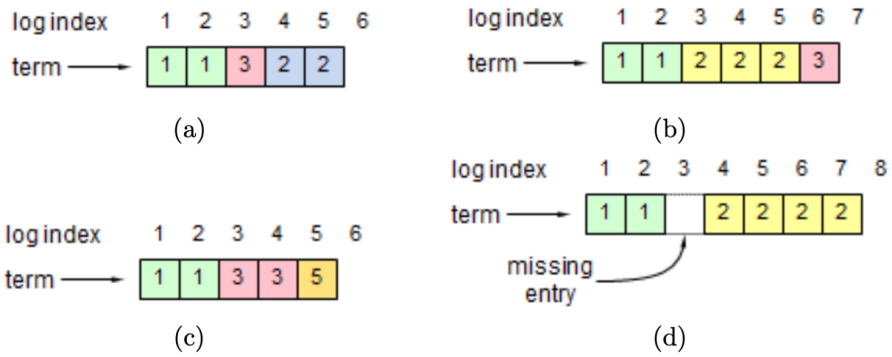
Question 1:

1. Each figure below shows a possible log configuration for a Raft server (the contents of log entries are not shown; just their indexes and terms). Considering each log in isolation, could that log configuration occur in a proper implementation of Raft? If the answer is "no," explain why not.



Answer:
Let's analyze each log configuration:

(a)
log index - 1 2 3 4 5 6
term - 1 1 3 2 2

This configuration is invalid according to Raft's rules. In Raft, the term of a log entry must always be monotonically increasing with the log index. Here, the term for index 3 is 3, while the term for index 4 is 2, violating the monotonically increasing term requirement.

(b)
log index - 1 2 3 4 5 6 7
term - 1 1 2 2 2 3 3

This configuration is valid according to Raft's rules. The terms are monotonically increasing with the log index.

(c)
log index - 1 2 3 4 5 6
term - 1 1 3 3 5

Similar to (a), this configuration is invalid because the term for index 3 is 3, while the term for index 4 is also 3, violating the monotonically increasing term requirement.

(d)
log index - 1 2 3 4 5 6 7 8
term - 1 1 (missing entry) 2 2 2 2

This configuration is also invalid. While the terms seem to be monotonically increasing where present, the missing entry at index 3 breaks the continuity of the log sequence, which is not allowed in a proper Raft implementation. All log entries must be present without any gaps in the sequence of log indexes.

So, the correct answer is:

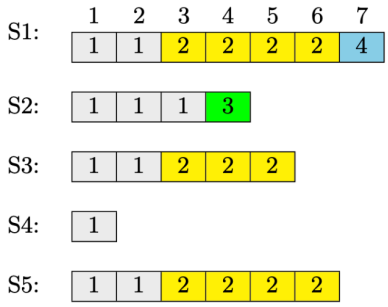
(a) No, because the term for index 4 is not monotonically increasing.
(c) No, because the term for index 4 is not monotonically increasing.
(d) No, because there is a missing entry at index 3.

Question 2:

2. The figure below shows the state of the logs in a cluster of 5 servers (the contents of log entries are not shown; just their indexes and terms) where the current leader is S1.

   (a) Which of the servers could be elected as the new leader if S1 were to fail?

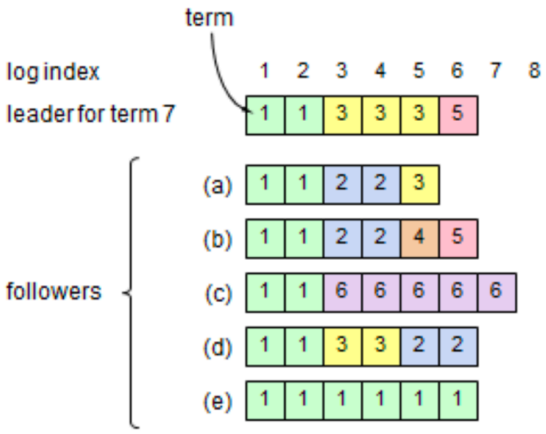
   (b) Which log entries may safely be applied to the state machines?



Answer:
(a) s5 because of max committed and max log entries.
(b) till index 5 because s1,s3,s5 have all same till 5.

Question 3:

3. Consider the figure below, which displays the logs in a cluster of 6 servers just after a new leader has just been elected for term 7 (the contents of log entries are not shown; just their indexes and terms). For each of the followers in the figure, could the given log configuration occur in a properly functioning Raft system? If yes, describe how this could happen; if no, explain why it could not happen.



Answer:
Certainly. Here's the complete analysis:

In the provided scenario, we have a new leader elected for term 7 with the following log:

Leader for term 7:
index - 1 2 3 4 5 6 7 8
term - 1 1 3 3 3 5

Now, let's analyze each follower's log configuration:

s1: 1 1 2 2 3
- This log configuration is valid. It's possible that s1 missed some log entries during network disruptions or other issues. Upon rejoining the cluster or reestablishing communication with the leader, s1 could catch up with the missing log entries through the Raft protocol's log replication mechanism.

s2: 1 1 2 2 4 5
- Similar to s1, this log configuration is valid. It's conceivable that s2 missed some log entries during network disruptions or other issues. The Raft protocol allows followers to catch up with missed entries during normal operation or when they regain connectivity with the leader.

s3: 1 1 6 6 6 6 6

- This log configuration is not valid in a properly functioning Raft system. While it is more up-to-date than the leader's log for terms 1, 2, 3, and 4, this situation contradicts the normal behavior of the Raft consensus algorithm. Followers should not have logs more up-to-date than the leader for a given term. This suggests a breakdown in the consistency and integrity of the Raft replication process.

s4: 1 1 3 3 2 2
- Similar to s3, this log configuration is invalid. There is a non-monotonic decrease in terms from index 3 to index 4, which violates the Raft protocol's requirement of monotonically increasing terms.

s5: 1 1 1 1 1 1
- This log configuration is valid. Although it seems s5 hasn't received any recent log updates, it hasn't violated any consistency rules. It's possible that s5 has not been able to catch up with the latest logs due to network issues or being in an isolated state, but its log consistency remains intact.

In summary:
- s1 and s2: Yes, these log configurations could occur in a properly functioning Raft system, and they could catch up with missing log entries.
- s3 and s4: No, these log configurations are not valid in a properly functioning Raft system due to inconsistencies in log terms.
- s5: Yes, this log configuration could occur, although it indicates that the server hasn't received recent updates.

Question 4:

4. Suppose that a hardware or software error corrupts the nextIndex value stored by the leader for a particular follower. Could this compromise the safety of the system? Explain your answer briefly.

Answer:
Yes, a corrupted nextIndex value stored by the leader for a follower could compromise the safety of the system, potentially leading to inconsistencies and failures in maintaining the replicated log.

In the Raft algorithm, the nextIndex value represents the index in the leader's log where the next log entry should be sent to a specific follower. If this value is corrupted due to a hardware or software error, it could lead to several issues:

1. Log Divergence: The corrupted nextIndex value might cause the leader to send incorrect log entries to the follower. This can result in the follower having a log that diverges from the leader's log, leading to inconsistencies across the cluster.

2. Data Loss or Corruption: If the corrupted nextIndex value causes the leader to skip or overwrite log entries on the follower, it can lead to data loss or corruption, as the follower's log would no longer accurately reflect the state of the system.

3. Safety Violations: Raft ensures safety by guaranteeing that only log entries committed by a majority of nodes are applied to the state machine. If the corrupted nextIndex value causes the leader to incorrectly advance the log on a follower without the necessary consensus, it can violate safety constraints and lead to inconsistencies or incorrect results.

Therefore, a corrupted nextIndex value can indeed compromise the safety of the system by introducing inconsistencies, data loss, or violations of safety guarantees within the Raft consensus algorithm.

Question 5:

5. Suppose that you implemented Raft and deployed it with all servers in the same datacenter. Now suppose that you were going to deploy the system with each server in a different datacenter, spread over the world. What changes would you need to make, if any, in the wide-area version of Raft compared to the single-datacenter version, and why?

Answer:
Deploying Raft in a wide-area network (WAN) scenario introduces additional challenges due to increased network latency, potential network partitions, and communication overhead. To ensure the reliability and performance of Raft in a distributed setting across different datacenters, several changes or considerations need to be made:

1. *Heartbeat and Election Timeout Adjustments*: Since network latencies can vary significantly across datacenters, the heartbeat and election timeout parameters may need to be adjusted to accommodate the increased latency. Longer timeouts may be necessary to account for network delays and prevent premature leader elections or false timeouts.

2. *Network Partition Handling*: WAN environments are more prone to network partitions, where communication between datacenters may be temporarily disrupted. Raft needs to handle network partitions gracefully, ensuring that the system remains available and eventually achieves consensus once the partition is resolved. This may involve implementing mechanisms such as dynamic cluster reconfiguration or adaptive timeout adjustments.

3. *Quorum Size and Communication Overhead*: In Raft, a majority of nodes need to agree on log entries for them to be committed. In a WAN scenario with nodes spread across multiple datacenters, the quorum size may need to be adjusted to maintain fault tolerance while minimizing communication overhead. A larger quorum size may be necessary to ensure resilience against datacenter failures but can increase communication latency.

4. *Network Partition Detection*: Implementing mechanisms for detecting network partitions and dynamically adjusting Raft configurations (e.g., election timeouts, quorum sizes) based on network conditions can help improve system resilience and performance in WAN deployments.

This may involve utilizing network monitoring tools or implementing custom heartbeat mechanisms for detecting network disruptions.

5. *Consistency Guarantees and Performance Trade-offs*: In WAN deployments, achieving strong consistency guarantees across datacenters may incur significant performance overhead due to increased latency. Depending on the application requirements, it may be necessary to relax consistency guarantees (e.g., using eventual consistency) to improve system performance and availability in WAN environments.

6. *Security Considerations*: Deploying Raft across different datacenters introduces additional security challenges, such as securing inter-datacenter communication, authenticating nodes across geographically distributed environments, and protecting against malicious attacks or network intrusions. Implementing robust security measures, such as encryption, authentication, and access control, is essential to safeguard the integrity and confidentiality of Raft communications in a WAN deployment.

Overall, deploying Raft in a wide-area network requires careful consideration of network characteristics, latency constraints, fault tolerance requirements, and security considerations to ensure the reliability, performance, and consistency of the distributed system across geographically dispersed datacenters.

Question 6:

6. Recall that each server stores three pieces of information on its disk: its current term, its most recent vote, and all of the log entries it has accepted.

   (a) Suppose that a follower F crashes, and when F restarts, its most recent vote has been lost. Is it safe for F to rejoin the cluster (assuming no modifications to the algorithm)? Explain your answer.

   (b) Now suppose that a follower F's log is truncated during a crash, losing some of the entries at the end. Is it safe for F to rejoin the cluster (assuming no modifications to the algorithm)? Explain your answer.

Answer:
Let's address each scenario:

(a) If a follower F crashes and loses its most recent vote upon restart, it is generally safe for F to rejoin the cluster without modifications to the Raft algorithm. Raft is designed to handle situations where nodes may crash and restart without compromising safety. When F reenters the cluster, it will start as a follower and participate in leader election processes. Since F's vote history is not critical to the safety of the system, its absence does not jeopardize the correctness of Raft's operation. F will simply follow the normal process of receiving AppendEntries RPCs from the leader and updating its state accordingly, eventually catching up with the rest of the cluster.

(b) If a follower F's log is truncated during a crash, losing some of the entries at the end, it may not be safe for F to rejoin the cluster without modifications to the Raft algorithm. The safety of

Raft relies on the assumption that all committed log entries are retained and applied to the state machine. If F's log is truncated, it may have missed some committed entries, leading to inconsistencies and potentially violating safety properties. In this case, F should not rejoin the cluster until it can synchronize its log with the rest of the cluster, either by requesting missing entries from other nodes or by undergoing a process of log compaction and re-synchronization. Modifications to the Raft algorithm may be necessary to handle scenarios where nodes have lost log entries and need to catch up with the rest of the cluster safely.

In summary, while it is generally safe for a follower to rejoin the cluster after a crash without its most recent vote, losing log entries can compromise the safety of the system, requiring additional precautions or modifications to ensure consistency and correctness.

Question 7:

7. It is possible for a leader to continue operating even after other servers have decided that it crashed and elected a new leader. The new leader will have contacted a majority of the cluster and updated their terms, so the old leader will step down as soon as it communicates with any of these servers. However, in the meantime it can continue to act as leader and issue requests to followers that have not yet been contacted by the new leader; furthermore, clients may continue to send requests to the old leader. We know that the old leader cannot commit any new log entries it receives after the election has completed, since it would need to contact at least one of the servers in the electing majority to do this. But, is it possible for an old leader to execute a successful AppendEntries RPC that completes the commitment of an old log entry that was received before the election started? If so, explain how this could happen, and discuss whether or not this will cause problems for the Raft protocol. If this cannot happen, explain why.

Answer:
In the Raft consensus algorithm, once a new leader has been elected and it has replicated its log entries to a majority of the cluster, including followers and potentially other candidates, the old leader will eventually step down upon realizing that its term is outdated. However, before stepping down, the old leader may still continue to receive AppendEntries RPC requests from followers and clients, and it might attempt to replicate log entries to its followers.

Regarding the scenario where an old leader attempts to execute a successful AppendEntries RPC that completes the commitment of an old log entry received before the election started, let's consider the following:

1. *AppendEntries RPC for Old Log Entry*: If the old leader has received an AppendEntries RPC request for a log entry before the new leader was elected, it may attempt to replicate this log entry to its followers.

2. *Commitment of Old Log Entry*: The old leader can successfully replicate the old log entry to its followers, and if a majority of the cluster acknowledges the receipt of this entry, it could potentially commit it to its own log. However, since the new leader has already been elected and has replicated its log entries to a majority of the cluster, including followers and potentially other candidates, the committed log entries from the old leader would be outdated and conflict with the log entries of the new leader.

3. *Problem for Raft Protocol*: This scenario could cause problems for the Raft protocol because it would lead to inconsistencies in the replicated log across the cluster. The Raft protocol relies on a consistent log replication process to ensure that all nodes agree on the same sequence of log entries. If an old leader were to commit outdated log entries after a new leader has been elected, it would violate the consistency guarantees of the Raft protocol and potentially lead to divergent states and safety violations.

Therefore, while it's technically possible for an old leader to attempt to commit old log entries after a new leader has been elected, it would cause significant problems for the Raft protocol, leading to inconsistencies and potentially compromising the safety and correctness of the distributed system. The Raft protocol is designed to handle these scenarios by ensuring that only the log entries from the current leader are committed and applied to the state machine, while outdated log entries from previous leaders are discarded.