machine command. We briefly review the Paxos consensus algorithm and how it is used to implement a state machine. We consider the static case, in which the same processes implement all consensus instances.

2.1 The Paxos Consensus Protocol

The Paxos consensus algorithm assumes three sets of processes: *leaders* that propose commands, *acceptors* that choose a command, and *learners* that learn the chosen command. These sets are not necessarily disjoint—in particular, leaders are usually learners. We ignore the clients, which provide commands for the leaders to propose. Leaders propose commands in numbered *ballots.* For simplicity, we take ballot numbers to be natural numbers. A configuration assigns to each ballot a unique leader that performs actions of that ballot. For example, the leader of ballot number b may be determined by the low-order bits of b. We also assume certain sets of acceptors to be *quorums*, subject only to the requirement that the intersection of any pair of quorums is non-empty.

Acceptors accept and store proposed commands and their ballot numbers. In particular, each acceptor a maintains the value $bal[a]$ that records the highest ballot number that a has received, "voted for", and acknowledged, and the value $vote[a][b]$ that records the command proposed with ballot number b that a has voted for. Initially, $bal[a]$ equals $-\infty$ and $vote[a][b]$ equals $\top$, a special value that is not a command.

For any acceptor a, let $maxbal(a)$ be the largest ballot number b for which $vote[a][b] \neq \top$, and to equal $-\infty$ if $vote[a][b] = \top$ for all b. Define $maxvote(a)$ to equal $vote[a][maxbal(a)]$, or $\top$ if $maxbal(a) = -\infty$. Instead of maintaining the entire array $vote[a]$, acceptor a need only record the values $maxbal(a)$ and $maxvote(a)$. For simplicity, we ignore this optimization.

At any point during the execution of the consensus algorithm, the state consists of the values of the arrays bal and $vote$ and the sets of messages that have been sent and received by the processes. A *state function* is an expression whose value depends on the state.

A ballot consists of two phases, each with two sub-phases. In the first phase, the leader determines whether a command may have been chosen in a lower-numbered ballot. In the second phase, it proposes a command and the acceptors vote for that command. The command is chosen if a quorum of acceptors vote for it.

The heart of the algorithm is the state function $val2a(b, Q)$, which the ballot b leader computes on the basis of messages it receives in phase 1 from acceptors in the quorum Q. If $val2a(b, Q)$ equals a command v, then v might have been chosen in a lower-numbered ballot and the leader must propose it in phase 2. If $val2a(b, Q)$ equals $\top$, then no command has been or ever will be chosen in a lower-numbered ballot, and the leader can propose any value. We define $val2a$ below. First, we describe the following actions that the algorithm can perform.

$Phase1a(b)$ The leader of ballot number b sends the message $\langle \text{“}\texttt{1a}\text{”}, b \rangle$ to all acceptors. (This action is always enabled.)

$Phase1b(a, b)$ When acceptor a receives a $\langle \text{“}\texttt{1a}\text{”}, b \rangle$ message with $b > bal[a]$, it sets $bal[a]$ to b and sends the message

$$\langle \text{“}\texttt{1b}\text{”},\ a,\ b,\ \langle maxbal(a), maxvote(a) \rangle \rangle$$

to the ballot b leader. (The acceptor ignores a $\langle \text{“}\texttt{1a}\text{”}, b \rangle$ message if $bal[a] \geq b$.)

$Phase2a(b, v, Q)$ This action is performed by the ballot b leader, for a command v and quorum Q. It is enabled iff the following three conditions are satisfied:

$E1(b, Q)$ The leader has received a message of the form $\langle\,$“1b”$, a, b, r\rangle$ from every acceptor a in Q.

$E2(b)$ The leader has not executed a $Phase2a(b, w, U)$ action for any w and any quorum U.

$E3(b, Q, v)$ If $val2a(b, Q) \neq \top$ then $v = val2a(b, Q)$.

The action sends the message $\langle\,$“2a”$, b, v\rangle$ to all acceptors.

$Phase2b(a, b, v)$ When acceptor a receives a $\langle\,$“2a”$, b, v\rangle$ message from the ballot b leader and $bal[a] \leq b$, it sets $bal[a]$ to b and $vote[a][b]$ to v and it sends a $\langle\,$“2b”$, b, v\rangle$ message to the learners. (The $\langle\,$“2a”$, b, v\rangle$ message is ignored if $bal[a] > b$.)

We omit the action by which a learner learns a command. It is enabled by the receipt of a $\langle\,$“2b”$, a, b, v\rangle$ message from every acceptor a in a quorum. Instead, we say that a command v is *chosen* if there exists a ballot number b and a quorum Q such that $vote[a][b] = v$ for all a in Q. Consistency is obviously satisfied by ensuring that, if any commands v and w are chosen, then $v = w$.

The state function $val2a(b, Q)$ is defined as follows. Let R be the set of all r such that the ballot b leader has received from some acceptor a in Q the message $\langle\,$“1b”$, a, b, r\rangle$. (The elements of R are pairs $\langle c, v\rangle$ with either c a ballot number and v a command, or $c = -\infty$ and $v = \top$.) Let $\langle c, v\rangle$ be an element of R such that $c \geq d$ for all $\langle d, w\rangle \in R$, and define $val2a(b, Q)$ to equal v. (For any state reachable during execution of the algorithm, $\langle c, v\rangle \in R$ and $\langle c, w\rangle \in R$ imply $v = w$, so this uniquely defines $val2a(b, Q)$.) For later reference, we also define $mbal2a(b, Q)$ to equal c.

Although the algorithm executes a sequence of ballots, those ballots need not be executed sequentially. It is possible for two or more leaders to be executing $Phase1a$ and/or $Phase2a$ actions concurrently, and for the resulting messages to be received by different acceptors in different orders. This can impede progress but cannot cause inconsistency.

To achieve progress, Paxos uses some algorithm to select a unique active leader. The active leader starts a new ballot with a number higher than that of any other ballot it knows to have been started. An acceptor a informs the leader it has chosen too low a ballot number if it receives a ballot b message with $b < bal[a]$, causing the leader to choose a higher-numbered ballot. This achieves progress if there is a unique active leader and a quorum of acceptors that are nonfaulty and can communicate in a timely fashion. For most systems, it is easy to devise a leader-selection algorithm that works properly when the system is behaving normally.

2.2 The Paxos State Machine Implementation

Paxos executes a sequence of instances of the Paxos consensus algorithm. For each instance i, it maintains an array vote_i, where $\mathit{vote}_i[a][b]$ is the value of $\mathit{vote}[a][b]$ for instance i of the consensus algorithm. Paxos achieves its efficiency by executing Phase 1 simultaneously for all instances of the consensus algorithm as follows, using the same value of $\mathit{bal}[a]$ for all of them. More precisely:

1. The ballot b leader simultaneously executes $\mathit{Phase}1a(b)$ for all instances, sending a single $\mathit{Phase}1a$ message to each acceptor.

2. Upon receipt of a $\mathit{Phase}1a$ message, an acceptor simultaneously executes $\mathit{Phase}1b$ actions for all instances, bundling the infinite set of $\mathit{Phase}1b$ messages in a single physical message. (That physical message contains only a finite amount of information because, for any acceptor a and ballot number b, the value of $\mathit{vote}_i[a][b]$ is $\top$ for all but a finite number of instances i.)

We add an extra instance parameter to the $\mathit{Phase}2a$ and $\mathit{Phase}2b$ actions and to the $\mathit{val}2a$ and $\mathit{mbal}2a$ state functions. For example, $\mathit{Phase}2a(i, b, v, Q)$ is the $\mathit{Phase}2a(b, v, Q)$ action of instance i and $\mathit{val}2a(i, b, Q)$ is the state function $\mathit{val}2a(b, Q)$ of instance i. We subscript messages with instance numbers, so $\langle \text{“}\mathtt{1b}\text{”}, \ldots \rangle_i$ is a $\mathit{Phase}1b$ message sent for instance i (and bundled with $\mathit{Phase}1b$ messages sent for other instances).

In normal operation, there is a single active leader that receives client commands and performs $\mathit{Phase}2a$ actions for them. When the active leader fails, a new active leader is selected that performs a $\mathit{Phase}1a(b)$ action for a new ballot number b higher than any that has been used so far. When the active leader receives $\mathit{Phase}1b$ messages from a quorum Q, for each i it finds either:

1. $\mathit{val}2a(i, b, Q)$ is a command v, meaning that v may have been chosen in instance i at some ballot less than b.

2. $\mathit{val}2a(i, b, Q) = \top$, meaning that no command can have been (or can ever be) chosen in instance i at any ballot less than b.

In case 1, it performs a $\mathit{Phase}2a(i, b, v, Q)$ action to try to get v chosen. Let k be the largest instance for which this case holds, so it is the highest instance in which any acceptor in Q has voted. For all instances i with $i < k$ such that $\mathit{val}2a(i, b, Q) = \top$, the leader performs a $\mathit{Phase}2a(i, b, \mathit{noop}, Q)$ action to try to choose a noop command that does nothing. Without waiting for responses to its $\mathit{Phase}2a$ messages, the leader can begin performing $\mathit{Phase}2a$ actions in instances higher than k for client commands.

It is possible for a leader to learn a set of commands that have already been chosen and to optimize this procedure to avoid unnecessary actions for those chosen commands. This optimization is straightforward and we will not discuss it.

Remember that what we have just described is how Paxos works in the normal case when there is a single active leader. Consistency is maintained even if multiple processes believe themselves to be the active leader. A single active leader is required only to ensure progress.

3 The Stoppable Paxos Algorithm

Stoppable Paxos uses the same variables and sends the same messages as Paxos. Before describing the actual algorithm, we sketch how the Stoppable Paxos algorithm works in the normal case when a (single) new active leader is selected.

As in ordinary Paxos, the new active leader performs a $\mathit{Phase}1a(b)$ action for a suitable ballot number b. The algorithm differs from Paxos if the leader finds that a stopping command stp might have been chosen in some instance i. In that case, the leader performs $\mathit{Phase}2a$ actions for lower-numbered instances as before. However, to ensure that the state machine stops when it should, we must ensure that the leader does not perform a $\mathit{Phase}2a$ action for any instance greater than i if the stopping command actually was chosen in instance i.

The problem is to decide what the leader should do if it finds $\mathit{val}2a(i, b, Q)$ equal to a stopping command stp and $\mathit{val}2a(j, b, Q)$ equal to any command, for some i and j with $j > i$. The answer depends on the values of $\mathit{mbal}2a(i, b, Q)$ and $\mathit{mbal}2a(j, b, Q)$. Remember that, for any k, the value of $\mathit{mbal}2a(k, b, Q)$ is the highest ballot number less than b for which some acceptor a in Q set $\mathit{voted}_k[a]$. If $\mathit{mbal}2a(j, b, Q) > \mathit{mbal}2a(i, b, Q)$, then the stopping command c could not have been chosen in (a lower ballot of) instance i, so stp is *voided*—meaning that the leader acts as if $\mathit{val}2a(i, b, Q)$ equals $\top$. Otherwise, the leader performs a $\mathit{Phase}2a(i, b, \mathit{stp}, Q)$ action to try to get stp chosen and does nothing in any higher-numbered instance, including instance j.

If the leader performs a $\mathit{Phase}2a$ action for a stopping command in instance i, then it performs no $\mathit{Phase}2a$ actions for instances greater than i. Otherwise, it begins processing new client commands as in ordinary Paxos.

It continues until it performs a $Phase2a$ action for a stopping command, whereupon it performs no further $Phase2a$ actions for any higher-numbered instance. Except when the leader is prevented from performing $Phase2a$ actions because of a stopping command, Stoppable Paxos allows all the concurrent execution that ordinary Paxos does.

We now begin our description of the actual Stoppable Paxos algorithm. As in ordinary Paxos, the algorithm can be optimized to take advantage of knowledge of already-chosen commands. For simplicity, we ignore this optimization. Stoppable Paxos then differs from Paxos only in the enabling conditions of the $Phase2a$ action. We begin with an intuitive description of these enabling conditions, which are labeled $E1$–$E6$.

Conditions $E1$–$E3$ are the same as for ordinary Paxos except that, in $E3$, we replace $val2a$ by a new state function $sval2a$. Recall that $E3$ requires $val2a(i, b, Q)$ to be the proposed command if it does not equal $\top$, because in that case it might have been chosen in a lower-numbered ballot. We will define $sval2a(i, b, Q)$ to be the same as $val2a(i, b, Q)$ except that it equals $\top$ if $val2a(i, b, Q)$ is a stopping command that is voided. As indicated above, a stopping command is voided if information about higher-numbered instances implies that the command could not have been chosen in this instance in a lower-numbered ballot.

Enabling condition $E4$ applies iff v is a stopping command, in which case it requires the two conditions:

$E4a$ A $Phase2a$ action must not have been performed for ballot b of a higher-numbered instance.

$E4b$ If the leader was not forced (by the value of $sval2a$) to propose the stopping command, then it must not be forced to propose any command in a higher-numbered instance.

Condition $E5$ asserts that the leader has not performed a $Phase2a$ action for a stopping command in ballot b of a lower-numbered instance, and condition $E6$ asserts that the value of $sval2a$ does not force the leader to propose such a value.

It appears that progress is impossible if the ballot b leader is forced (by $E3$) to propose a stopping command in an instance i and to propose some command in another instance $j > i$. If it executes the $Phase2a$ action for instance i, then $E5$ prevents it from executing the $Phase2a$ action for instance j. If it executes the action for instance j first, then $E4a$ prevents it from executing the action for instance i. This situation is prevented by voiding. The definition of $sval2a$ ensures that the $Phase1b$ messages for instance j void the stopping command in instance i.

Unlike in ordinary Paxos, in Stoppable Paxos the separate consensus instances are not logically separate. Enabling conditions $E4$–$E6$ and the definition of $sval2a$ for a ballot in one instance depend on $Phase2a$ actions performed and $Phase1b$ messages received for that ballot in other instances. However, this implies no extra messages or delays. As in ordinary Paxos, the $Phase1b$ messages for all instances are bundled together; and no enabling condition requires that a $Phase2a$ action for another instance be done first. The enabling conditions require only that certain actions *not* have been done.

We now precisely define $sval2a$ and $E1$–$E6$. For clarity, we write mathematical formulas in mathematics, using English only where necessary to avoid distracting formalization. We let the range over which a variable is quantified, if not stated explicitly, depend on the variable name as follows:

$i,\ j,\ k$: instance numbers	$b,\ c$: ballot numbers	Q : quorums
$u,\ v,\ w$: commands	$a,\ q$: acceptors	

We use customary abbreviations such as $\exists\, i < j \ :\ P$ for $\exists\, i\ :\ (i < j) \wedge P$. The definition of $sval2a$ is:

$$\begin{array}{lll} sval2a(i, b, Q) & \triangleq & \text{IF} \quad (val2a(i, b, Q) \in StopCmd) \\ & & \quad\ \wedge\ (\exists\, j > i \ :\ mbal2a(j, b, Q) \geq mbal2a(i, b, Q)) \\ & & \text{THEN}\ \ \top \\ & & \text{ELSE}\ \ val2a(i, b, Q) \end{array}$$

Define $Done2a(i, b, v)$ to be the state function that is true iff a $Phase2a(i, b, v, Q)$ action has been executed for some quorum Q. (More precisely, it is true iff there is a $\langle \text{“2a”}, b, v\rangle_i$ message in the set of sent messages.) The enabling conditions of the $Phase2a(i, b, v, Q)$ action are:

$E1(b, Q) \ \triangleq\ \forall\, a \in Q,\ i\ :$ the ballot b leader has received a message of the form $\langle \text{“1b”}, a, b, r\rangle_i$ from a

$E2(i, b) \ \triangleq\ \forall\, w\ :\ \neg Done2a(i, b, w)$

$E3(i, b, Q, v) \ \triangleq\ (sval2a(i, b, Q) \neq \top) \Rightarrow (v = val2a(i, b, Q))$

$E4(i, b, Q, v) \ \triangleq\ (v \in StopCmd) \ \Rightarrow\ E4a(i, b, v) \wedge E4b(i, b, Q, v)$

where

$E4a(i, b, v) \ \triangleq\ \forall\, j > i,\ w\ :\ \neg Done2a(j, b, w)$

$$\mathit{E4b}(i, b, Q, v) \triangleq \forall j > i : \\ (\mathit{sval2a}(i, b, Q) = \top) \Rightarrow (\mathit{sval2a}(j, b, Q) = \top)$$

$$\mathit{E5}(i, b) \triangleq \forall j < i,\ w \in \mathit{StopCmd} : \neg \mathit{Done2a}(j, b, w)$$

$$\mathit{E6}(i, b, Q) \triangleq \forall j < i : \\ (\mathit{sval2a}(j, b, Q) \neq \top) \Rightarrow (\mathit{sval2a}(j, b, Q) \notin \mathit{StopCmd})$$

The ballot b leader can propose commands in different instances in any order. This applies to a stopping command as well. In particular, the leader can propose a stopping command as command number i and then propose lower-numbered commands. Since stopping commands are used for reconfiguration, this allows the leader to let i be the next available command number for reconfigurations that must occur quickly and to be larger for reconfigurations that may occur lazily.

4 Correctness

We now state the correctness properties satisfied by Stoppable Paxos. A rigorous informal proof of these properties appears in the appendix. We have also written a formal hand proof that gives us greater confidence in the algorithm's correctness than such an informal proof can provide.

First, we define $\mathit{Chosen}(i, b, v)$ to assert that command v is chosen in ballot b of instance i. As in ordinary Paxos, $\mathit{Chosen}(i, b, v)$ is defined to be true iff there is a quorum Q such that $\mathit{vote}_i[a][b] = v$ holds for all a in Q.

$$\mathit{Chosen}(i, b, v) \triangleq \exists\, Q : \forall\, a \in Q : \mathit{vote}_i[a][b] = v$$

Our algorithm satisfies the same consistency property as ordinary Paxos plus the property that a stopping command stops the state machine. These properties are expressed by the invariance of the following state predicates.

$$\mathit{Consistency} \triangleq \forall\, i, b, c, v, w : \mathit{Chosen}(i, b, v) \wedge \mathit{Chosen}(i, c, w) \Rightarrow (b = c)$$

$$\mathit{Stopping} \triangleq \forall\, i,\ j < i,\ v \in \mathit{StopCmd},\ w : \\ \mathit{Chosen}(j, b, v) \Rightarrow \neg \mathit{Chosen}(i, c, w)$$

Like ordinary Paxos, our Stoppable Paxos assures progress if eventually there is a unique leader for a high enough ballot number that is nonfaulty and can communicate with a nonfaulty quorum. The precise property we prove is that the following condition holds, for all b and Q.

$$
\begin{array}{l}
\mathit{Progress}(b, Q) \;\triangleq \\
\quad P1(b, Q) \land P2(b, Q) \land P3(b) \Rightarrow \\
\qquad\qquad \text{eventually} \quad (\exists\, v : \mathit{Chosen}(i, b, v)) \\
\qquad\qquad\qquad\qquad\quad \lor\; (\exists\, j < i,\; v \in \mathit{StopCmd} : \mathit{Chosen}(j, b, v))
\end{array}
$$

where

$P1(b, Q) \triangleq$ No ballot b action of the ballot b leader or of an acceptor in Q can become forever enabled and never executed.

$P2(b, Q) \triangleq$ Every ballot b message sent between the ballot b leader and the acceptors in Q is eventually received.

$P3(b) \triangleq \forall\, c > b$: No $\mathit{Phase}1a(c)$ action is ever executed.

Condition $P1(b, Q)$ means that the ballot b leader eventually executes the $\mathit{Phase}1a(b)$ action, and that it and the acceptors in Q perform ballot b actions that are enabled by the receipt of messages. Condition $P2(b, Q)$ is satisfied if eventually the leader and the acceptors in Q are nonfaulty and communicate reliably with one another, using a retransmission protocol to recover from lost messages. Condition $P3(b)$ asserts that no ballot numbered greater than b is ever started.

Conditions $P1$–$P3$ are the same ones under which ordinary Paxos achieves progress. As with ordinary Paxos, they are satisfied in practice by using a leader-selection algorithm. However, because of the extra enabling conditions in the $\mathit{Phase}2a$ action, the proof that they ensure progress is more difficult.

References

[1] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella. The Totem single-ring ordering and membership protocol. *tocs*, 13(4):311–342, November 1995.

[2] Özalp Babaoglu, Renzo Davoli, and Alberto Montresor. Group communication in partitionable systems: Specification and algorithms. *IEEE Transactions on Software Engineering*, 27(4):308–336, 2001.

[3] Kenneth P. Birman and Thomas A. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47–76, February 1987.

[4] Bernadette Charron-Bost and André Schiper. Uniform consensus is harder than consensus (extended abstract). Technical Report

DSC/2000/028, École Polytechnique Fédérale de Lausanne, Switzerland, May 2000.

[5] Gregory V. Chockler, Idit Keidar, and Roman Vitenberg. Group communication specifications: A comprehensive study. *ACM Computing Surveys*, 33(4):427–469, December 2001.

[6] Danny Dolev and Dalia Malki. The Transis approach to high availability cluster communication. *Communications of the ACM*, 39(4):64–70, April 1996.

[7] Alan Fekete, Nancy Lynch, and Alex Shvartsman. Specifying and using a partitionable group communication service. *ACM Transactions on Computer Systems*, 19(2):171–216, May 2001.

[8] Idit Keidar and Danny Dolev. Efficient message ordering in dynamic networks. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, New York, NY, May 1996. ACM.

[9] Leslie Lamport. How to write a proof. *American Mathematical Monthly*, 102(7):600–608, August-September 1995.

[10] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.

[11] Butler W. Lampson. How to build a highly available system using consensus. In Ozalp Babaoglu and Keith Marzullo, editors, *Distributed Algorithms*, volume 1151 of *Lecture Notes in Computer Science*, pages 1–17, Berlin, 1996. Springer-Verlag.

[12] L. E. Moser, Y. Amir, P. M. Melliar-Smith, and D. A. Agarwal. Extended virtual synchrony. In *The 14th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 56–65, 1994.

[13] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on the Foundations of Computer Science*, pages 46–57. IEEE, November 1977.

[14] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.

[15] J. Wensley et al. SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(10):1240–1254, October 1978.

Appendix: The Proof of Correctness

We now prove that Stoppable Paxos satisfies its safety and liveness properties. For clarity and conciseness, we write simple temporal logic formulas with two temporal operators: $\Box$ meaning *always*, and $\Diamond$ meaning *eventually* [13]. We use a linear-time logic, so $\Diamond$ can be defined by $\Diamond F \triangleq \neg\Box\neg F$, for any formula F. For a state predicate P, the formula $\Box P$ asserts that P is an invariant, meaning that it is true for every reachable state. The temporal formula $\Diamond\Box P$ asserts that at some point in the execution, P holds from that point onward.

We define a predicate P to be *stable* iff it satisfies the following condition: if P is true in any reachable state s, then P is true in any state reachable from s by any action of the algorithm. We let $\mathsf{stable}\, P$ be the assertion that state predicate P is stable. It is clear that a stable predicate is invariant if it is true in the initial state. Because stability is an assertion only about reachable states s, we can assume that all invariants of the algorithm are true in state s when proving stability.

Our proofs are informal, but careful. The two complicated, multi-page proofs are written with a hierarchical numbering scheme in which $\langle x \rangle y$ is the number of the y^{th} step of the current level-x proof [9]. Although it may appear intimidating, this kind of proof is easy to check and helps to avoid errors.

A.1 The Proof of Safety

We now prove that *Consistency* and *Stopping* are invariants of Stoppable Paxos. First, we define:

$$
\begin{aligned}
&\mathit{NotChoosable}(i, b, v) \;\triangleq\\
&\qquad\quad (\, \exists\, Q \,:\, \forall\, a \in Q \,:\, (\mathit{bal}[a] > b) \land (\mathit{vote}_i[a][b] \neq v)\,)\\
&\qquad \lor\; (\, \exists\, j < i,\; w \in \mathit{StopCmd} \,:\, \mathit{Done2a}(j, b, w)\,)\\
&\qquad \lor\; (\, (v \in \mathit{StopCmd}) \land (\exists\, j > i,\; w \,:\, \mathit{Done2a}(j, b, w))\,)
\end{aligned}
$$

We next prove a number of simple invariance and stability properties of the algorithm.

Lemma 1

1. $\forall\, i, b, v \,:\, \Box\,(\mathit{Chosen}(i, b, v) \Rightarrow \mathit{Done2a}(i, b, v))$.
2. $\forall\, i, b, v, w \,:\, \Box\,((\mathit{Done2a}(i, b, v) \land \mathit{Done2a}(i, b, w) \Rightarrow (v = w))$
3. $\forall i, b, a, v \,:\, \Box\,((\mathit{vote}_i[a][b] = v) \Rightarrow \mathit{Done2a}(i, b, v))$

4. $\forall i, b, v, a, q : \Box((vote_i[a][b] = v) \Rightarrow (vote_i[q][b] \in \{v, \top\}))$

5. (a) $\forall\, i, a, b, v : \textsf{stable}\,((bal[a] > b) \land (vote_i[a][b] = v))$
   (b) $\forall\, i, a, b : \textsf{stable}\,((bal[a] > b) \land (vote_i[a][b] = \top))$

6. $\forall\, i,\ j < i,\ b,\ w \in StopCmd,\ v :$
   $\Box\,(Done2a(j, b, w) \Rightarrow \neg Done2a(i, b, v))$

7. $\forall\, i,\ b,\ v : \textsf{stable}\, NotChoosable(i, b, v)$

8. $\forall\, i, b, Q : \Box\,(E1(b, Q) \Rightarrow (mbal2a(i, b, Q) < b))$

Proof:

1. $Chosen(i, b, v)$ implies that $vote_i[a][b] = v$ for some acceptor a, which implies a received a $\langle \text{“2a”}, b, v \rangle_i$ message, which implies $Done2a(i, b, v)$.

2. This follows from enabling condition $E2$ for the $Phase2a$ action.

3. $vote_i[a][b] = v$ implies that acceptor a must have received the $Phase2a$ message sent by executing $Phase2a(i, b, v, Q)$ for some quorum Q.

4. This follows from Lemmas 1.2 and 1.3.

5. No action decreases $bal[a]$, and $vote_i[a][b]$ is changed to (a command) u only by a $Phase2b(i, a, b, u)$ action, which is enabled only if $bal[a] \leq b$.

6. $Done2a(j, b, w) \Rightarrow \neg Done2a(i, b, v)$ is obviously true initially. It is stable because enabling condition $E4a(j, b, w)$ of $Phase2a(j, b, w, Q)$ implies that $Done2a(j, b, w)$ can become true only when $\neg Done2a(i, b, v)$ is true, and enabling condition $E5(i, b)$ of $Phase2a(i, b, v, Q)$ implies that $\neg Done2a(i, b, v)$ can become false only when $Done2a(j, b, w)$ is false.

7. It suffices to show that each of the disjuncts in the definition of $NotChoosable(i, b, v)$ is stable. The first disjunct is the conjunction of formulas $(bal[a] > b) \land (vote_i[a][b] \neq v)$, each of which can be written as the conjunction of formulas $(bal[a] > b) \land (vote_i[a][b] = w)$ (for w a command or $\top$) which are stable by part 5 of this lemma. The stability of the second and third conjuncts follows easily from the obvious stability of $Done2a(j, b, w)$ for all j and w.

8. An acceptor a changes $vote_i[a][b]$ only by performing a $Phase2b$ action that sets $bal[a]$ to b. Because $bal[a]$ is never decreased, $(vote_i[a][b] \neq \top) \Rightarrow (bal[a] \geq b)$ is an invariant. A $\langle \text{“1b”}, a, b, \langle c, v \rangle \rangle_i$ message is sent by a $Phase1b(a, b)$ action that is enabled only if $b > bal[a]$, so $c < b$ for any such message. The definition of $mbal2a$ then implies that $mbal2a(i, b, Q) < b$ if some acceptor in Q has sent a $Phase1b$ message for ballot b of instance i, which is the case if $E1(b, Q)$ is true. □

We now prove some less obvious invariants.

Lemma 2 $\forall\, i, b, v : \Box\,(NotChoosable(i, b, v) \Rightarrow \neg Chosen(i, b, v))$

Proof: We assume $\mathit{NotChoosable}(i, b, v)$ is true in a reachable state (so all invariants are true) and prove $\neg\mathit{Chosen}(i, b, v)$. By definition of *NotChoosable*, there are three cases to consider.

1. Case: $\exists\, Q : \forall\, a \in Q : (\mathit{bal}[a] > b) \land (\mathit{vote}_i[a][b] \neq v)$

   Proof: Since any two quorums have non-empty intersection, any quorum contains an acceptor a in Q, for which the case assumption implies $\mathit{vote}_i[a][b] \neq v$. By definition of *Chosen*, this implies $\neg\mathit{Chosen}(i, b, v)$.

2. Case: $\exists\, j < i,\ w \in \mathit{StopCmd} : \mathit{Done2a}(j, b, w)$

   Proof: Lemma 1.6 implies $\neg\mathit{Done2a}(i, b, v)$, and Lemma 1.1 then implies $\neg\mathit{Chosen}(i, b, v)$.

3. Case: $\exists\, j > i,\ w : \mathit{Done2a}(j, b, w)$.

   Proof: Lemma 1.6 (with $i \leftrightarrow j$ and $v \leftrightarrow w$) implies $\neg\mathit{Done2a}(i, b, v)$, and Lemma 1.1 then implies $\neg\mathit{Chosen}(i, b, v)$. □

Lemma 3 $\forall\, i,\ b,\ c < b,\ w,\ Q$:

$$\begin{array}{l} \Box\,(\,\mathit{Hyp}(i, b, c, Q) \land E1(b, Q) \Rightarrow \mathit{NotChoosable}(i, c, w)\,) \\ \text{where } \mathit{Hyp}(i, b, c, Q) \ \triangleq \\ \qquad (\mathit{mbal2a}(i, b, Q) < c) \\ \qquad \lor\ (\,(\mathit{mbal2a}(i, b, Q) = c) \land (w \neq \mathit{val2a}(i, b, Q))\,) \end{array}$$

Proof: We assume $c < b$, $\mathit{Hyp}(i, b, c, Q)$, and $E1(b, Q)$ and prove $\mathit{NotChoosable}(i, c, w)$. Assumption $E1(b, Q)$ implies that every acceptor a in Q has sent a “1b” message for ballot b of instance i. Since $\mathit{Phase1b}(a, b)$ is enabled only if $b > \mathit{bal}[a]$ and sets $\mathit{bal}[a]$ to b, acceptor a can have sent only one such “1b” message. Let $\langle \text{“1b”}, a, b, \langle b_a, v_a \rangle \rangle_i$ be that message. We consider the two disjuncts of the assumption $\mathit{Hyp}(i, b, c, Q)$ separately.

1. Case: $\mathit{mbal2a}(i, b, Q) < c$

   Proof: Let a be any acceptor in Q. The case assumption implies $b_a < c$, so $\mathit{val}_i[a][c]$ equaled $\top$ when a executed its $\mathit{Phase1b}(a, b)$ action. That action made $\mathit{bal}[a] = b$ true, so $c < b$ and Lemma 1.5 imply $\mathit{val}_i[a][c] = \top$ is still true. Every quorum contains an acceptor a in Q, for which we have shown that $\mathit{val}_i[a][c] = \top$, so $\mathit{NotChoosable}(i, c, w)$ is true.

2. Case: $\mathit{mbal2a}(i, b, Q) = c$ and $w \neq \mathit{val2a}(i, b, Q)$

   Proof: Let a be any acceptor in Q. The assumption $\mathit{mbal2a}(i, b, Q) = c$ implies $b_a \leq c$. The definitions of b_a and v_a imply that, when a executed its $\mathit{Phase1b}(a, b)$ action, the value of $\mathit{vote}_i[a][c]$ was v_a if $b_a = c$ and was $\top$ if $b_a < c$. Since the action set $\mathit{bal}[a]$ to b and $c < b$, Lemma 1.5 implies that $\mathit{vote}_i[a][c]$ still has that value. If $b_a = c$, then Lemma 1.4 and the definition of *val2a* imply $v_a = \mathit{val2a}(i, b, Q)$. The case assumption $w \neq \mathit{val2a}(i, b, Q)$ therefore implies that $\mathit{vote}_i[a][c] \neq w$ for all acceptors a in Q. Since every quorum contains an acceptor in a, this implies $\mathit{NotChoosable}(i, c, w)$. □

We now make some more definitions, culminating in the key invariant $\mathit{PropInv}(i, b, v)$.

$$\mathit{SafeAt}(i, b, v) \triangleq \forall\, c < b,\; w \neq v : \mathit{NotChoosable}(i, c, w)$$

$$\begin{aligned}&\mathit{NoReconfigBefore}(i, b) \triangleq \\ &\qquad \forall\, j < i,\; c \leq b,\; w \in \mathit{StopCmd} : \mathit{NotChoosable}(j, c, w)\end{aligned}$$

$$\begin{aligned}&\mathit{NoneChoosableAfter}(i, b, v) \triangleq \\ &\qquad (v \in \mathit{StopCmd}) \Rightarrow \forall\, j > i,\; c < b,\; w : \mathit{NotChoosable}(j, c, w)\end{aligned}$$

$$\begin{aligned}\mathit{PropInv}(i, b, v) \triangleq \mathit{Done2a}(i, b, v) \Rightarrow &\quad \mathit{SafeAt}(i, b, v) \\ &\wedge \mathit{NoReconfigBefore}(i, b) \\ &\wedge \mathit{NoneChoosableAfter}(i, b, v)\end{aligned}$$

The heart of the safety proof is the following proof that *PropInv* is invariant.

Lemma 4 $\Box(\forall\, i, b, v : \mathit{PropInv}(i, b, v))$

PROOF: $\forall, i, b, v : \mathit{PropInv}(i, b, v)$ is true in the initial state because $\mathit{Done2a}(\ldots)$ is initially false. We therefore need only show that it is stable. We do this by assuming that it is true in a state s and proving it is true in state t. For any state function f we let f be its value in state s and f' be its value in state t.

$\langle 1 \rangle 1$. It suffices to

ASSUME: 1. $\forall\, j, c, w : \mathit{PropInv}(j, c, w)$

2. i is an instance number, b a ballot number, v a command, and Q a quorum.

3. $s \rightarrow t$ is a $\mathit{Phase2a}(i, b, v, Q)$ step.

4. $E1(b, Q)$

PROVE: $\quad \mathit{SafeAt}(i, b, v)'$

$\wedge\ \mathit{NoReconfigBefore}(i, b)'$

$\wedge\ \mathit{NoneChoosableAfter}(i, b, v)'$

PROOF: To prove $(\forall, i, b, v : \mathit{PropInv}(i, b, v))'$, it suffices to prove it for a particular i, b, and v. It follows from Lemma 1.7 (the stability of $\mathit{NotChoosable}(\ldots)$) that

$$\mathit{SafeAt}(i, b, v) \wedge \mathit{NoReconfigBefore}(i, b) \wedge \mathit{NoneChoosableAfter}(i, b, v)$$

is stable. Hence, the first step that can possibly make $\mathit{PropInv}(i, b, v)$ false is one that makes $\mathit{Done2a}(i, b, v)$ true. We can therefore assume $s \rightarrow t$ is a $\mathit{Phase2a}(i, b, v, Q)$ step for some quorum Q. Formula $E1(b, Q)$ holds because it is an enabling condition of the $\mathit{Phase2a}(i, b, v, Q)$ action.

The three primed formulas of the "PROVE" clause of $\langle 1 \rangle 1$ are proved as steps $\langle 1 \rangle 5$, $\langle 1 \rangle 6$, and $\langle 1 \rangle 7$ below. The next three steps are used in their proofs.

$\langle 1 \rangle 2$. $\forall\, j : (\mathit{mbal2a}(j, b, Q) \neq -\infty) \Rightarrow \mathit{Done2a}(j,\ \mathit{mbal2a}(j, b, Q),\ \mathit{val2a}(j, b, Q))$

PROOF: Assume $mbal2a(j, b, Q) \neq -\infty$. By definition of $mbal2a$, this implies $val2a(j, b, Q)$ is a command (and not $\top$). Since $E1(b, Q)$ holds by assumption $\langle 1 \rangle 1.4$, the definitions of $mbal2a$ and $val2a$ imply that some acceptor a in Q has sent a $\langle \text{"1b"}, a, b, \langle mbal2a(j, b, Q), val2a(j, b, Q) \rangle \rangle_j$ message, which implies $vote_j[a][mbal2a(j, b, Q)] = val2a(j, b, Q)$ when the message was sent. Lemma 1.3 then implies $Done2a(j, mbal2a(j, b, Q), val2a(j, b, Q))$ was true when the message was sent, and is still true because $Done2a(\ldots)$ is stable.

$\langle 1 \rangle 3$. $\forall j,\ c < b,\ w : (c \leq mbal2a(j, b, Q)) \land (w \neq val2a(j, b, Q)) \Rightarrow NotChoosable(j, c, w)$

PROOF: We assume $c \leq mbal2a(j, b, Q)$ and $w \neq val2a(j, b, Q)$ and we prove $NotChoosable(j, c, w)$. Since $-\infty < c \leq mbal2a(j, b, Q)$, step $\langle 1 \rangle 2$ implies $Done2a(j, mbal2a(j, b, Q), val2a(j, b, Q))$. By assumption $\langle 1 \rangle 1.1$, this implies $SafeAt(j, mbal2a(j, b, Q), val2a(j, b, Q))$. The assumption $c \leq mbal2a(j, b, Q)$, together with assumption $\langle 1 \rangle 1.4$ and Lemma 1.8 (which imply $mbal2a(j, b, Q) < b$), implies $c < b$. The assumption $w \neq val2a(j, b, Q)$ and $SafeAt(j, mbal2a(j, b, Q), val2a(j, b, Q))$ then imply $NotChoosable(j, c, w)$.

$\langle 1 \rangle 4$. $\forall j,\ c < b,\ w : (sval2a(j, b, Q) = \top) \Rightarrow NotChoosable(j, c, w)$

PROOF: We assume $c < b$ and $sval2a(j, b, Q) = \top$ and prove $NotChoosable(j, c, w)$. We split the proof into two cases.

$\langle 2 \rangle 1$. CASE: $mbal2a(j, b, Q) = -\infty$

PROOF: The case assumption implies $mbal2a(j, b, Q) < c$, so assumption $\langle 1 \rangle 1.4$ and Lemma 3 imply $NotChoosable(j, c, w)$.

$\langle 2 \rangle 2$. CASE: $mbal2a(j, b, Q) \neq -\infty$

PROOF: Since $c < b$, we can split the proof into the following three cases.

$\langle 3 \rangle 1$. CASE: $mbal2a(j, b, Q) < c < b$

PROOF: By assumption $\langle 1 \rangle 1.4$, the case assumption and Lemma 3 imply $NotChoosable(j, c, w)$.

$\langle 3 \rangle 2$. CASE: $c \leq mbal2a(j, b, Q)$ and $w \neq val2a(j, b, Q)$

PROOF: By $\langle 1 \rangle 3$.

$\langle 3 \rangle 3$. CASE: $c \leq mbal2a(j, b, Q)$ and $w = val2a(j, b, Q)$

$\langle 4 \rangle 1$. $val2a(j, b, Q) \in StopCmd$ and we can choose $k > j$ such that $mbal2a(k, b, Q) \geq mbal2a(j, b, Q)$.

PROOF: We deduce that $val2a(j, b, Q) \in StopCmd$ and such a k exists by the $\langle 2 \rangle 2$ case assumption, the assumption $sval2a(j, b, Q) = \top$, and the definition of $sval2a$.

$\langle 4 \rangle 2$. $Done2a(k, mbal2a(k, b, Q), val2a(k, b, Q))$

PROOF: The $\langle 3 \rangle 3$ case assumption and $\langle 4 \rangle 1$ imply $mbal2a(k, b, Q) \neq -\infty$. Step $\langle 1 \rangle 2$ then proves $\langle 4 \rangle 2$.

$\langle 4 \rangle 3$. $NotChoosable(j, c, w)$

PROOF: Assumption $\langle 1 \rangle 1.1$ (with $j \leftarrow k,\ c \leftarrow mbal2a(k, b, Q)$, and $w \leftarrow val2a(k, b, Q)$) and $\langle 4 \rangle 2$ imply $NoReconfigBefore(k, mbal2a(k, b, Q))$. Step $\langle 4 \rangle 1$ asserts $j < k$; case assumption $\langle 3 \rangle 3$ and $\langle 4 \rangle 1$ imply

$c \leq mbal2a(k, b, Q)$; and $\langle 4\rangle 1$ and case assumption $\langle 3\rangle 3$ imply $w \in StopCmd$. Therefore, $NoReconfigBefore(k, mbal2a(k, b, Q))$ implies $NotChoosable(j, c, w)$.

$\langle 1\rangle 5$. $SafeAt(i, b, v)'$

PROOF: We assume $c < b$ and $w \neq v$ and prove $NotChoosable(i, c, w)'$. By Lemma 1.7, it suffices to prove $NotChoosable(i, c, w)$. We split the proof into two cases.

$\langle 2\rangle 1$. CASE: $sval2a(i, b, Q) = \top$

PROOF: $\langle 1\rangle 4$ (substituting $j \leftarrow i$) implies $NotChoosable(i, c, w)$.

$\langle 2\rangle 2$. CASE: $sval2a(i, b, Q) \neq \top$

PROOF: Since $c < b$, we can break the proof into two sub-cases.

$\langle 3\rangle 1$. CASE: $mbal2a(i, b, Q) < c < b$

PROOF: Assumption $\langle 1\rangle 1.4$ and Lemma 3 imply $NotChoosable(i, c, w)$

$\langle 3\rangle 2$. CASE: $c \leq mbal2a(i, b, Q)$

PROOF: Assumption $\langle 1\rangle 1.3$ implies $E3(i, b, Q, v)$. Case assumption $\langle 2\rangle 2$ and $E3(i, b, Q, v)$ imply $v = sval2a(i, b, Q)$. Case assumption $\langle 2\rangle 2$ and the definition of $sval2a$ then imply $v = val2a(i, b, Q)$. Case assumption $\langle 3\rangle 2$, the assumption $w \neq v$, and step $\langle 1\rangle 3$ (substituting $j \leftarrow i$) then imply $NotChoosable(i, c, w)$.

$\langle 1\rangle 6$. $NoReconfigBefore(i, b)'$

PROOF: We assume $j < i$, $w \in StopCmd$, and $c \leq b$ and we prove $NotChoosable(j, c, w)'$. By Lemma 1.7, it suffices to prove $NotChoosable(j, c, w)$. Since $c \leq b$, we need consider only the following two cases.

$\langle 2\rangle 1$. CASE: $b = c$

PROOF: Assumption $\langle 1\rangle 1.3$ implies $Done2a(i, b, v)'$. Since $i > j$ and $w \in StopCmd$, this implies the third disjunct of $NotChoosable(j, b, w)'$ (substituting i and v for the existentially quantified variables), which by the case assumption proves $NotChoosable(j, c, w)'$.

$\langle 2\rangle 2$. CASE: $c < b$

PROOF: We consider two sub-cases.

$\langle 3\rangle 1$. CASE: $sval2a(j, b, Q) = \top$

PROOF: $\langle 1\rangle 4$ and case assumption $\langle 2\rangle 2$ imply $NotChoosable(j, c, w)$.

$\langle 3\rangle 2$. CASE: $sval2a(j, b, Q) \neq \top$

PROOF: By case assumption $\langle 2\rangle 2$, we have the following two sub-cases.

$\langle 4\rangle 1$. CASE: $mval2a(j, b, Q) < c < b$

PROOF: Assumption $\langle 1\rangle 1.4$, the case assumption, and Lemma 3 imply $NotChoosable(j, c, w)$.

$\langle 4\rangle 2$. CASE: $c \leq mval2a(j, b, Q)$

PROOF: Assumption $\langle 1\rangle 1.3$ implies $E6(i, b, Q)$. The $\langle 3\rangle 2$ case assumption, the assumption $j < i$, and $E6(i, b, Q)$ imply $sval2a(j, b, Q) \notin StopCmd$. The assumption $w \in StopCmd$ then implies $w \neq sval2a(j, b, Q)$. By the $\langle 3\rangle 2$ case assumption and the definition of $\mathit{sval2a}$, we then have $w \neq \mathit{val2a}(j, b, Q)$. The $\langle 4 \rangle 2$ case assumption (which implies $\mathit{mval2a}(j, b, Q) \neq -\infty$) and $\langle 1 \rangle 3$ then imply $\mathit{NotChoosable}(j, c, w)$.

$\langle 1 \rangle 7$. $\mathit{NoneChoosableAfter}(i, b, v)'$

PROOF: We assume $v \in \mathit{StopCmd}$, $j > i$, $c < b$, and w any command and we prove $\mathit{NotChoosable}(j, c, w)'$. By Lemma 1.7, it suffices to prove $\mathit{NotChoosable}(j, c, w)$. We split the proof into two cases.

$\langle 2 \rangle 1$. CASE: $\mathit{sval2a}(i, b, Q) = \top$

PROOF: Assumption $\langle 1 \rangle 1.3$ implies $E4(i, b, Q, v)$, so the assumption $v \in \mathit{StopCmd}$ implies $E4b(i, b, Q, v)$. The case assumption, the assumption $j > i$, and $E4b(i, b, Q, v)$ imply $\mathit{sval2a}(j, b, Q) = \top$. The assumption $c < b$ and step $\langle 1 \rangle 4$ then imply $\mathit{NotChoosable}(j, c, w)$.

$\langle 2 \rangle 2$. CASE: $\mathit{sval2a}(i, b, Q) \neq \top$

$\langle 3 \rangle 1$. $\mathit{sval2a}(i, b, Q) = \mathit{val2a}(i, b, Q) = v$

PROOF: Assumption $\langle 1 \rangle 1.3$ implies $E3(i, b, Q, v)$, which implies $\mathit{sval2a}(i, b, Q) = v$. The case assumption and the definition of $\mathit{sval2a}$ then implies $\mathit{val2a}(i, b, Q) = v$.

$\langle 3 \rangle 2$. $\mathit{Done2a}(i, \mathit{mbal2a}(i, b, Q), v)$

PROOF: $\langle 3 \rangle 1$, assumption $\langle 1 \rangle 1.4$, and the definition of $\mathit{val2a}$ imply $\mathit{vote}_i[a][\mathit{mbal2a}(i, b, Q)] = v$ for some acceptor a in Q, which by Lemma 1.3 implies $\mathit{Done2a}(i, \mathit{mbal2a}(i, b, Q), v)$.

By the assumption $c < b$, it suffices to consider the following two cases.

$\langle 3 \rangle 3$. CASE: $c < \mathit{mbal2a}(i, b, Q)$

PROOF: Step $\langle 3 \rangle 2$ and assumption $\langle 1 \rangle 1.1$ imply $\mathit{NoneChoosableAfter}(i, \mathit{mbal2a}(i, b, Q), v)$. By the case assumption and the assumptions $v \in \mathit{StopCmd}$ and $j > i$, this implies $\mathit{NotChoosable}(j, c, w)$.

$\langle 3 \rangle 4$. CASE: $\mathit{mbal2a}(i, b, Q) \leq c < b$

$\langle 4 \rangle 1$. $\mathit{mbal2a}(j, b, Q) < \mathit{mbal2a}(i, b, Q)$

PROOF: The assumption $v \in \mathit{StopCmd}$ and $\langle 3 \rangle 1$ imply $\mathit{sval2a}(i, b, Q) \in \mathit{StopCmd}$. Case assumption $\langle 2 \rangle 2$ and the definition of $\mathit{sval2a}$ then imply $\mathit{mbal2a}(k, b, Q) < \mathit{mbal2a}(i, b, Q)$ for all $k > i$.

$\langle 4 \rangle 2$. $\mathit{NotChoosable}(j, c, w)$

PROOF: $\langle 4 \rangle 1$ and case assumption $\langle 3 \rangle 4$ imply $\mathit{mbal2a}(j, b, Q) < c < b$. By assumption $\langle 1 \rangle 1.4$, Lemma 3 implies $\mathit{NotChoosable}(j, c, w)$. $\square$

Theorem 1 $\Box\,\mathit{Consistency}$

PROOF: By definition of $\mathit{Consistency}$, it suffices to assume $\mathit{Chosen}(i, b, v)$ and $\mathit{Chosen}(i, c, w)$ and to prove $v = w$. Without loss of generality, we can assume $b \leq c$. We then have two cases.

1. CASE: $b < c$

PROOF: We assume $v \neq w$ and obtain a contradiction. Lemma 1.1 and $\mathit{Chosen}(i, c, w)$ imply $\mathit{Done2a}(i, c, w)$. By Lemma 4, this implies

$SafeAt(i, c, w)$. The assumptions $b < c$, an $v \neq w$ then imply $NotChoosable(i, b, v)$. By Lemma 2, this contradicts the assumption $Chosen(i, b, v)$.

2. CASE: $b = c$

   PROOF: Lemma 1.1 implies $Done2a(i, b, v) \land Done2a(i, c, w)$, which by Lemma 1.2 implies $b = c$. □

Theorem 2 $\Box$ *Stopping*

PROOF: By definition of *Stopping*, it suffices to assume $Chosen(i, b, v)$, $Chosen(j, c, w)$, $v \in StopCmd$, and $j > i$ and to obtain a contradiction. We split the proof into two cases.

1. CASE: $c < b$

   PROOF: $Chosen(i, b, v)$ and Lemma 1.1 imply $Done2a(i, b, v)$. This and Lemma 4 imply $NoneChoosableAfter(i, b, v)$, which by the case assumption and the assumptions $v \in StopCmd$ and $j > i$ implies $NotChoosable(j, c, w)$. The assumption $Chosen(j, c, w)$ and Lemma 2 then provide the required contradiction.

2. CASE: $c \geq b$

   PROOF: $Chosen(j, c, w)$ and Lemma 1.1 imply $Done2a(j, c, w)$. Lemma 4 then implies $NoReconfigBefore(j, c)$. The case assumption, the assumptions $v \in StopCmd$ and $j > i$, and $NoReconfigBefore(j, c)$ imply $NotChoosable(i, b, v)$. The assumption $Chosen(i, b, v)$ and Lemma 2 then provide the required contradiction. □

A.2 The Proof of Progress.

Theorem 3 $\forall\, b, Q : Progress(b, Q)$

PROOF: We assume $P1(b, Q)$, $P2(b, Q)$ and $P3(b)$ and we must prove that there exists a v such that either $\Diamond Chosen(i, b, v)$ or $(v \in StopCmd) \land \Diamond Chosen(j, b, v)$, for some $j < i$.

$\langle 1 \rangle 1.$ $\Diamond\Box E1(b, Q)$

PROOF: $P1(b, Q)$ implies that the ballot b leader eventually executes a $Phase1a(b)$ action. By $P2(b, Q)$, every acceptor a in Q eventually receives the $Phase1a$ messages. Because $bal[a]$ is set to a value c only by receiving a ballot c message, assumption $P3(b)$ implies $bal[a] \leq b$. Hence, a must eventually receive the $Phase1a$ message and execute $Phase1b(a, b)$. By $P2(b, Q)$, the $Phase1b$ message it sends is eventually received by the leader.

$\langle 1 \rangle 2.$ $\forall\, i, w : \Box(Done2a(i, b, w) \Rightarrow \Diamond Chosen(i, b, w))$

PROOF: $Done2a(i, b, w)$ means that a $Phase2a(i, b, w)$ action has been executed sending a $\langle \text{“2a”}, b, w \rangle_i$ message to every acceptor a. If a is in Q, then assumption $P2(b, Q)$ implies that it eventually receives that message. Assumption $P3(b)$ implies $bal[a] \leq b$, so $P1(b, Q)$ implies that every a in Q eventually executes $Phase2b(i, a, b, w)$, setting $vote_i[a][b]$ to w. Hence, eventually $Chosen(i, b, w)$ becomes true.

Since $\neg\Diamond F$ is equivalent to $\Box\neg F$, for any formula F, we can split the proof into the following two cases.

$\langle 1\rangle 3$. Case: $\exists\, k > i,\ w : \Diamond Done2a(k, b, w)$

$\langle 2\rangle 1$. $\Box\, E5(i, b)$

Proof: By definition of $E5(i, b)$, it suffices to assume $j < i$, $v \in StopCmd$, and $\Diamond Done2a(j, b, v)$ and obtain a contradiction. By the $\langle 1\rangle 3$ case assumption, we have $\Diamond Done2a(k, b, w)$ for $k > i > j$. Since $k \neq j$, the following two cases are exhaustive.

$\langle 3\rangle 1$. Case: $Phase2a(k, b, w)$ is executed after $Phase2a(j, b, v)$

Proof: This is impossible because the enabling condition $E5(k, b)$ of $Phase2a(k, b, w)$ implies $\neg Done2a(j, b, v)$.

$\langle 3\rangle 2$. Case: $Phase2a(j, b, v)$ is executed after $Phase2a(k, b, w)$

Proof: This is impossible because $E4a(j, b, v)$, which by the assumption $v \in StopCmd$ is an enabling condition of $Phase2a(j, b, v)$, implies $\neg Done2a(k, b, w)$.

$\langle 2\rangle 2$. Pick a quorum U such that $\Diamond\Box(E1(b, U) \wedge E6(i, b, U))$

Proof: Case assumption $\langle 1\rangle 3$ implies that we can choose U such that $\Diamond(E1(b, U) \wedge E6(k, b, U))$. By definition of $sval2a$, we have $E1(b, U)$ implies $E6(k, b, U)$ is stable. Since $E1(b, U)$ is obviously stable, $\Diamond(E1(b, U) \wedge E6(k, b, U))$ implies $\Diamond\Box\,(E1(b, U) \wedge E6(k, b, U))$. The assumption $k > i$ and the definition of $E6$ imply $\Box\,(E6(k, b, U) \Rightarrow E6(i, b, U))$.

$\langle 2\rangle 3$. Pick $w \notin StopCmd$ such that $\Diamond\Box E3(i, b, U, w)$

Proof: By $\langle 2\rangle 2$, we can choose a point in the execution at which $\Box(E1(b, U) \wedge E6(k, b, U))$ holds. By $\Box E1(b, U)$, the value of $sval2a(i, b, U)$ remains constant from that point on. If $sval2a(i, b, U) = \top$, let w be any command not in $StopCmd$. Otherwise, let $w = val2a(i, b, U)$, which by $E6(k, b, U)$ and the assumption $k > i$ is not in $StopCmd$.

$\langle 2\rangle 4$. $\Diamond Chosen(i, b, w)$

Proof: The theorem is proved if $\Diamond Chosen(i, b, u, V)$ for any u and V. Hence, by $\langle 1\rangle 2$ it suffices to assume $\Box\,\forall u : \neg Done2a(i, b, u)$, which is $\Box\, E2(i, b)$. We have proved $\Box\, E5(i, b)$ ($\langle 2\rangle 1$), $\Diamond\Box\, E3(i, b, U, w)$ ($\langle 2\rangle 3$), and $\Diamond\Box(E1(b, U) \wedge E6(i, b, U))$ ($\langle 2\rangle 2$). By $\langle 2\rangle 3$ ($w \notin StopCmd$), $E4(i, b, U, w)$ holds trivially. Hence, the $Phase2a(i, b, w, U)$ action is eventually always enabled, so by assumption $P1(b, Q)$ it is eventually executed by the ballot b leader. Step $\langle 1\rangle 2$ then implies $\Diamond Chosen(i, b, w)$.

$\langle 1\rangle 4$. Case: $\forall\, k > i,\ w : \Box\neg Done2a(k, b, w)$

$\langle 2\rangle 1$. $\exists\, j, v : \Diamond Done2a(j, b, v)$

Proof: We assume $\forall\, j, v : \Box\neg Done2a(j, b, v)$ and prove that eventually a $Phase2a(1, b, v)$ step occurs for some v. (Recall that 1 is the lowest instance number.)

$\langle 3\rangle 1$. $\Box\, E2(1, b)$

Proof: By the assumption $\forall\, j, v \,:\, \Box\neg \mathit{Done2a}(j, b, v)$.

$\langle 3\rangle 2.$ $\Box\,(E5(1, b) \wedge E6(1, b, Q))$

Proof: Conditions $E5$ and $E6$ are vacuously true for instance 1.

$\langle 3\rangle 3.$ $\exists\, v \,:\, \Diamond\Box\,(E3(1, b, Q, v) \wedge E4(1, b, Q, v))$

Proof: $\langle 1\rangle 1$ implies either (a) $\Diamond\Box(\mathit{sval2a}(1, b, Q) = \top)$ or (b) $\Diamond\Box(\mathit{sval2a}(1, b, Q) = v)$ for some command v. In case (a), $E3(1, b, Q, v)$ and $E4(1, b, Q, v)$ are trivially satisfied for any command v not in $\mathit{StopCmd}$. In case (b), let $v = \mathit{sval2a}(1, b, Q)$, so $E3(1, b, Q, v)$ is satisfied. If $v \notin \mathit{StopCmd}$, then $E4(1, b, Q, v)$ is trivially satisfied. If $v \in \mathit{StopCmd}$, then $E4a(i, b, v)$ is satisfied by the assumption $\forall\, j, v \,:\, \Box\neg \mathit{Done2a}(j, b, v)$ and $E4b(i, b, v, Q)$ is trivially satisfied.

$\langle 3\rangle 4.$ $\exists\, v \,:\, \Diamond \mathit{Done2a}(1, b, v)$

Proof: $\langle 1\rangle 1$, $\langle 3\rangle 1$, $\langle 3\rangle 2$, and $\langle 3\rangle 3$ imply that the $\mathit{Phase2a}(1, b, v, Q)$ action is eventually always enabled. By $P1(b, Q)$, this action must eventually be executed.

$\langle 2\rangle 2.$ It suffices to:

Assume: 1. h an instance number, v_h a command not in $\mathit{StopCmd}$, and $\Diamond \mathit{Done2a}(h, b, v_h)$

2. $\forall\, j > h,\ v \,:\, \Box\neg \mathit{Done2a}(j, b, v)$

Prove: $\exists v \,:\, \Diamond \mathit{Done2a}(h + 1, b, v)$

Proof: $\langle 2\rangle 1$ and case assumption $\langle 1\rangle 4$ implies that there is a largest instance number h and a command v_h such that $\Diamond \mathit{Done2a}(h, b, v_h)$, and that $h \leq i$. If $v_h \in \mathit{StopCmd}$, then $\langle 1\rangle 2$ implies $\Diamond \mathit{Chosen}(h, b, v_h)$, and $h \leq i$ then implies we are done. Therefore, it suffices to assume $v_h \notin \mathit{StopCmd}$ and obtain a contradiction, which we do by proving that the assumptions imply the Prove clause.

$\langle 2\rangle 3.$ $\Diamond\Box\, E5(h + 1, b)$

Proof: Assumption $\langle 2\rangle 2.1$ asserts $\Diamond \mathit{Done2a}(h, b, v_h)$, which implies $\Diamond E5(h, b)$. Since $\mathit{Done2a}(h, b, v_h)$ implies $\forall\, j < h,\ w \in \mathit{StopCmd} \,:\, \neg E4a(j, b, w)$, it implies that $\mathit{Phase2a}(j, b, w, U)$ is not enabled for any $j < i$, $w \in \mathit{StopCmd}$, and quorum U, which implies that $E5(h, b)$ is stable, proving $\Diamond\Box\, E5(h, b)$. Assumption $\langle 2\rangle 2.1$ and Lemma 1.2 imply $\forall v \in \mathit{StopCmd} \,:\, \Box\neg \mathit{Done2a}(h, b, v)$, which together with $\Diamond\Box\, E5(h, b)$ implies $\Diamond\Box\, E5(h+1, b)$.

$\langle 2\rangle 4.$ Choose a quorum U such that $\mathit{Phase2a}(h, b, v_h, U)$ is eventually executed.

Proof: U exists by assumption $\langle 2\rangle 2.1$.

$\langle 2\rangle 5.$ $\Diamond\Box\, E1(b, U)$

Proof: By $\langle 2\rangle 4$ and the stability of $E1(b, U)$.

$\langle 2\rangle 6.$ $\Diamond\Box\, E6(h + 1, b, U)$

Proof: $\langle 2\rangle 4$, $\langle 2\rangle 5$, and the enabling condition $E6(h, b, U)$ imply

$(*)$ $\forall\, j < h \,:\, \Diamond\Box((\mathit{sval2a}(j, b, U) \neq \top) \Rightarrow (\mathit{sval2a}(j, b, U) \notin \mathit{StopCmd}))$

Step $\langle 2\rangle 4$, assumption $\langle 2\rangle 2.1$ (which implies $v_h \notin \mathit{StopCmd}$), and enabling condition $E3(h, b, U, v_h)$ imply $\Diamond\Box(sval2a(h, b, U) \notin \mathit{StopCmd})$. This and $(*)$ imply $\Diamond\Box\, E6(h+1, b, U)$.

⟨2⟩7. $\exists\, v : \Diamond\Box(E3(h+1, b, U, v) \land E4(h+1, b, U, v))$

PROOF: ⟨2⟩5 implies that it suffices to consider the following two cases.

⟨3⟩1. CASE: $\Diamond\Box(sval2a(h+1, b, U) = v)$ for some command v.

PROOF: The case assumption implies $\Diamond\Box\, E3(h+1, b, U, v)$. Assumption ⟨2⟩2.2 implies $\Box\, E4a(h, b, v)$, and the case assumption trivially implies $\Diamond\Box\, E4b(h, b, U, v)$.

⟨3⟩2. CASE: $\Diamond\Box\,(sval2a(h+1, b, U) = \top)$

PROOF: The case assumption implies $\Diamond\Box(E3(h+1, b, U, v) \land E4(h+1, b, U, v))$ for any command v not in $\mathit{StopCmd}$.

⟨2⟩8. $\Box\, E2(h+1, b)$

PROOF: Assumption ⟨2⟩2.2.

⟨2⟩9. $\exists\, v : \Diamond \mathit{Done}2a(h+1, b, v)$

PROOF: ⟨2⟩3, ⟨2⟩5, ⟨2⟩6, ⟨2⟩7, and ⟨2⟩8 show that the $\mathit{Phase}2a(h+1, b, v, U)$ action is eventually always enabled for some v. Assumption $P1(b, Q)$ implies that the ballot b leader eventually executes this action. By ⟨2⟩2, this completes the proof of ⟨1⟩4. □