Understanding the Raft Consensus Algorithm

Shobhit Singh & Akhoury Shauryam

Chennai Mathematical Institute

October 14, 2024

Outline

What is a Consensus Protocol?

What is a Consensus Protocol?

- A consensus protocol is a method used in distributed systems to ensure that all the nodes agree on a single data value or state, even in the presence of failures or unreliable communication.

What is a Consensus Protocol?

- A consensus protocol is a method used in distributed systems to ensure that all the nodes agree on a single data value or state, even in the presence of failures or unreliable communication.
- Types of Fault Tolerance:

What is a Consensus Protocol?

- A consensus protocol is a method used in distributed systems to ensure that all the nodes agree on a single data value or state, even in the presence of failures or unreliable communication.
- Types of Fault Tolerance:
  - **Byzantine Fault Tolerance (BFT):** Deals with nodes behaving arbitrarily or maliciously.

What is a Consensus Protocol?

- A consensus protocol is a method used in distributed systems to ensure that all the nodes agree on a single data value or state, even in the presence of failures or unreliable communication.
- Types of Fault Tolerance:
  - **Byzantine Fault Tolerance (BFT):** Deals with nodes behaving arbitrarily or maliciously.
  - **Crash Fault Tolerance (CFT):** Deals with nodes that might crash but do not exhibit malicious behavior.

What is Raft?

What is Raft?

- Raft is a consensus algorithm designed for managing a replicated log.

What is Raft?

- Raft is a consensus algorithm designed for managing a replicated log.
- It ensures that a distributed system's nodes agree on a sequence of operations, even with failures.

What is Raft?

- Raft is a consensus algorithm designed for managing a replicated log.
- It ensures that a distributed system's nodes agree on a sequence of operations, even with failures.
- Raft is known for being easier to understand compared to Paxos.

Raft Process Overview

Raft Process Overview

- **Term:** Each term begins with an election, and the term increments each time after it ends.

Raft Process Overview

- **Term:** Each term begins with an election, and the term increments each time after it ends.
- **Election:** Candidates request votes from other nodes. A majority vote makes a candidate the leader.

Raft Process Overview

- **Term:** Each term begins with an election, and the term increments each time after it ends.
- **Election:** Candidates request votes from other nodes. A majority vote makes a candidate the leader.
- **Log Replication:** The leader handles client requests and replicates logs to all the followers.

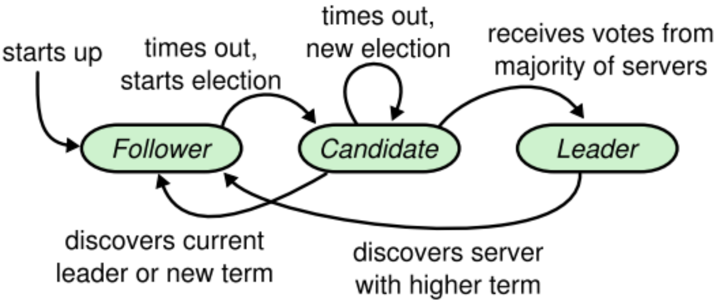
Raft Process Overview

- **Term:** Each term begins with an election, and the term increments each time after it ends.
- **Election:** Candidates request votes from other nodes. A majority vote makes a candidate the leader.
- **Log Replication:** The leader handles client requests and replicates logs to all the followers.
- **Commitment:** After a majority of servers replicate the log entries, they are committed.

Raft Process Overview

- **Term:** Each term begins with an election, and the term increments each time after it ends.
- **Election:** Candidates request votes from other nodes. A majority vote makes a candidate the leader.
- **Log Replication:** The leader handles client requests and replicates logs to all the followers.
- **Commitment:** After a majority of servers replicate the log entries, they are committed.
- **Failure Handling:** If the leader fails, a new election is initiated.

Overview (Continued)



States for Servers

States for Servers

- **currentTerm:** Latest term the server has seen.

States for Servers

- **currentTerm:** Latest term the server has seen.
- **votedFor:** Candidate ID that received the vote in the current term (or null if none).

States for Servers

- **currentTerm:** Latest term the server has seen.
- **votedFor:** Candidate ID that received the vote in the current term (or null if none).
- **log:** Log entries.

States for Servers

- **currentTerm:** Latest term the server has seen.
- **votedFor:** Candidate ID that received the vote in the current term (or null if none).
- **log:** Log entries.
- **commitIndex:** Index of the highest log entry known to be committed.

States for Servers

- **currentTerm:** Latest term the server has seen.
- **votedFor:** Candidate ID that received the vote in the current term (or null if none).
- **log:** Log entries.
- **commitIndex:** Index of the highest log entry known to be committed.
- **lastApplied:** Index of the highest log entry applied to the state machine.

Voting Protocol

Voting Protocol

- During the start of a term, a candidate sends these arguments:

Voting Protocol

- During the start of a term, a candidate sends these arguments:

Voting Protocol

- During the start of a term, a candidate sends these arguments:
  - **term:** Candidate's term.

Voting Protocol

- During the start of a term, a candidate sends these arguments:
  - **term:** Candidate's term.
  - **candidateId:** Candidate's ID.

Voting Protocol

- During the start of a term, a candidate sends these arguments:
  - **term:** Candidate's term.
  - **candidateId:** Candidate's ID.
  - **lastLogIndex:** Index of candidate's last log entry.

Voting Protocol

- During the start of a term, a candidate sends these arguments:
  - **term:** Candidate's term.
  - **candidateId:** Candidate's ID.
  - **lastLogIndex:** Index of candidate's last log entry.
  - **lastLogTerm:** Term of candidate's last log entry.

Voting Protocol

- During the start of a term, a candidate sends these arguments:
    - **term:** Candidate's term.
    - **candidateId:** Candidate's ID.
    - **lastLogIndex:** Index of candidate's last log entry.
    - **lastLogTerm:** Term of candidate's last log entry.
- Nodes reply **False** if the term is less than **currentTerm**.

Voting Protocol

- During the start of a term, a candidate sends these arguments:
  - **term:** Candidate's term.
  - **candidateId:** Candidate's ID.
  - **lastLogIndex:** Index of candidate's last log entry.
  - **lastLogTerm:** Term of candidate's last log entry.
- Nodes reply **False** if the term is less than **currentTerm**.
- If **votedFor** is null or matches the candidate, and the candidate's log is at least as up-to-date, grant the vote.

Logging Protocol

Logging Protocol

- The leader sends these arguments as a heartbeat:

Logging Protocol

- The leader sends these arguments as a heartbeat:
  - **term:** Leader's term.

Logging Protocol

- The leader sends these arguments as a heartbeat:
  - **term:** Leader's term.
  - **leaderId:** Leader's ID.

Logging Protocol

- The leader sends these arguments as a heartbeat:
  - **term:** Leader's term.
  - **leaderId:** Leader's ID.
  - **prevLogIndex:** Index of the log entry preceding the new ones.

Logging Protocol

- The leader sends these arguments as a heartbeat:
  - **term:** Leader's term.
  - **leaderId:** Leader's ID.
  - **prevLogIndex:** Index of the log entry preceding the new ones.
  - **prevLogTerm:** Term of **prevLogIndex**.

Logging Protocol

- The leader sends these arguments as a heartbeat:
  - **term:** Leader's term.
  - **leaderId:** Leader's ID.
  - **prevLogIndex:** Index of the log entry preceding the new ones.
  - **prevLogTerm:** Term of **prevLogIndex**.
  - **entries:** The newest packet of data to be added.

Logging Protocol

- The leader sends these arguments as a heartbeat:
  - **term:** Leader's term.
  - **leaderId:** Leader's ID.
  - **prevLogIndex:** Index of the log entry preceding the new ones.
  - **prevLogTerm:** Term of **prevLogIndex**.
  - **entries:** The newest packet of data to be added.
  - **leaderCommit:** Leader's **commitIndex**.

Logging Protocol

- The leader sends these arguments as a heartbeat:
  - **term:** Leader's term.
  - **leaderId:** Leader's ID.
  - **prevLogIndex:** Index of the log entry preceding the new ones.
  - **prevLogTerm:** Term of **prevLogIndex**.
  - **entries:** The newest packet of data to be added.
  - **leaderCommit:** Leader's **commitIndex**.
- Followers randomly generate a timeout. If they don't receive a heartbeat, they become candidates and initiate a re-election.

Logging Protocol (Continued)

Logging Protocol (Continued)

- Followers implement the following:

Logging Protocol (Continued)

- Followers implement the following:
  - Reply **False** if **term ¡ currentTerm**.

Logging Protocol (Continued)

- Followers implement the following:
  - Reply **False** if **term ¡ currentTerm**.
  - Reply **False** if the log doesn't contain an entry at **prevLogIndex**.

Logging Protocol (Continued)

- Followers implement the following:
  - Reply **False** if **term ¡ currentTerm**.
  - Reply **False** if the log doesn't contain an entry at **prevLogIndex**.
  - For a conflict with the entry, delete the existing entry and the ones following it.

Logging Protocol (Continued)

- Followers implement the following:
  - Reply **False** if **term ¡ currentTerm**.
  - Reply **False** if the log doesn't contain an entry at **prevLogIndex**.
  - For a conflict with the entry, delete the existing entry and the ones following it.
  - Append any new entry not in the log.

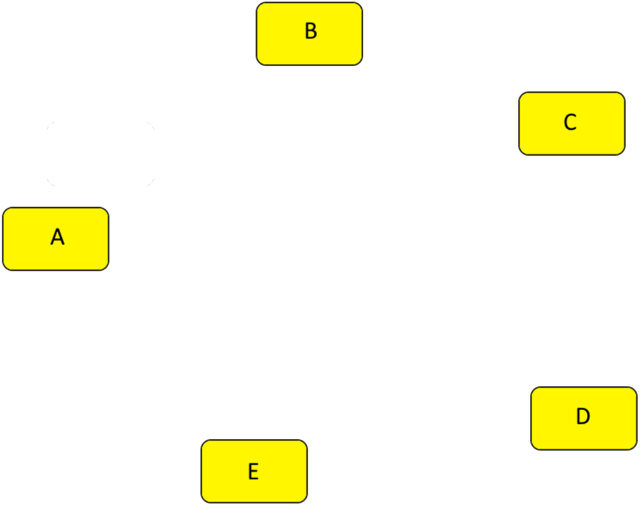
Logging Protocol (Continued)

- Followers implement the following:
  - Reply **False** if **term ¡ currentTerm**.
  - Reply **False** if the log doesn't contain an entry at **prevLogIndex**.
  - For a conflict with the entry, delete the existing entry and the ones following it.
  - Append any new entry not in the log.
  - If **leaderCommit ¿ commitIndex**, set **commitIndex = min(leaderCommit, index of last new entry)**.

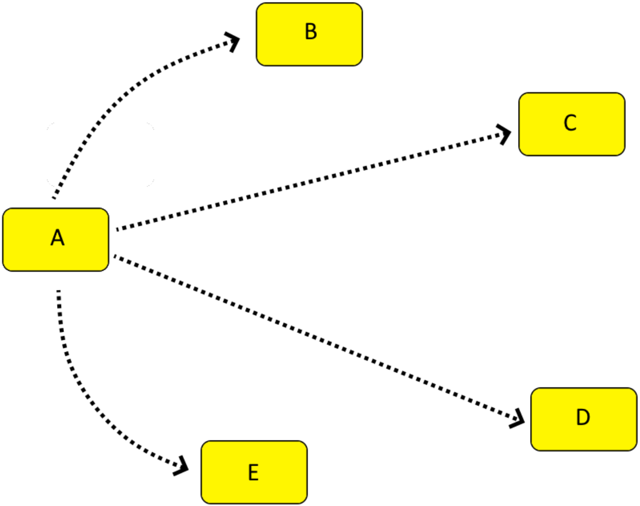
Logging Protocol (Continued)

- Followers implement the following:
  - Reply **False** if **term ¡ currentTerm**.
  - Reply **False** if the log doesn't contain an entry at **prevLogIndex**.
  - For a conflict with the entry, delete the existing entry and the ones following it.
  - Append any new entry not in the log.
  - If **leaderCommit ¿ commitIndex**, set **commitIndex = min(leaderCommit, index of last new entry)**.
- After the leader receives a majority of **Trues**, it requests everyone to commit the latest entry to their state machine.

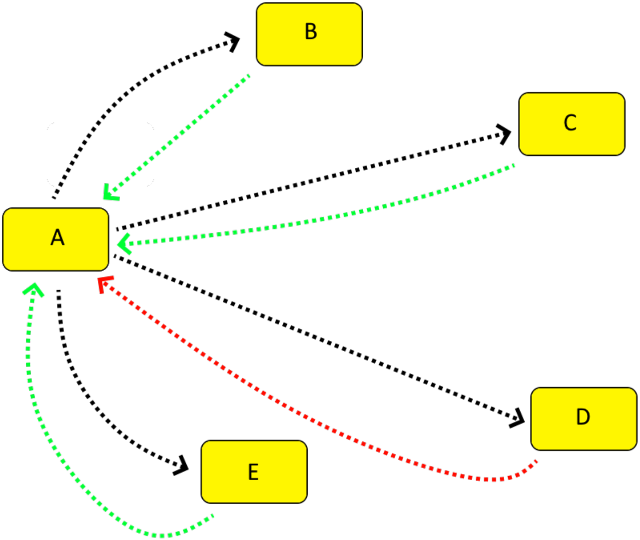
Example



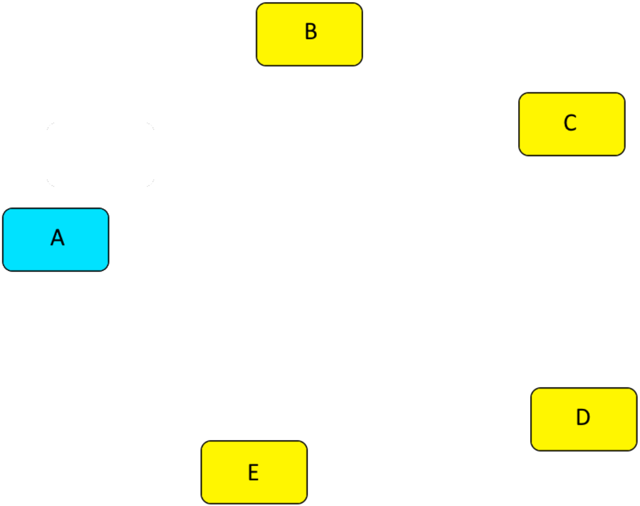
Example



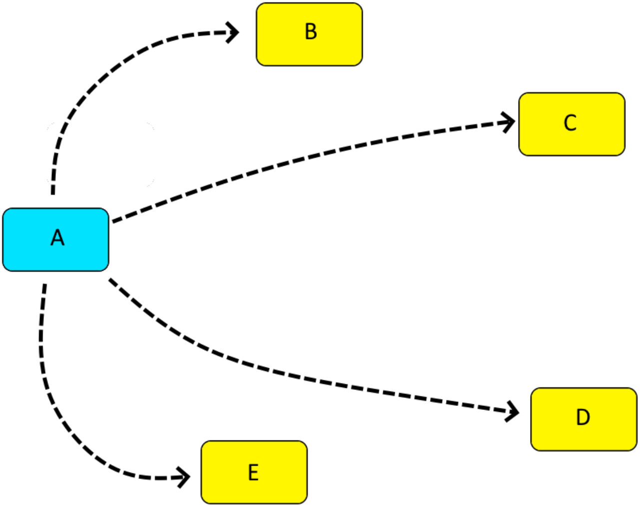
Example



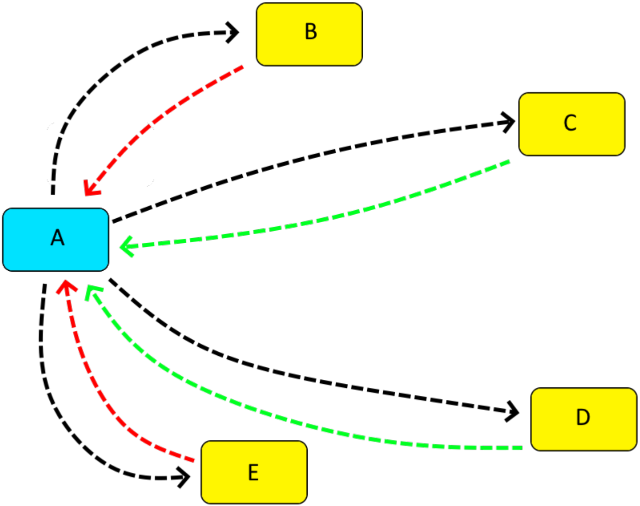
Example



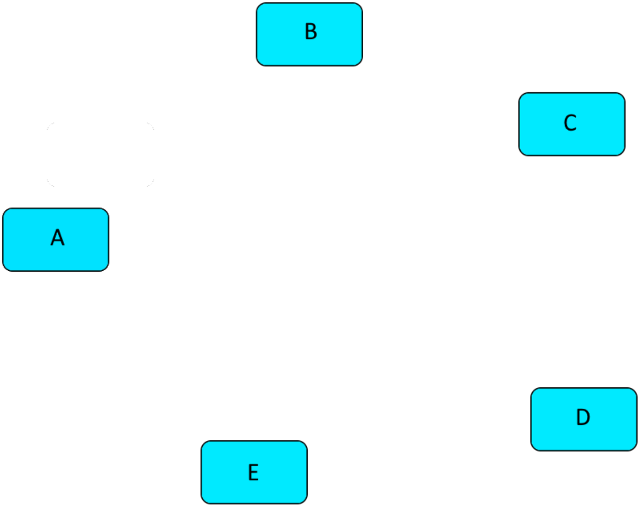
Example



Example



Example



Safety Invariants in Raft

Safety Invariants in Raft

- **Election Safety:** At most one leader per term.

Safety Invariants in Raft

- **Election Safety:** At most one leader per term.
- **Log Matching:** Logs are identical up to a certain point.

Safety Invariants in Raft

- **Election Safety:** At most one leader per term.
- **Log Matching:** Logs are identical up to a certain point.
- **Leader Completeness:** Committed entries are preserved.

Safety Invariants in Raft

- **Election Safety:** At most one leader per term.
- **Log Matching:** Logs are identical up to a certain point.
- **Leader Completeness:** Committed entries are preserved.
- **State Machine Safety:** No conflicting committed entries.

Safety Invariants in Raft

- **Election Safety:** At most one leader per term.
- **Log Matching:** Logs are identical up to a certain point.
- **Leader Completeness:** Committed entries are preserved.
- **State Machine Safety:** No conflicting committed entries.
- **Leader Election Guarantee:** Only the most up-to-date candidate can be elected.

Crashes in Leaders and Followers

Crashes in Leaders and Followers

- **Follower Crashes:** Upon restarting, it checks if **term ¡ currentTerm**. If true, it requests the leader to send its state machine data for replication.

Crashes in Leaders and Followers

- **Follower Crashes:** Upon restarting, it checks if **term ¡ currentTerm**. If true, it requests the leader to send its state machine data for replication.
- **Leader Crashes:**

Crashes in Leaders and Followers

- **Follower Crashes:** Upon restarting, it checks if **term ¡ currentTerm**. If true, it requests the leader to send its state machine data for replication.
- **Leader Crashes:**
  - Between two terms: A re-election occurs.

Crashes in Leaders and Followers

- **Follower Crashes:** Upon restarting, it checks if **term ¡ currentTerm**. If true, it requests the leader to send its state machine data for replication.
- **Leader Crashes:**
  - Between two terms: A re-election occurs.
  - Before committing: Follows the protocol for a crashed follower.

Comparisons with Paxos

Comparisons with Paxos

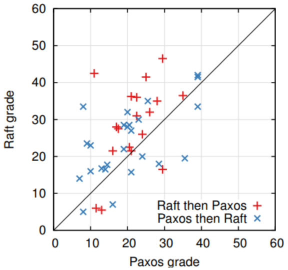
- Raft implements **Snapshotting** to compact the log when it becomes too large.

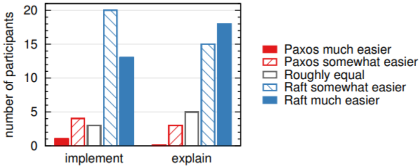
Comparisons with Paxos

- Raft implements **Snapshotting** to compact the log when it becomes too large.
- Studies show that Raft is easier to understand than Paxos.

Comparisons with Paxos

- Raft implements **Snapshotting** to compact the log when it becomes too large.
- Studies show that Raft is easier to understand than Paxos.





Summary

Summary

- Raft ensures consensus in a distributed system through leader election, log replication, and safety invariants.

Summary

- Raft ensures consensus in a distributed system through leader election, log replication, and safety invariants.
- The protocol is designed to be easy to understand and robust in the face of failures.

Future Work

Future Work

- Implement the protocol in Python, simulate, and check the safety properties using an SMT solver like Z3.

Future Work

- Implement the protocol in Python, simulate, and check the safety properties using an SMT solver like Z3.
- Investigate different crash failure scenarios to find more invariants to check.

Questions?

Questions?

Thank You!