CSE446: Blockchain & Cryptocurrencies

Lecture – 5: Distributed Systems & Consensus Algorithms

BRAC UNIVERSITY

Inspiring Excellence

Agenda

- FLP Impossibility, CAP Theorem
- Various consensus algorithms

FLP Impossibility

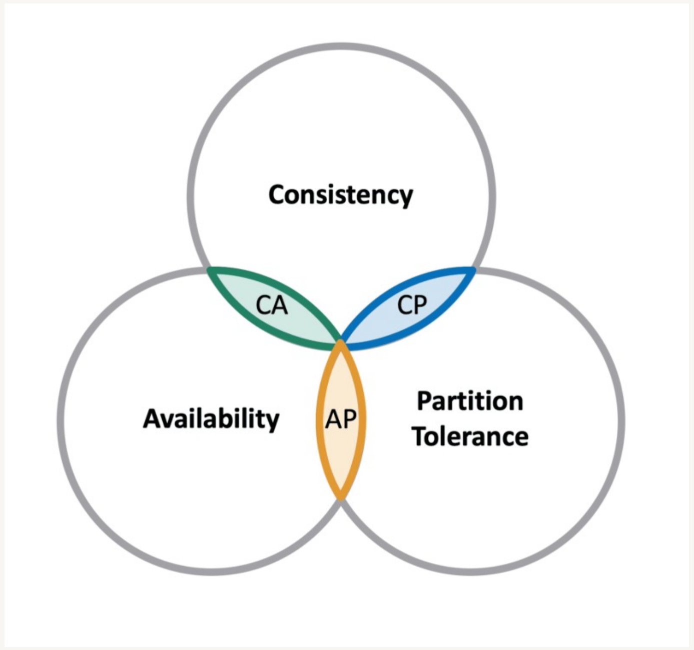
- A well-known theorem*, by Fischer, Lynch and Paterson, called FLP Impossibility has shown that
  - a deterministic consensus protocol cannot satisfy all three properties in an asynchronous network
- Safety and liveness are favoured over fault tolerance in the domain of distributed system applications

*M. J. Fischer, N. A. Lynch, and M. S. Paterson. "Impossibility of distributed consensus with one faulty process". Journal of the ACM (JACM), 32(2):374382, 1985

CAP Theorem

- CAP theorem* states that a shared replicated datastore (or, more generally, a replicated state machine) cannot achieve both consistency and availability when a network partitions in such a way that an arbitrary number of messages might be dropped

Pick any two



https://hazelcast.com/wp-content/uploads/2021/12/cap-theorem-diagram-800x753-1.png

*Brewer, E. A. "Towards robust distributed systems.". In PODC (Vol. 7), July 2000.

Crash-tolerant consensus algorithm

- Algorithms belonging to this class aim to guarantee the atomic broadcast (total order) of messages within the participating nodes in the presence of certain number of node failures
- These algorithms utilise the notion of views or epochs
    - which imply a certain duration of time
- A leader is selected for each epoch who takes decisions regarding the atomic broadcast, and all other nodes comply with its decision
- In case a leader fails due to a crash failure, the protocols elect a new leader to function

Crash-tolerant consensus algorithm

- The best known algorithms belonging to this class can continue to function if the following condition holds:
    - $f < n/2$ where f is the number of faulty nodes and n is the total number of participating nodes
- Examples of some well-known crash-tolerant consensus protocol are
    - Paxos, Viewstamped Replication, ZooKeeper, and Raft

RAFT

- Raft is a distributed consensus algorithm
- It solves the problem of getting multiple servers to agree on a shared state even in the face of failures
- The shared status is usually a data structure supported by a replicated log
- We need the system to be fully operational as long as a majority of the servers are up

RAFT

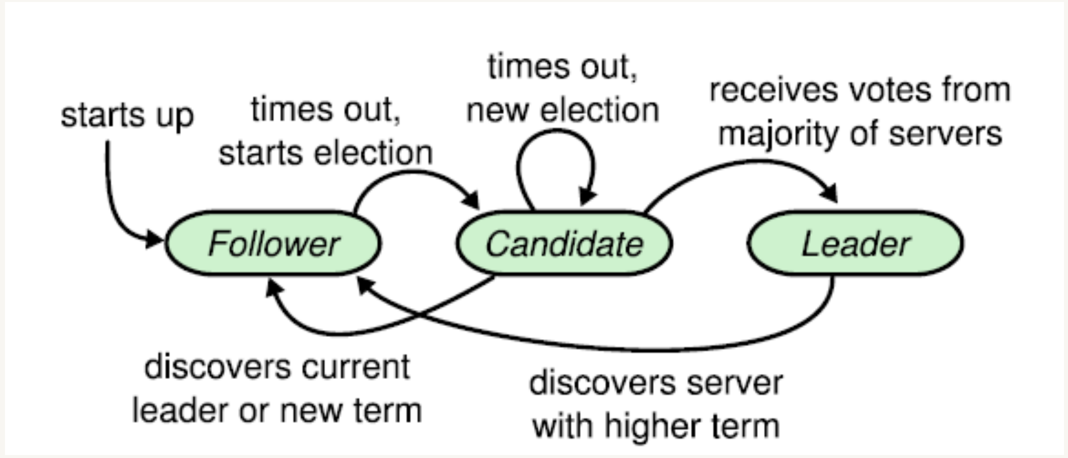
- Raft works by electing a leader in the cluster
- The leader is responsible for accepting client requests and managing the replication of the log to other servers
- The data flows only in one direction: leader -> other servers
- Raft decomposes consensus into three sub-problems
  - Leader Election: A new leader needs to be elected in case of the failure of an existing one
  - Log replication: The leader needs to keep the logs of all servers in sync with its own through replication
  - Safety: to ensure consistency across all nodes

RAFT slides prepared from this location:https://www.freecodecamp.org/news/in-search-of-an-understandable-consensus-algorithm-a-summary-4bc294c97e0d/

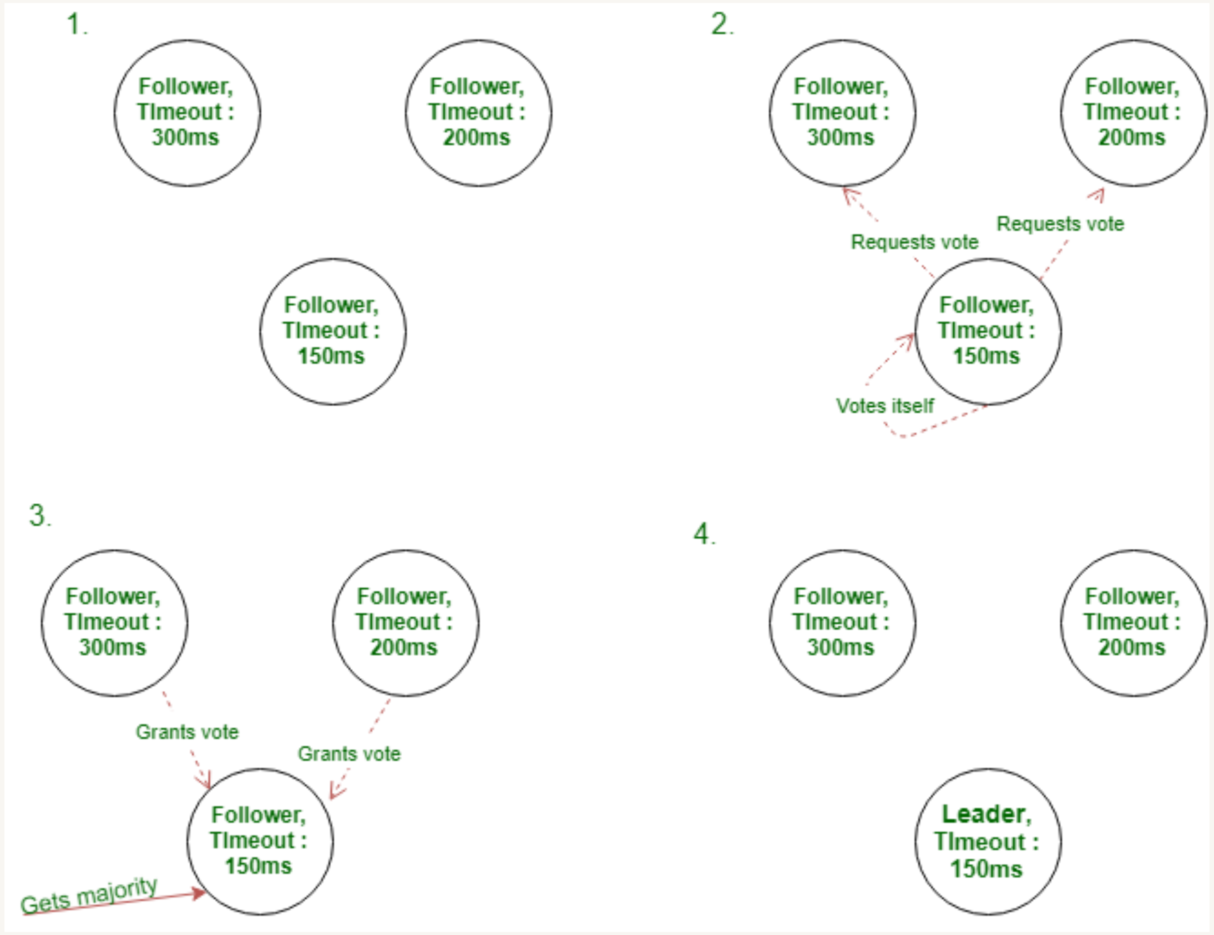
RAFT: leader election

- An epoch is termed as a term in RAFT
- A leader is selected in each term
- If a candidate wins the election, it remains the leader for the rest of the term
  - If the vote is split, then that term ends without a leader
- The term number increases monotonically and is also exchanged in every communication

Each node can only be a leader, follower, or candidate


starts up
times out, starts election
times out, new election
receives votes from majority of servers
Follower
Candidate
Leader
discovers current leader or new term
discovers server with higher term

RAFT: leader election


1.
Follower, TImeout : 300ms
Follower, TImeout : 200ms
Follower, TImeout : 150ms
2.
Follower, TImeout : 300ms
Follower, TImeout : 200ms
Requests vote
Requests vote
Follower, TImeout : 150ms
Votes itself
3.
Follower, TImeout : 300ms
Follower, TImeout : 200ms
Grants vote
Grants vote
Follower, TImeout : 150ms
Gets majority
4.
Follower, TImeout : 300ms
Follower, TImeout : 200ms
Leader, TImeout : 150ms

https://media.geeksforgeeks.org/wp-content/uploads/raft-leader-election.png

RAFT: log replication

- A client communicates with the leader to update the log (append to the log)
- The leader updates its log entry and replicates the value to the followers
- Each follower updates their log entries and replies back to the leader
- The leader commits the value to the log if the majority of the nodes reply back
- The leader replicates this commitment to the followers and they also commit the value

RAFT slides prepared from this location:https://www.freecodecamp.org/news/in-search-of-an-understandable-consensus-algorithm-a-summary-4bc294c97e0d/

RAFT: safety

- RAFT ensures that it can provide consistency (safety) and availability when the network partitions using the following rule:
  - If one of the previous leaders has committed a log entry at a particular index, no other leader can apply a different log entry for that index
- This is assuming that this condition holds: $f < n/2$
- A fantastic visualisation of RAFT can be viewed from this location: https://thesecretlivesofdata.com/raft

RAFT slides prepared from this location:https://www.freecodecamp.org/news/in-search-of-an-understandable-consensus-algorithm-a-summary-4bc294c97e0d/

Byzantine consensus algorithm

- Byzantine algorithms aim to reach consensus amid of certain nodes exhibiting Byzantine behaviour
- Such Byzantine nodes are assumed to be under the control of an adversary and behave unpredictably with malicious intent
- Similar to any crash-tolerant consensus protocol, these protocols also utilise the concept of views/epochs
- A leader is elected in each view to order messages for atomic broadcast
  - other honest nodes are assumed to follow the instructions from the leader

Practical Byzantine Fault Tolerance (PBFT)

- One of the most well-known algorithms under this class is called Practical Byzantine Fault Tolerant (PBFT)
- It can achieve consensus in the presence of a certain number of Byzantine nodes under an eventual synchronous network assumption
- PBFT uses cryptographic algorithms such as signature, signature verification, and hash to ensure that everything stays irrevocable, unforgeable, and indisputable

PBFT slides prepared from this location:https://coredevs.medium.com/an-introduction-to-pbft-consensus-algorithm-11cbd90aaec

Practical Byzantine Fault Tolerance (PBFT)

- The following conditions are required
- $f < n/3$, where f is the number of Byzantine nodes and n denotes the number of total nodes participating in the network, that is at least $n = 3f+1$
  - A distributed system that contains $3f+1$ nodes can have at most f faulty nodes (byzantine nodes)

PBFT: roles

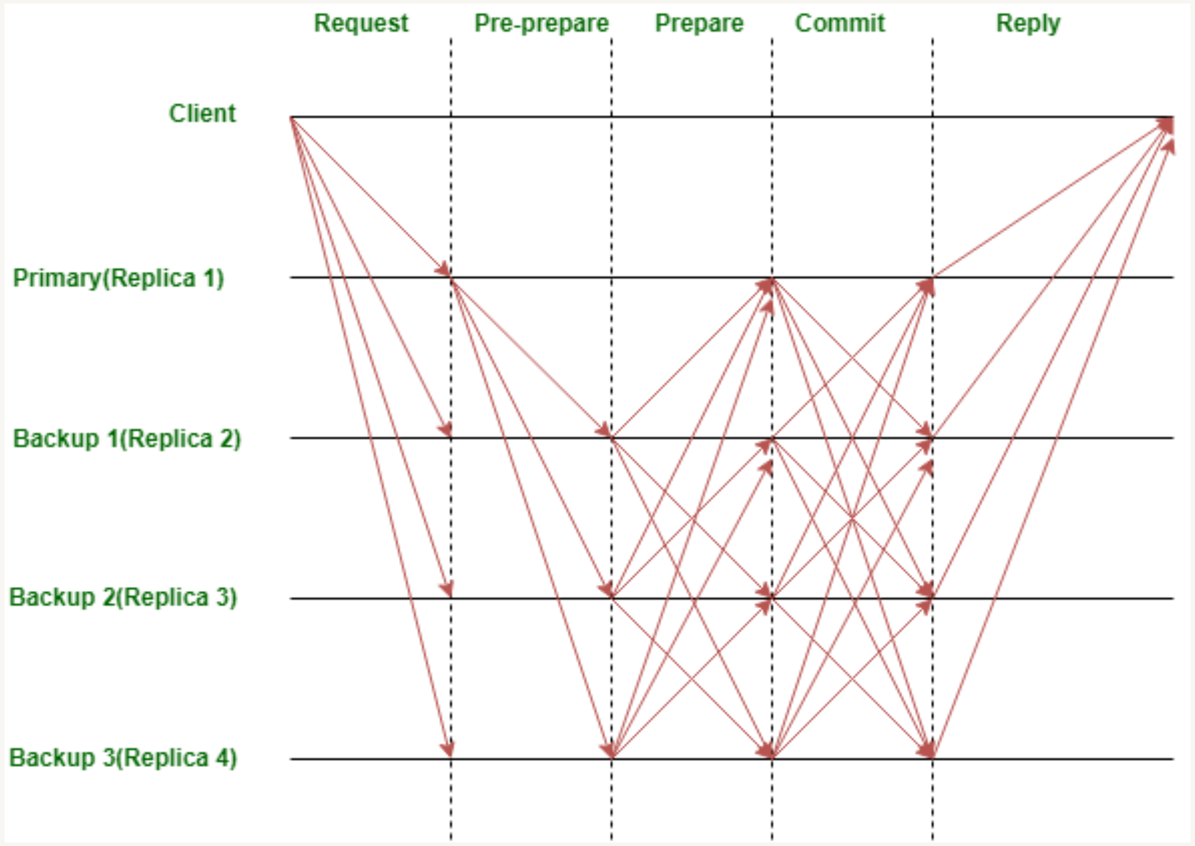
- An epoch is known as a view (a consensus round) in PBFT
- There are three different roles in PBFT
- Client:
  - Client nodes send transaction requests
- Primary:
  - These are main nodes which act as leaders
  - They initiate the consensus mechanism
  - In each view, there is one and only one primary node

PBFT: roles

- Replica:
  - These nodes are responsible for processing each request
  - There are many replica nodes and they all act in the same fashion
- Both Primary and Replica nodes are considered as consensus nodes, because they participate in the consensus process

PBFT: steps

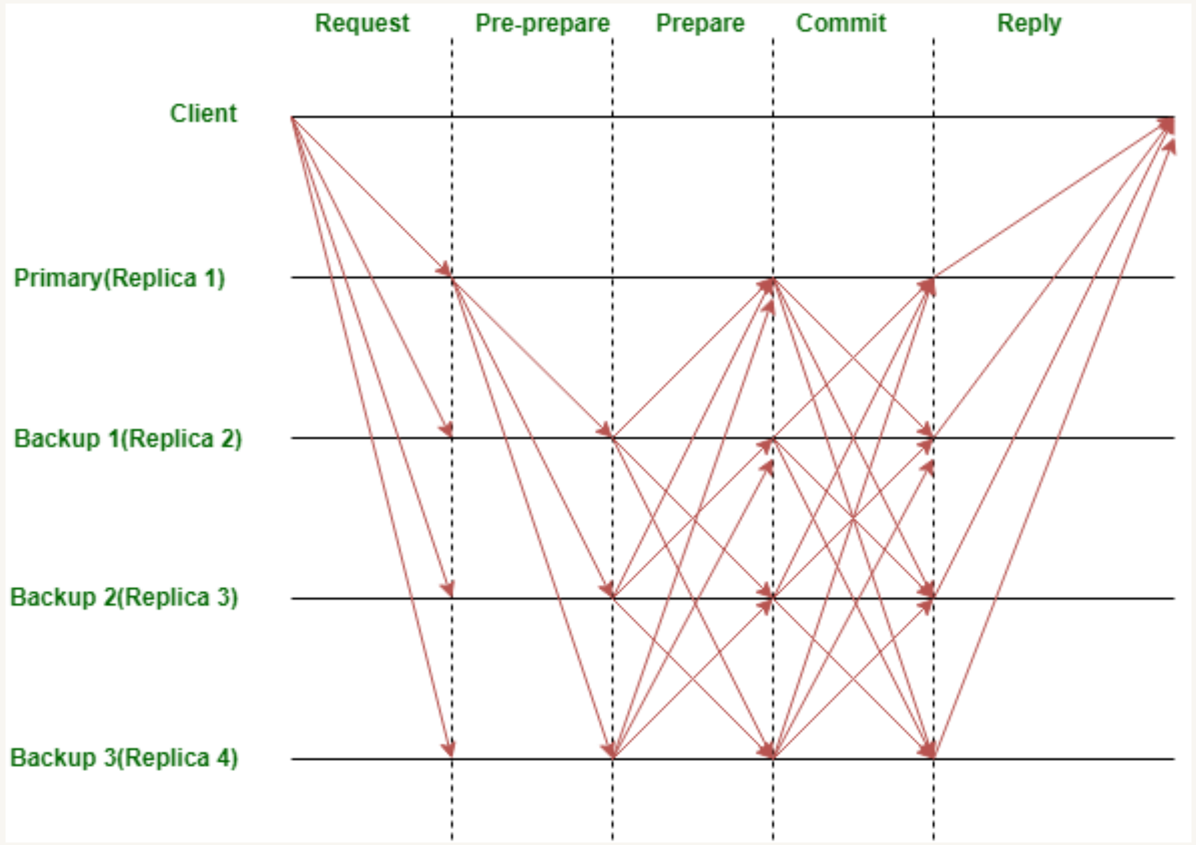
- Client sends a request to the system
- Consensus nodes (Replica1, Replica2, Replica3 & Replica4) receive the request
- Then the system participates in a three-phase consensus protocol: pre-prepare, prepare and commit



https://media.geeksforgeeks.org/wp-content/uploads/pbft-algorithm-communication-visual.png

PBFT: steps – pre-prepare

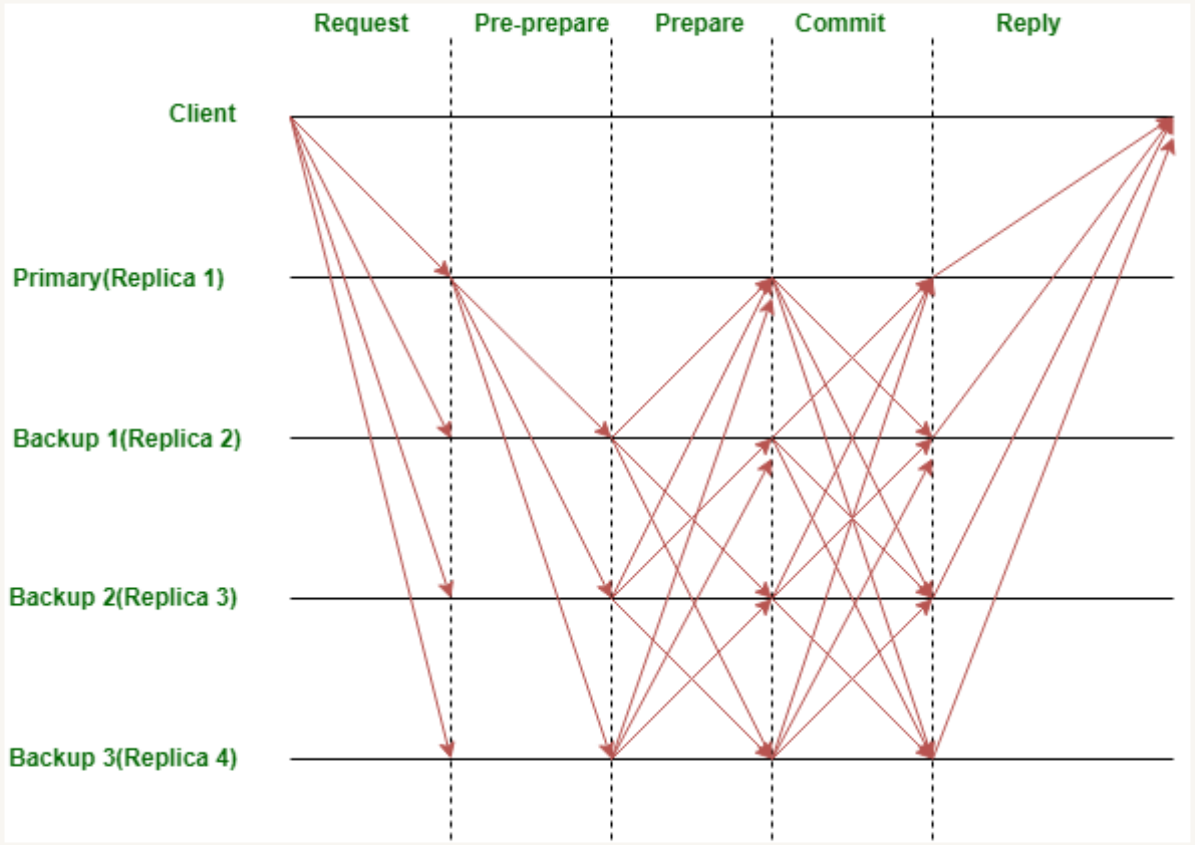
- Primary node verifies the request and generates a pre-prepare message which is broadcast to all replica nodes
- After receiving the messages, each replica node will validate the message
- If validated, each replica node will broadcast a corresponding prepare message



https://media.geeksforgeeks.org/wp-content/uploads/pbft-algorithm-communication-visual.png

PBFT: steps – prepare

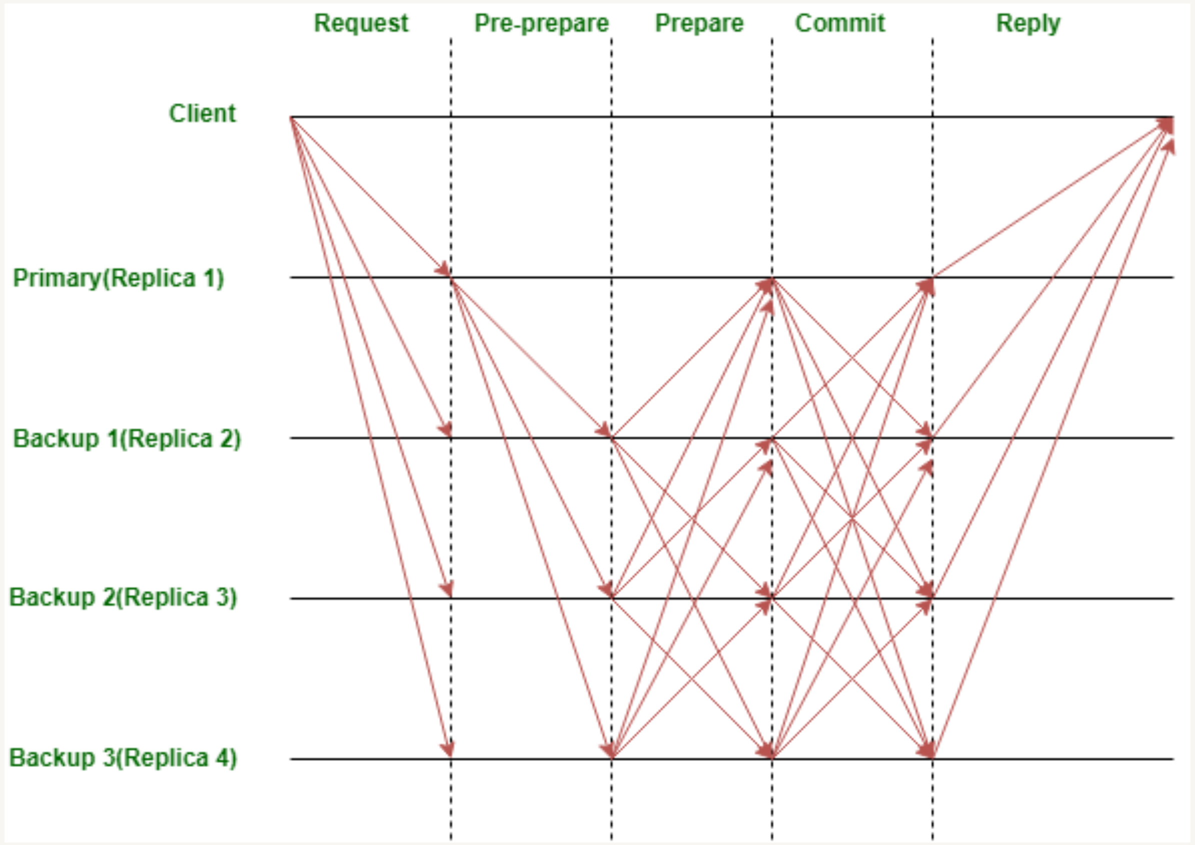
- Each node waits until 2f+1 prepare messages are received
- Then each node will broadcast commit messages to other nodes
  - message implying that it is ready for processing the request



https://media.geeksforgeeks.org/wp-content/uploads/pbft-algorithm-communication-visual.png

PBFT: steps – commit

- Each node waits until 2f+1 commit messages are received
- Then, the node processes each request, which changes the system states
- Each node replies back to the client
- The primary is rotated when there is a proposal from the replicas to do so or the primary remains unavailable for a duration
    - This mechanism is called a view change



https://media.geeksforgeeks.org/wp-content/uploads/pbft-algorithm-communication-visual.png

THANK YOU
QUESTIONS?
memegenerator.net