A Proof of Correctness

1 Problem Definition

SDPaxos guarantees the following safety properties, which are similar to those provided by the other Paxos-based protocols:

Nontriviality Any command executed in any slot of the global log must have been proposed by a client.

Consistency At most one command can be executed by different replicas in the same slot of the global log.

Linearizability If two commands α and β are operations on the same object, and β is proposed by a client after the request for α is responded to by any replica (both are eventually responded to), then α will be executed before β by every replica.

Now we prove these properties for SDPaxos.

2 Proof

First, we prove that C-instances satisfy the following properties.

Lemma 2.1 *Any command learned in any C-instance must have been proposed by a client, and at most one command can be learned in the same C-instance.*

Proof. C-instances run classic Paxos algorithm. Paxos satisfies nontriviality and consistency, which has been proved.

Theorem 1 (Nontriviality) *Any command executed in any position of the global log must have been proposed by a client.*

Proof. Any command in any position of the global log must have learned in some C-instance. By Lemma 1, C-instances only learn proposed commands.

For consistency and linearizability, there are two cases: one is 3 or more than 5 replicas, the other is 5 replicas. We will discuss the two case respectively. Note that in this paper, we say a command is "committed" when it is "ready" in the SDPaxos paper, for simplicity.

Lemma 2.2 *In SDPaxos for 3 or more than 5 replicas, at most one replica can be learned in an O-instance.*

Proof. In SDPaxos for 5 replicas, O-instances also run classic Paxos algorithm. Similarly to the proof of Lemma 1.1, Paxos ensures consistency.

Theorem 2 (Consistency for 3 or more than 5 replicas) *At most one command can be executed by different replicas in the same position of the global log.*

Proof.

1. SUFFICES ASSUME: i is a positive integer and any O-instance O_j with $j \leq i$ has been learned. The value chosen in O_i is n.

   PROVE: There can be at most one command in the i^{th} slot of the global log.

   PROOF: Since a command in the ith slot of the global log can be executed only after every O-instance O_j with $j \leq i$ is learned, it suffices to show that there can be at most one command in the ith slot of the global log after all preceding O-instances have been learned.

2. The number of times that n is chosen in all O-instances O_j with $j \leq i$ is a determined value.

   PROOF: By Lemma 1.2, there will be a determined value chosen in each O-instance after learned. Therefore, the number of times that n is chosen is also determined.

3. ASSUME: O_i is the k^{th} O-instance in which n is chosen.
   PROVE: At most one command can be chosen in C-instance C_{nk}.

   PROOF: By Lemma 1.1.

4. Q.E.D.

   PROOF: By 1, 2, and 3.

Theorem 3 (Linearizability for 3 or more than 5 replicas) *If α and β are commands on the same object, and β is requested by client after α is committed (both are eventually committed), then α will be executed before β.*

Proof.

1. ASSUME: α is proposed in C-instance C_{ni}.
   PROVE: At the moment when α is committed, there must be at least i O-instances for R_n in the sequencer's assignment log.

   PROOF: We prove this contradiction. Assume there are fewer than i O-instances for R_n in the sequencer's assignment log when α is committed. The only possible case is, the sequencer failed is reelected, and there are fewer than i O-instances for R_n seen by a majority by that time. However, since the command in C_{ni} has been committed, there must be i O-instances for R_n seen by a majority when the command is committed. Then the O-instances for R_n that the sequencer did not see in the view change must be proposed later by the new sequencer (because they will be rejected with a stale view number). Because there is a contradiction, the assumption that there are fewer than i O-instances for R_n in the sequencer's assignment log must be false.

2. At the moment when the request for β is received, there must be at least i O-instances for R_n in the sequencer's assignment log.

   2.1. CASE: The sequencer has never failed after α is committed.

   PROOF: By 1, there must be at least i O-instances for R_n in the sequencer's assignment log. Because the O-instance counter increases monotonically, there must be at least i O-instances for R_n in the sequencer's assignment log when β is received.

   2.2. CASE: The sequencer has ever failed and been reelected after α is committed.

   PROOF: Because α has been committed, there must be at least i O-instances for R_n that have been accepted by a majority. Whenever a sequencer is elected in a view change, it can see at least i O-instances for R_n in its assignment log.

   2.3. Q.E.D.

   PROOF: Cases 2.1 and 2.2 are exhaustive.

3. Q.E.D.

   By 1, 2, and the fact that a sequencer will never assign a command to a slot preceding a non-empty one in its assignment log.

Now we discuss the case of 5 replicas, where O-instances are not learned by Paxos. We say an O-instance is learned in the k^{th} view, if the learner learns this instance on receiving an *O-ACK* with a view number of k. For simplicity, we assume there is a sequencer elected with every view number, because there will be no O-instance learned in a view without a elected sequencer.

Theorem 4 (Consistency for 5 replicas) *At most one command can be executed by different replicas in the same position of the global log.*

Proof.

1. SUFFICES ASSUME: A replica R executes a command α proposed in C-instance C_{ni} in the j^{th} global slot. It learns O-instance O_j in the v^{th} view.
   PROVE: Every replica can only learn n in O_j. And for every replica, after all preceding O-instances are learned, O_j will be the i^{th} O-instance for R_n.

   PROOF: Since C-instance executes classic Paxos algorithm, every replica can only learn α in C_{ni}. If a replica learns α in C_{ni}, and learn n in O_j as the i^{th} O-instance for R_n, it will also execute α in the j^{th} global slot.

2. If the sequencer of the v^{th} view never fails, then consistency always holds.

   PROOF: For those O-instances that are preceding O_j and learned by R in the v^{th} view, other replicas will learn the same result if the sequencer of the v^{th} view never fails. For those O-instances that are preceding O_j and learned by R in previous views, they have been accepted by a majority in the $(v-1)^{th}$ view change, so other replicas will also learn the same result. Because all O-instances preceding O_j must be learned by R in the v^{th} or some previous view, other replicas will learn the same result in all those O-instances.

3. Whenever a sequencer is elected in the $(v')^{th}$ view change ($v' > v$), the values in all O-instances O_k with $k \leq j$ in the new sequencer's assignment log must be the same as those in R's.

   3.1. CASE: The replica R is one of the majority voters.

   PROOF: The replica R and other voters will report all these O-instances in its vote. All these O-instances in these votes must have the same values: for those O-instances that are preceding O_j and learned by R in the v^{th} view, they are proposed by the same proposer; for those O-instances that are preceding O_j and learned by R in previous views, they have been accepted by a majority in the $(v-1)^{th}$ view change. Therefore, the new sequencer must see the same values in these O-instances as R.

   3.2. CASE: The replica R is not one of the majority voters.

   3.2.1. The values in all O-instances O_k with $k \leq j$ for the majority voters in the new sequencer's assignment log must be the same as those in R's.

   PROOF: Similarly to 3.1, for any one of the majority voters, the new sequencer will see all O-instances for it.

   3.2.2. The values in all O-instances O_k with $k \leq j$ for the sequencer of the v^{th} view in the new sequencer's assignment log must be the same as those in R's.

   PROOF: Those O-instances learned in the v^{th} view must have been accepted by a majority. While those O-instances learned in previous views must also have been accepted by a majority in the $(v-1)^{th}$ view change. So the new sequencer will see the same values in all O-instances for the sequencer of the v^{th} view.

   3.2.3. Q.E.D.

   PROOF: By 3.2.1 and 3.2.2, all O-instances O_k with $k \leq j$ for all replicas except one replica (say R_m) can be traced by the new sequencer. So the gaps in the new sequencer's assignment must be originally allocated to R_m. Because R has executed the command in C_{ni} and all preceding ones, which the corresponding C-instances must have been accepted by a majority, the new sequencer will make sure that there are at least the same number of O-instances for R_m as that in R's assignment log. Therefore, the values in all O-instances O_k with $k \leq j$ in the new sequencer's assignment log must be the same as those in R's.

3.3. Q.E.D.

PROOF: Cases 3.1 and 3.2 are exhaustive.

4. Q.E.D.

PROOF: By 1, 2 and 3.

Theorem 5 (Linearizability for 5 replicas) *If α and β are commands on the same object, and β is requested by client after α is committed (both are eventually committed), then α will be executed before β.*

Proof.

1. ASSUME: α is proposed in C-instance C_{ni}.
   PROVE: At the moment when α is committed, there must be at least i O-instances for R_n without preceding gap (empty ones between non-empty ones) in the sequencer's assignment log.

   PROOF: We prove this by contradiction. Assume there are fewer than i O-instances for R_n or with preceding gap in the sequencer's assignment log when α is committed. The only possible case is, the sequencer failed is reelected, R_n is not one of the majority voters, and C_{ni} has not been seen by a majority by that time. (If R_n is a voter, then the new sequencer will see all its O-instances; otherwise, the new sequencer will ensure there will be at least i O-instances for R_n with no preceding gap, according to the view change protocol.) However, since the command in C_{ni} has been committed, C_{ni} must has been seen by a majority when the command is committed. After the new sequencer is elected, C_{ni} will be rejected due to a stale view number. Then R_n will contact the new sequencer and synchronize all the O-instances, so it will change the value in its i^{th} O-instances for itself to *no-cl*. Here we get a contradiction, hence the assumption that there are fewer than i O-instances for R_n or with preceding gaps in the sequencer's assignment log must be false.

2. At the moment when the request for β is received, there must be at least i O-instances for R_n without preceding gap in the sequencer's assignment log.

   2.1. CASE: The sequencer has never failed after α is committed.

   PROOF: By 1, there must be at least i O-instances for R_n in the sequencer's assignment log. Because the O-instance counter increases monotonically, there must be at least i O-instances for R_n without preceding gap in the sequencer's assignment log when β is received.

   2.2. CASE: The sequencer has ever failed and been reelected after α is committed.

   2.2.1. CASE: R_n is one of the majority voters.

   PROOF: Since R_n has committed the command α, it must have seen at least i O-instances for itself with no preceding gap. Then the new sequencer will see all these O-instances in R_n's vote.

   2.2.2. CASE: R_n is the old sequencer.

   PROOF: Since R_n has committed the command α, then all preceding O-instances must have been seen by a majority. Then the new sequencer will see all these O-instances in the view change.

   2.2.3. CASE: R_n is neither a voter nor the old sequencer.

   PROOF: Since R_n has committed the command α, C_{ni} must have been seen by a majority. So the new sequencer will make sure there are at least i O-instances for R_n with no preceding gap, *i.e.,* by proposing n in the gaps.

   2.2.4. Q.E.D.

   PROOF: Cases 2.2.1, 2.2.2 and 2.2.3 are exhaustive.

2.3. Q.E.D.

PROOF: Cases 2.1 and 2.2 are exhaustive.

3. ASSUME: β is proposed in C-instance C_{mj}. The sequencer has ever failed and been reelected after it proposes m in an O-instance as the j^{th} one for R_m.

   PROVE: The new sequencer will not propose m in an O-instance preceding the i^{th} one for R_n.

3.1. CASE: R_n is one of the majority voters.

PROOF: Since R_n has committed the command α, it must have seen at least i O-instances for itself with no preceding gap. Then the new sequencer will see all these O-instances in R_n's vote. So it cannot propose m in an O-instance preceding the i^{th} one for R_n.

3.2. CASE: R_n is the old sequencer.

PROOF: Since R_n has committed the command α, then all preceding O-instances must have been seen by a majority. Then the new sequencer will see all these O-instances in the view change. So it cannot propose m in an O-instance preceding the i^{th} one for R_n.

3.3. CASE: R_n is neither a voter nor the old sequencer.

PROOF: Since R_n has committed the command α, C_{ni} must have been seen by a majority. So the new sequencer will make sure there are at least i O-instances for R_n with no preceding gap, *i.e.,* by proposing n in the gaps. So it cannot propose m in an O-instance preceding the i^{th} one for R_n.

3.4. Q.E.D.

PROOF: Cases 3.1, 3.2 and 3.3 are exhaustive.

4. Q.E.D.

   By 1, 2, and 3.

Finally, our system also satisfies the following property:

Liveness Any proposed command will eventually be executed by every non-faulty replica, given that at least a majority of replicas are non-faulty, clients and replicas keep resending messages and messages are eventually delivered to their destinations, and a new sequencer will eventually be elected after the old one's failure.

The assumption that sequencer elections always make progress can be realized by setting randomized election timeouts for different replicas, in which non-sequencer replicas will detect sequencer's failure sporadically. This design prevents an election from ending up with no winner, since different candidates may compete for votes.