

CS 425 / ECE 428

Distributed Systems

Fall 2026

Aishwarya Ganesan w/ Ram Kesavan

Lecture 19: Raft – questions w/ answers

Raft Review

- State machine safety
- Leader-driven replication
 - log consistency check
- Committing a command
- Electing a leader



Respond at

pe.app/[aishwaryaganesa](#)

The correct answer will be revealed when this question is presented.

Poll Everywhere

Client sends command X to a Raft cluster with 5 servers.
The leader appends X to its log.
Two followers acknowledge successfully. Remaining two followers are unreachable.
What can the leader do?



Anonymous 81

9%

Wait for all 5 servers to acknowledge before applying the entry to its state machine.



67%

Commit the entry by applying it to its state machine, and respond to the client.

22%

Commit the entry, but wait for all followers before responding to the client.

2%

Start a new leader election because two followers are unreachable.

Two followers + leader (3 nodes) appended X to the log. This means that a majority logged X. So, it is safe to commit X as long X is current-term entry.

+ Add choice

✓ Set correctness

☆ Q&A

[Manual flow](#)

✓ Choices

✓ Results

✓ Lock

✓ Correctness



Respond at

pe.app/[aishwaryaganesa](#)

The correct answer will be revealed when this question is presented.

In a 5-server Raft cluster, the current leader is in term 4 and has an uncommitted log entry from term 2. Assume all preceding entries have already been applied. When can the leader apply this term-2 entry to its state machine?



Anonymous 75

53%

Once the term-2 entry is replicated on a majority of servers.

8%

Once the term-2 entry is replicated on all 5 servers.



27%

Once leader replicates entry from term 4 on a majority

12%

Once the leader appends entry from term 4 to its log

Option A is incorrect because replication on a majority is insufficient to commit an entry from an earlier term. A future leader could still overwrite it.

Option C is correct because once a current-term entry is replicated on a majority, future term's leader will have that entry in the log (along with preceding entries).

+ Add choice

✓ Set correctness

🔖 Q&A

[Manual flow](#)

✓ Choices

✓ Results

✓ Lock

✓ Correctness

Question. Valid log states?

1.

1	2	3	4	5
1	1	3	2	2
a	b	c	d	e

Is this a valid log? – No!

Once an entry from term 3 appears, entries from term 2 cannot be appended afterwards.

This is because only leaders append new entries and they do so in their current term, and terms increase monotonically.

2. Can two servers have these logs? – No! Raft ensures at most one leader per term. The same leader will not be appending two different commands for the same index in the same term.

S1

1	1	2
a	b	c

S2

1	1	2
a	b	d

Question. Valid log states (contd)?

3. Can two servers have these logs?

S1

1	1	3
a	b	c

S2

1	1	2
a	b	c

Yes, leader for term 2, appended c to some nodes but crashed before replicating to majority.

Leader for term 3 got majority votes without the nodes that appended the term-2 entry and it then appended c

4. Can two servers have these logs?

S1

1	2	4
a	c	d

S2

1	3	4
a	b	d

No, if S1 and S2 accepted <d> from term-4 leader at index 3, their log prefix should have matched the leader's. But that is not the case here; so, the prefix check would have failed and rejected <d> until the log matched leader's

Question. Valid log states (contd)?

Consider following log states. S3's current term is 2. Can S3 become a leader?

S3 cannot become a leader in term 2 – No (case discussed in lecture)

- Assume S3 is a candidate with current term 2 running for election
- From entry 3,c at index 3 on S1, we know there was a leader for term 3.
- This implies that at least a majority would have voted for leader in term 3 and advanced to current term 3.
- This means that S3 cannot get a majority vote for term 2 (the check that *candidate term* < *server term* in election will fail on at least a majority)

S1

1	1	3
a	b	c

S2

1	1
a	b

S3

1	1
a	b

Now, can S3 become a leader in a different term, say 4 or higher? – Yes!

- S3 runs for election with current term 2
- S3 fails to get majority vote but when S3 sees a message carrying term 3, it will update its current term to 3.
- Suppose S3's election timer expires. It increments its term to 4, becomes a candidate, votes for itself, and requests votes from others.
- S1 rejects: S3's log is less up-to-date than S1's
- S2 grants its vote: S2 and S3 have identical logs, and S2 has not yet voted in term 4 and S2's term is not ahead.
- **Result:** S3 wins 2 out of 3 votes (S2's and S3's) and becomes leader in term 4.

Log Properties

- Log entries have terms that are **monotonically increasing**
- If entries on different servers have the same index and term:
 - they store the **same command**
 - the logs are **identical in all preceding entries**
- If a given entry is **committed**, all **preceding** entries are also committed.

Question: Which entries are safe to apply?

Current term is 4 and S1 is the leader. Which entries can S1 safely apply to its state machine?

Assume all nodes are in term 4.

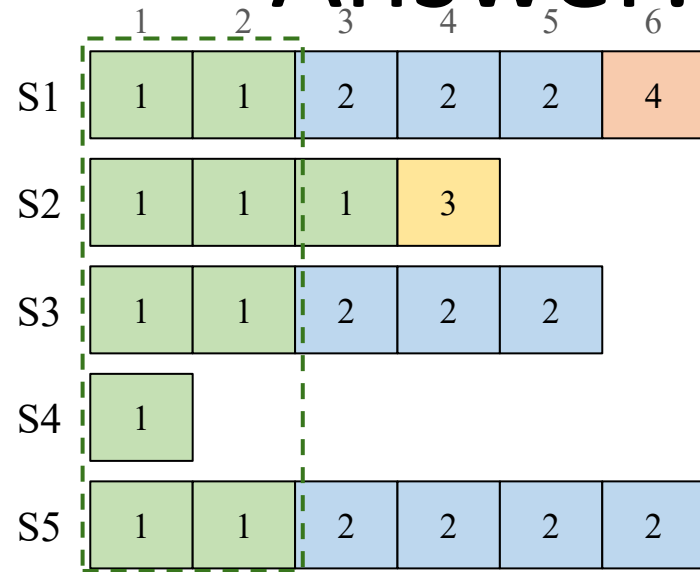
	1	2	3	4	5	6
S1	1	1	2	2	2	4
S2	1	1	1	3		
S3	1	1	2	2	2	
S4	1					
S5	1	1	2	2	2	2

Hint: think about which nodes could become leader if S1 fails right now.

For each of S1–S5: whose votes can it get?

Question: Which entries are safe to apply?

Answer: Only entries 1 and 2



If there is an election now, who can become leader for term 5?

S1: Yes — votes from self, S3, S4

S2: Yes — votes from self, S3, S4, S5 (last term 3 beats 2)

S3: No — only S4 and itself (S1, S2, S5 are more complete)

S4: No — only itself

S5: Yes — votes from self, S4, S3 (same term 2, longer log)

- Although

1	1	2	2	2
---	---	---	---	---

 present on a majority, only entries present on all potential leaders (S1, S2, S5) are

1	1
---	---

 and guaranteed to survive. For instance, If S2 wins, then

2	2	2
---	---	---

 will be overwritten

- However, in a real protocol run, one cannot determine which entries will survive in all future leaders (because some nodes may be down during leader election to make this).
- This is why a server denies the vote to a candidate whose log is lagging and the leader replicates at least one entry in the current term on majority before committing any previous term entries.
- Once S1 replicates entry 6 (term 4) on a majority, S2 can no longer win, and entries 3–5 become safe to be committed.

Restrictions on Commitment

For a leader to decide an entry is committed:

- It must be stored on a **majority** of servers
- At least one new entry from the leader's **current term** must also be stored on a **majority** of servers

Question: Deposed leader

In a 5-server Raft cluster, S1 is the leader in term 2. S1 becomes partitioned from the other three servers, which elect S3 as the new leader in term 3. S1 is still alive and believes it is the leader. S2 is in term 2. Can this violate safety?

Does not violate state-machine safety

S1 cannot commit new entries because majority are in term3 and majority will reject S1's RPCs

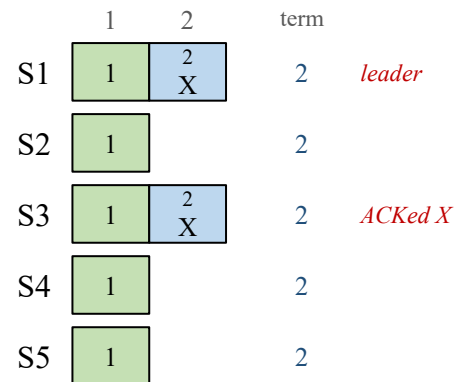
Question: Deposed leader (variant 2)

In a 5-server Raft cluster, S1 is the leader in term 2. S1 becomes partitioned from the other three servers, which elect S3 as the new leader in term 3. S1 is still alive and believes it is the leader. S2 is in term 2. After this, can S1 commit an old log entry that was received before the election started (by completing appendentries on a follower)?

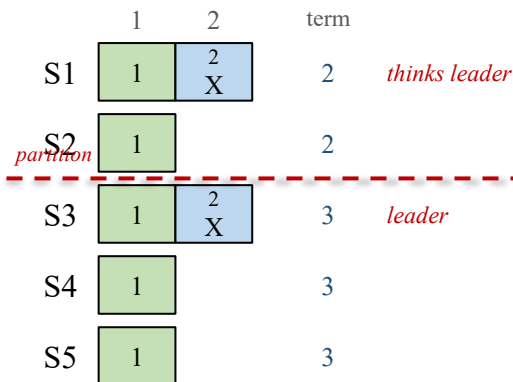
Deposed leader (variant 2) (contd)

Yes, but only if S1 had ack from one node in the other majority (S3—S5) before election

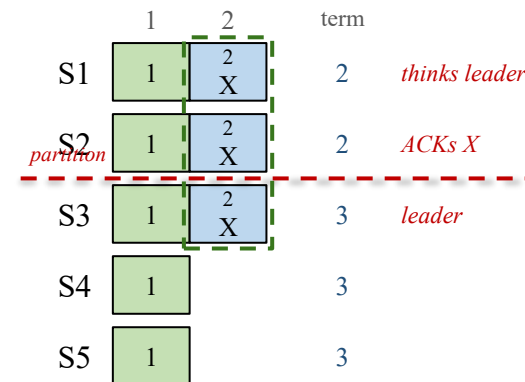
① Before election (term 2)



② Election (term 3)



③ After election



X committed by a deposed leader

- Before election:** S1 replicates X to S3 and receives an ACK.
- Election:** S3 is leader in term 3 with votes from S3, S4, S5.
- After election:** S1, continues to replicate X to S2 and receives an ACK. X from S1's current term is now stored on a majority (S1, S2, S3), so S1 commits X.

Why is this safe? The new leader S3 already contains X. Thus, committing X does not violate safety.

Deposed leader (variant 2) contd.

Note that Raft will not let S1 commit X after the election if none of S3-S5 had it previously! Why?

Consider:

① Before election (term 2)

	1	2	term	
S1	1	2 X	2	leader
S2	1		2	
S3	1		2	
S4	1		2	
S5	1		2	

② Election (term 3)

	1	2	term	
S1	1	2 X	2	thinks leader
S2	1		2	
S3	1		3	leader
S4	1		3	
S5	1		3	

③ After election

	1	2	term	
S1	1	2 X	2	thinks leader
S2	1	2 X	2	ACKs X
S3	1		3	leader
S4	1		3	X rejects
S5	1		3	X rejects

X on 2/5 — S1 cannot commit X

- Before election:** S1 had appended X only to its own log.
- Election:** S3 is leader in term 3 with votes from S3, S4, S5 – none of them contain X
- After election:** S1, continues to replicate X to S2 and receives an ACK. **but cannot replicate X to any of S3—S5** (they will all reject a leader of an old term).

X is stored only on S1 and S2 (2/5). So, S1 cannot commit X, preserving safety!

Question: Reads

Do reads have to go through consensus? Can you just read from leader's state machine?

- **No**, while reads do not change state and don't have to be replicated, just reading leader's state machine without going through a majority is **unsafe**.
 - Consider a Key-value store as the state machine and the example in slide 12
 - S1 still thinks it is the leader for term 2 while S3-S5 moved to term 3
 - If reads only execute on S1's state machine without going through majority, S1 never discovers there is a new leader
 - The new leader could use majority from S3-S5 to commit new entries that S1 don't know about
 - S1 could therefore serve a **stale read** from its state machine
- Does this mean a KV-store read will write to the log on a majority?
 - Not required! Raft handles this as a special case.
 - Leader contacts a majority to see if it still the leader and serve the read from state machine, avoiding log writes on a majority.

Question: Follower corrupt disk

Suppose a follower crashes, and when it restarts, its most recent vote has been lost. Is it safe for the follower to rejoin the cluster ?

- **No**, follower could vote **twice** in same term leading to **two leaders** obtaining majorities
- Consider 3 servers all with identical logs.
- S3 votes for S1 in term 2, then crashes and restarts, losing its persisted vote.
- S3 votes **again** for S2 in term 2. Both S1 and S2 now have majority (self + S3) and believe they are leaders in term 2.
- S1 commits X in term2 at index i on S1 and S3
- S2 then tries to commits a different entry Y in term2 at index i on S2 and S3 at nextindex i on S3
 - S3 overwrites new entry Y at i (entries up to i-1 match between s2 and s3).
 - Y is committed at i. Thus, the system commits two different entries for index i violating safety.