“Is it feasible for a set of machines to come to an agreement, and how?”

Distributed Consensus

Dayanand Ambawade

Overview:

What is Distributed System?

Examples of DS

Key characteristics of DS

DS-Client-Server and P2P

Consensus, DC

DC requirements: Safety and Liveliness

Properties of DC: AVIT

Randomly

Synchronous, Asynchronous or Semisynchronous

Replication-Active /Passive SMR

Types of faults- CFT, BFT, Partition Faults

Overview

- Introducing the consensus problem
- Analysis and design
- Classification
- Algorithms
- Choosing an algorithm

Introducing the consensus problem

Distributed systems are classified into two main categories:

Message passing

shared memory.

Concept of Consensus and Blockchain

Blockchain is a distributed system that relies upon a **consensus mechanism,** which **ensures the safety and liveness of the blockchain network.**

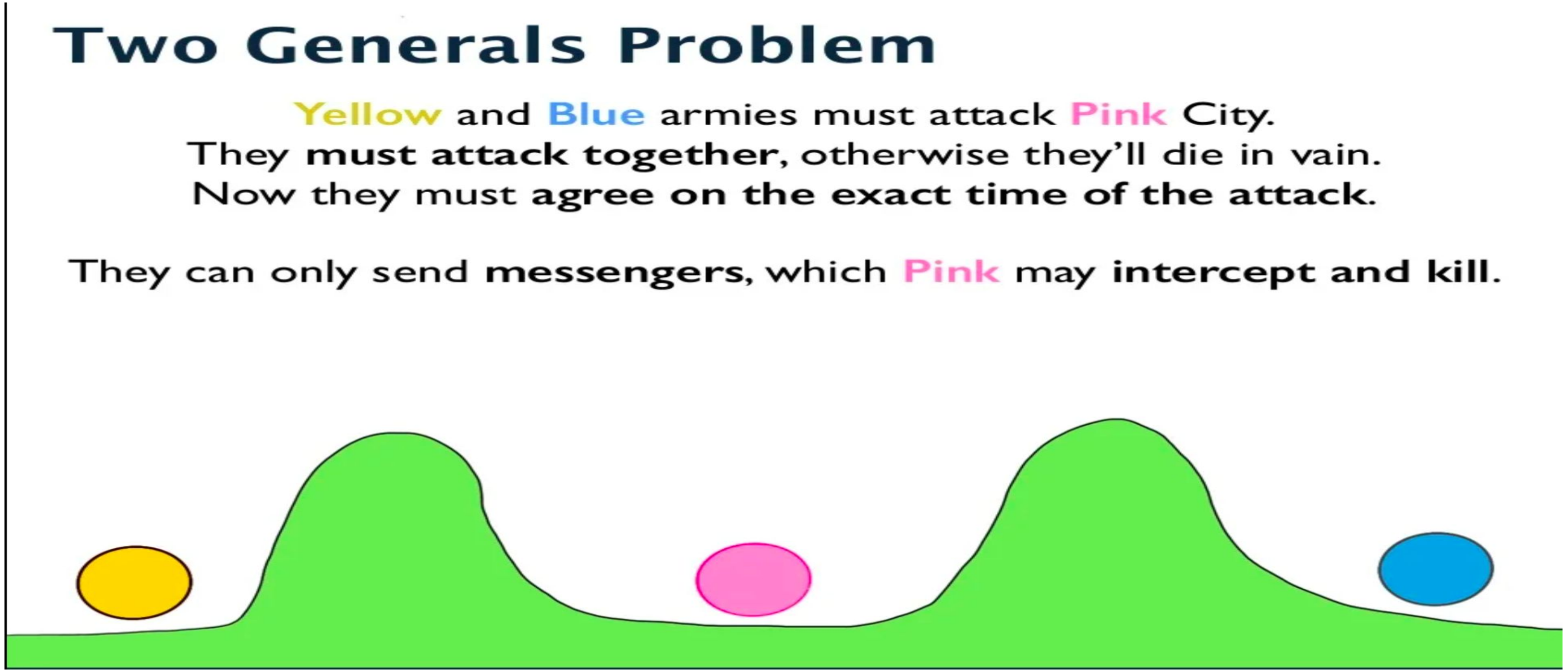
The Byzantine generals problem

The **problem of reaching agreement in the presence of faults or Byzantine** consensus was first formulated by M. Pease, R. Shostak, and L. Lamport.

In distributed systems, a common goal is to achieve consensus (agreement) among nodes on the network even in the presence of faults.

In order to explain the problem, Lamport came up with an allegorical representation of the problem and named it the Byzantine generals problem.

The Byzantine Generals Problems

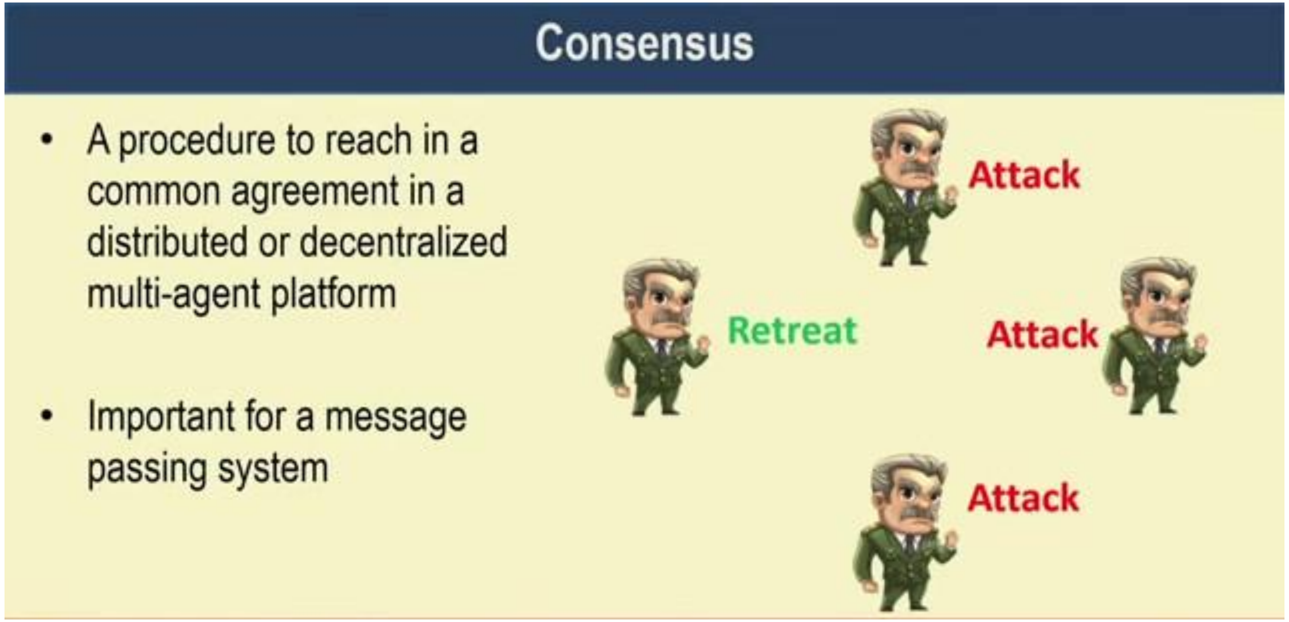


Two Generals Problem

Yellow and Blue armies must attack Pink City.
They **must attack together**, otherwise they'll die in vain.
Now they must **agree on the exact time of the attack.**

They can only send **messengers**, which Pink may **intercept and kill.**

Byzantine Generals Problems



Consensus

- A procedure to reach in a common agreement in a distributed or decentralized multi-agent platform
- Important for a message passing system

Attack
Retreat
Attack
Attack

Fault tolerance:

Availability

Reliability

Safety

Maintainability

Correctness

Types of fault-tolerant consensus

Fault-tolerant algorithms can be divided into two types of fault-tolerance. The first is **Crash fault-tolerance** (CFT) and the other is **Byzantine fault-tolerance (BFT)**.

CFT covers only crash faults or, in other words, benign faults. In contrast,

BFT deals with the type of faults that are arbitrary and can even be malicious.

Replication

Replication is a standard approach to make a system fault-tolerant.

Replication results in a synchronized copy of data across all nodes in a network.

This technique improves the fault tolerance and availability of the network.

This means that even if some of the nodes become faulty, the overall system/network remains available due to the data being available on multiple nodes.

Replication:

There are two main types of replication techniques:

- **Active replication,** which is a type where each replica becomes a copy of the original state machine replica.
- **Passive replication**, which is a type where there is only a single copy of the state machine in the system kept by the primary node, and the rest of the nodes/replicas only maintain the state.

State machine replication

State machine replication (SMR) is a de facto technique that is used to provide deterministic replication services in order to achieve fault tolerance in a distributed system.

State machine replication (SMR) was first proposed by Lamport in 1978 in his paper.

SMR

The fundamental idea behind SMR can be summarized as follows:

1. All servers always start with the same initial state.

2. All servers receive requests in a totally ordered fashion (sequenced as generated from clients).

3. All servers produce the same deterministic output for the same input.

SMR Working

State machine replication is implemented under a primary/backup paradigm, where a primary

node is responsible for receiving and broadcasting client requests.

This broadcast mechanism is called total order broadcast or atomic broadcast, which ensures that backup or replica nodes receive and execute the same requests in the same sequence as the primary.

Consequently, this means that all replicas will eventually have the same state as the primary,

thus resulting in achieving consensus.

In other words, this means that total order broadcast and distributed consensus are equivalent problems; if you solve one, the other is solved too.

FLP impossibility

FLP impossibility is a fundamental unsolvability result in distributed computing theory that states that in an asynchronous environment, the deterministic consensus is impossible, even if only one process is faulty.

FLP Impossibility

These techniques include:

- **Failure detectors,** which can be seen as oracles associated with processors to detect failures.
- **Randomized algorithms** have been introduced to provide a probabilistic termination guarantee. The core idea behind the randomized protocols is that the processors in such protocols can make a random choice of decision value if the processor does not receive the required quorum of trusted messages.
- **Synchrony assumptions**, where additional synchrony and timing assumptions are made to ensure that the consensus algorithm terminates and makes progress.

Bounds

Lower bounds on the number of processors to solve consensus

In the case of CFT, at least 2F + 1 number of nodes is required to achieve consensus.

• In the case of BFT, at least 3F + 1 number of nodes is required to achieve consensus.

F represents the number of failures.

Analysis and design

Model: we need to define a model under which our algorithm will run.

This model provides **some assumptions about the operating environment of the algorithm and provides a way to intuitively study and reason about the various properties of the algorithm.**

Processes

Processes communicate with each other by **passing messages** to each other. This is why these systems are called message-passing distributed systems.

There is another class, **called shared memory**, which we will not discuss here as we are only dealing with message-passing systems.

Timing assumptions

Synchrony:

In synchronous systems, there is a known upper bound on the communication and processor delays. Synchronous algorithms are designed to be run on synchronous networks. At a fundamental level, in a synchronous system, a message sent by a processor to another is received by the receiver in the same communication round as it is sent.

Asynchrony:

In asynchronous systems, there is no upper bound on the communication and processor delays. In other words, it is impossible to define an upper bound for communication and processor delays in asynchronous systems. Asynchronous algorithms are designed to run on asynchronous networks without any timing assumptions.

Classification

The consensus algorithms can be classified into two broad categories:

- Traditional—voting-based consensus
- Lottery-based—Nakamoto and post-Nakamoto consensus

Algorithms

CFT algorithms:

Paxos

Raft

BFT Algorithms

PBFT

IBFT

CFT Algorithms: Paxos

The Paxos protocol assumes an asynchronous message-passing network with less than **50% of crash faults.**

As usual, the critical properties of the Paxos consensus algorithm are safety and liveness.

Under **safety,** we have:

• **Agreement**, which specifies that no two different values are agreed on. In other words, no two different learners learn different values.

• **Validity**, which means that only the proposed values are decided. In other words, the values chosen or learned must have been proposed by a processor.

Paxos:

Under **liveness**, we have:

•**Termination,** which means that, eventually, the protocol is able to decide and terminate.

In other words, if a value has been chosen, then eventually learners will learn it.

Working of Paxos

A single process in a Paxos network can assume all **three roles.**

Processes can assume **different roles**, which are listed as follows:

- **Proposers**, elected leader(s) that can propose a new value to be decided.
- **Acceptors**, which participate in the protocol as a means to provide a majority decision.
- **Learners**, which are nodes that just observe the decision process and value.

Working of Paxos: step by step:

1. The proposer proposes a value by broadcasting a message, <prepare(n)>, to all acceptors.

2. Acceptors respond with an acknowledgment message if proposal n is the highest that the acceptor has responded to so far. The acknowledgment message <ack(n,v, s)> consists of three variables where n is the proposal number, v is the proposal value of the highest numbered proposal the acceptor has accepted so far, and s is the sequence number of the highest proposal accepted by the acceptor so far.

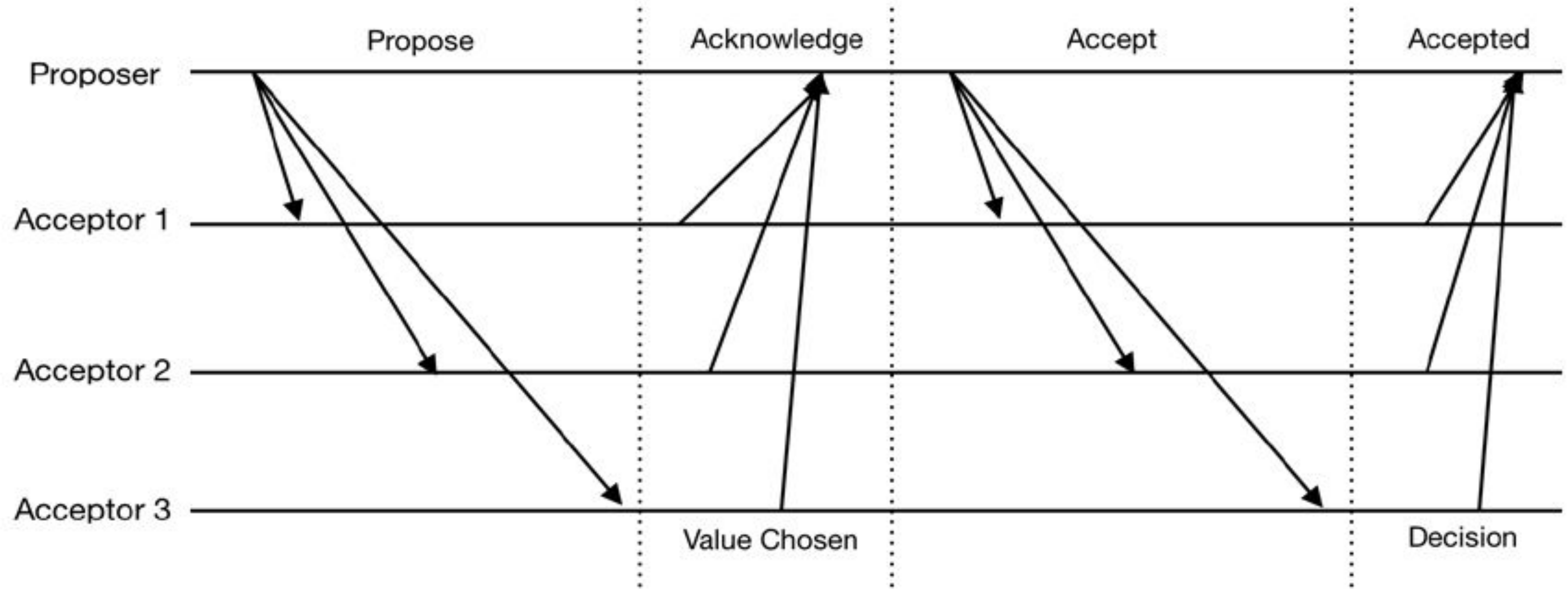
This is where acceptors agree to commit the proposed value. The proposer now waits to receive acknowledgment messages from the majority of the acceptors indicating the chosen value.

3. If the majority is received, the proposer sends out the "accept" message <accept(n, v)> to the acceptors.

4. If the majority of the acceptors accept the proposed value (now the "accept" message), then it is decided: that is, agreement is achieved.

5. Finally, in the learning phase, acceptors broadcast the "accepted" message <accepted(n,v)> to the proposer. This phase is necessary to disseminate which proposal has been finally accepted. The proposer then informs all other learners of the decided value. Alternatively, learners can learn the decided value via a message that contains the accepted value (decision value) multicast by acceptors.

Paxos:


Propose
Acknowledge
Accept
Accepted
Proposer
Acceptor 1
Acceptor 2
Acceptor 3
Value Chosen
Decision

How Paxos Works

RAFT

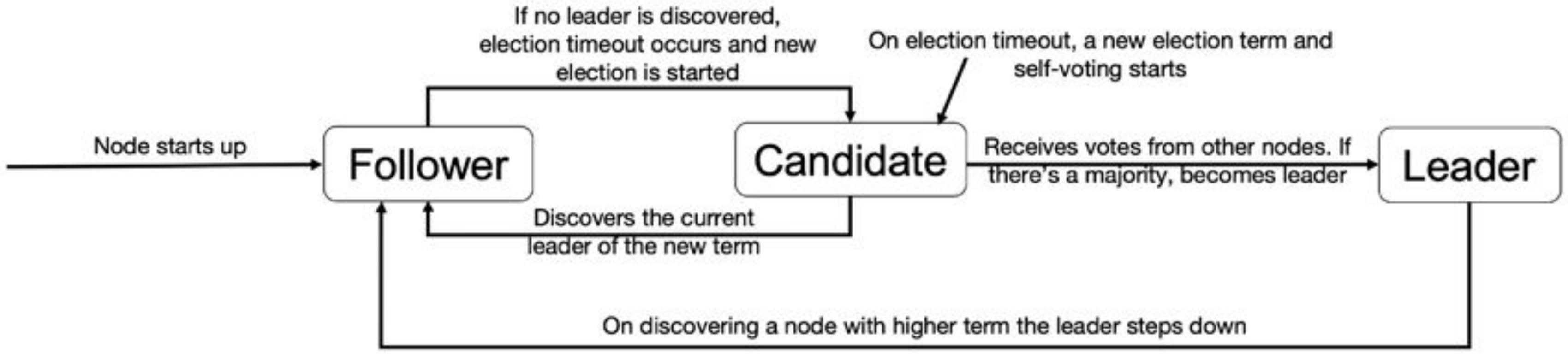
RAFT Consensus is a distributed consensus algorithm that is designed to manage replicated logs in a distributed system.

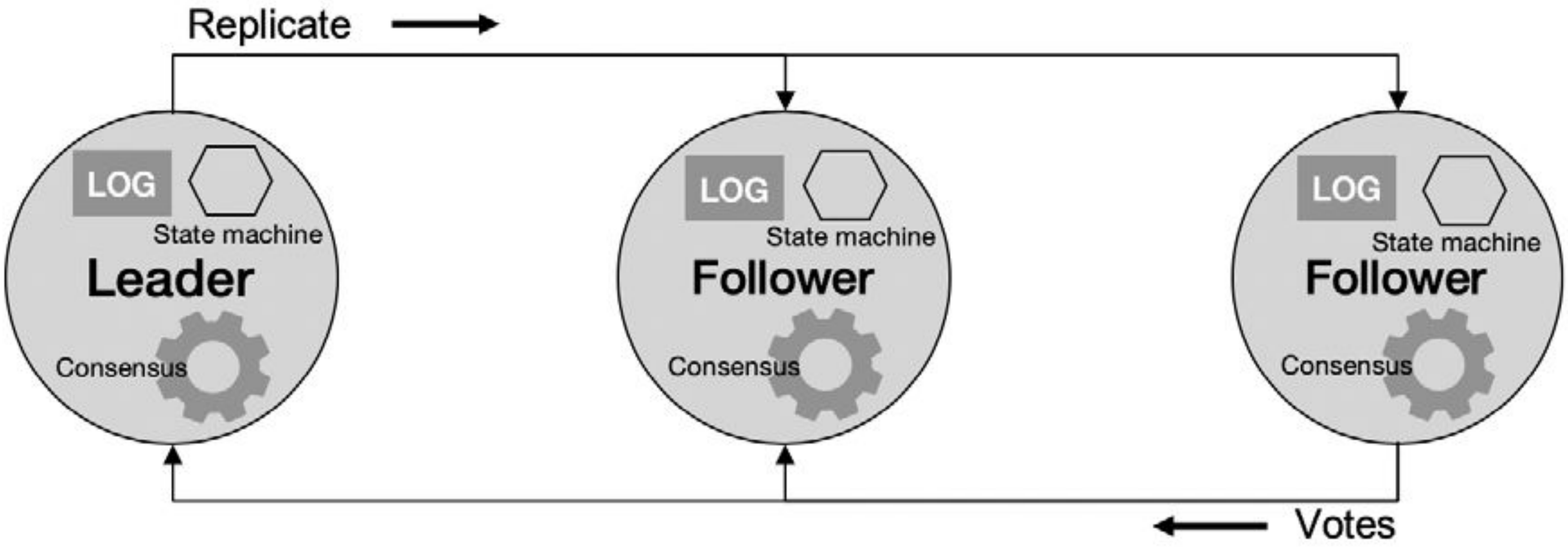
It was proposed by Diego Ongaro and John Ousterhout in 2014 as an alternative to the widely-used Paxos consensus algorithm.

The term "RAFT" is an acronym that stands for the three main components of the algorithm:

- Leader Election
- Log Replication
- Safety

Raft


If no leader is discovered, election timeout occurs and new election is started
On election timeout, a new election term and self-voting starts
Node starts up
Follower
Candidate
Receives votes from other nodes. If there's a majority, becomes leader
Leader
Discovers the current leader of the new term
On discovering a node with higher term the leader steps down


Replicate
LOG
State machine
Leader
Consensus
LOG
State machine
Follower
Consensus
LOG
State machine
Follower
Consensus
Votes

RAFT Process Flow

The RAFT consensus process :

1. Leader Election
2. Log Replication
3. Log Entry Commitment
4. Safety
5. Handling Node Failure

Overall, the RAFT consensus process is designed to be fault-tolerant and ensure that all nodes in the system are in agreement on the current state of the system. This can help to prevent errors and ensure that the system is reliable and consistent.

Nakamoto consensus

Nakamoto consensus, or PoW, was first introduced with Bitcoin in 2009.

At a fundamental level, the PoW mechanism is designed to mitigate Sybil attacks, which facilitates consensus and the security of the network.

How PoW works

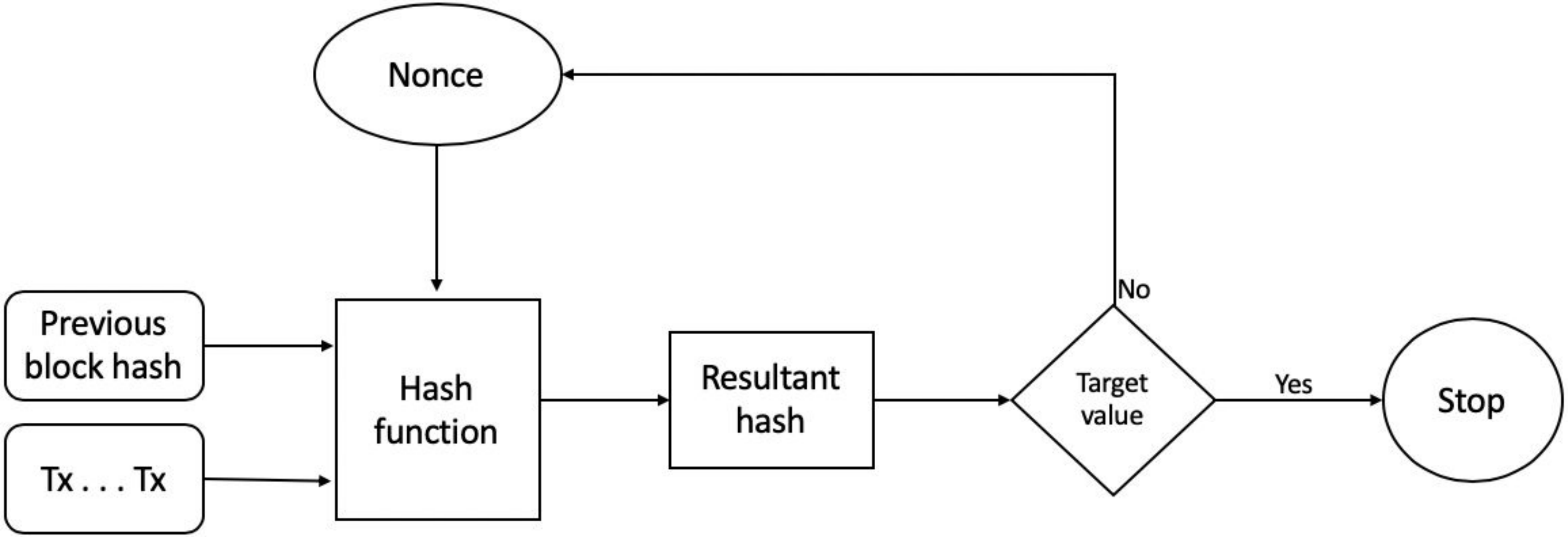
PoW makes use of hash puzzles.

A node proposes a block has to find a nonce such that H (nonce || previous hash || Tx ||Tx|| . . . ||Tx) < Threshold value.

How PoW works

- New transactions are broadcast to all nodes on the network.
- Each node collects the transactions into a candidate block.
- Miners propose new blocks.
- Miners concatenate and hash with the header of the previous block.
- The resultant hash is checked against the target value, that is, the network difficulty target value.
- If the resultant hash is less than the threshold value, then PoW is solved, otherwise, the nonce is incremented and the node tries again. This process continues until a resultant hash is found that is less than the threshold value.

Proof-of-Work (PoW)



Variants of PoW

CPU-bound PoW

Memory-bound PoW

PoW consumes tremendous amounts of energy

Proof-of-Stake (PoS)

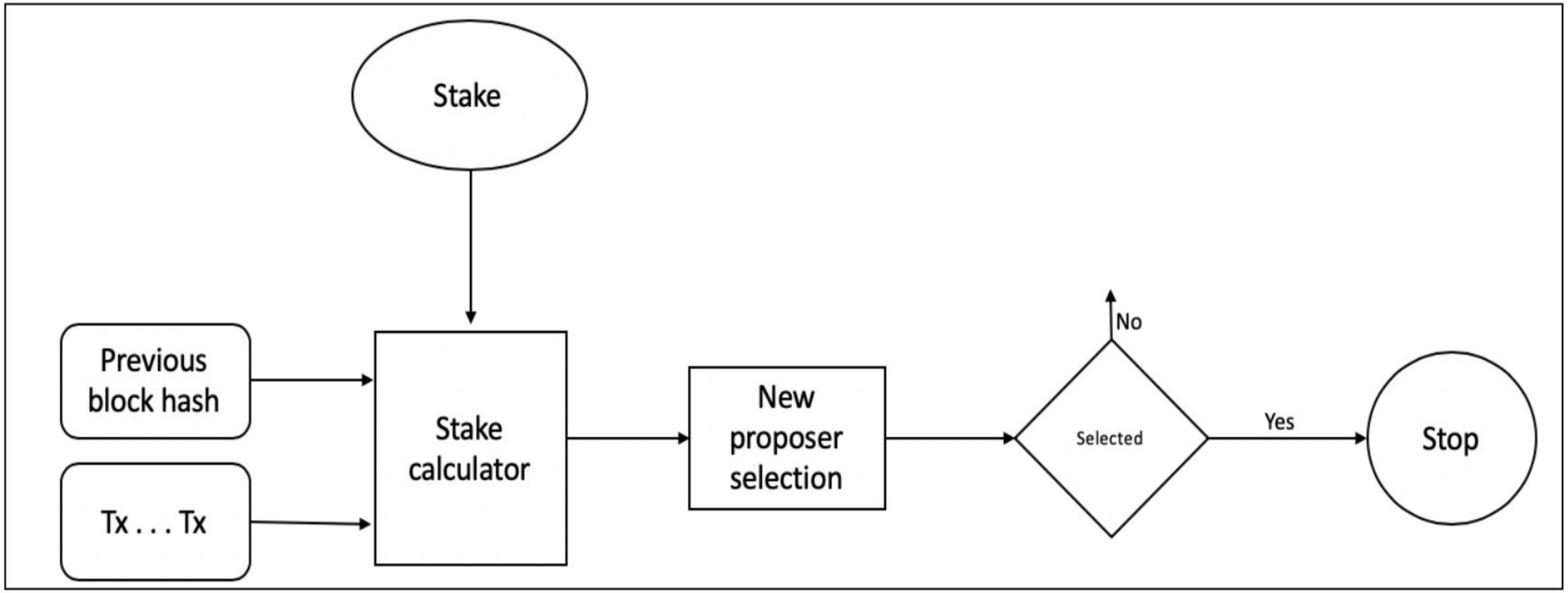
PoS is an energy-efficient alternative to the PoW algorithm, which consumes an enormous

amount of energy.

PoS was first used in Peercoin, and now, prominent blockchains such as EOS,NxT, Steem, and Tezos are using PoS algorithms. Ethereum, with its Serenity release, will soon transition to a PoS-based consensus mechanism.

The stake represents the number of coins (money) in the consensus protocol staked by a blockchain participant.

How Proof-of-Stake (PoS) works



Choosing an consensus algorithm

It depends on the several factors:

It is not only use case-dependent,but some trade-offs may also have to be made to create a system that meets all the requirements without compromising the core safety and liveness properties of the system.

These factors include, but are not limited to **finality, speed, performance, and scalability.**

BFT algorithms

Practical Byzantine Fault Tolerance

Istanbul Byzantine Fault Tolerance

PBFT- Practical Byzantine Fault Tolerance

Practical Byzantine Fault Tolerance (PBFT) was developed in 1999 by Miguel Castro and Barbara Liskov.

PBFT, as the name suggests, is a protocol developed to provide consensus in the presence of Byzantine faults.

PBFT- Practical Byzantine Fault Tolerance

PBFT comprises three sub-protocols called:

normal operation,

view change, and

checkpointing.

PBFT- Practical Byzantine Fault Tolerance

How the PBFT protocol works:

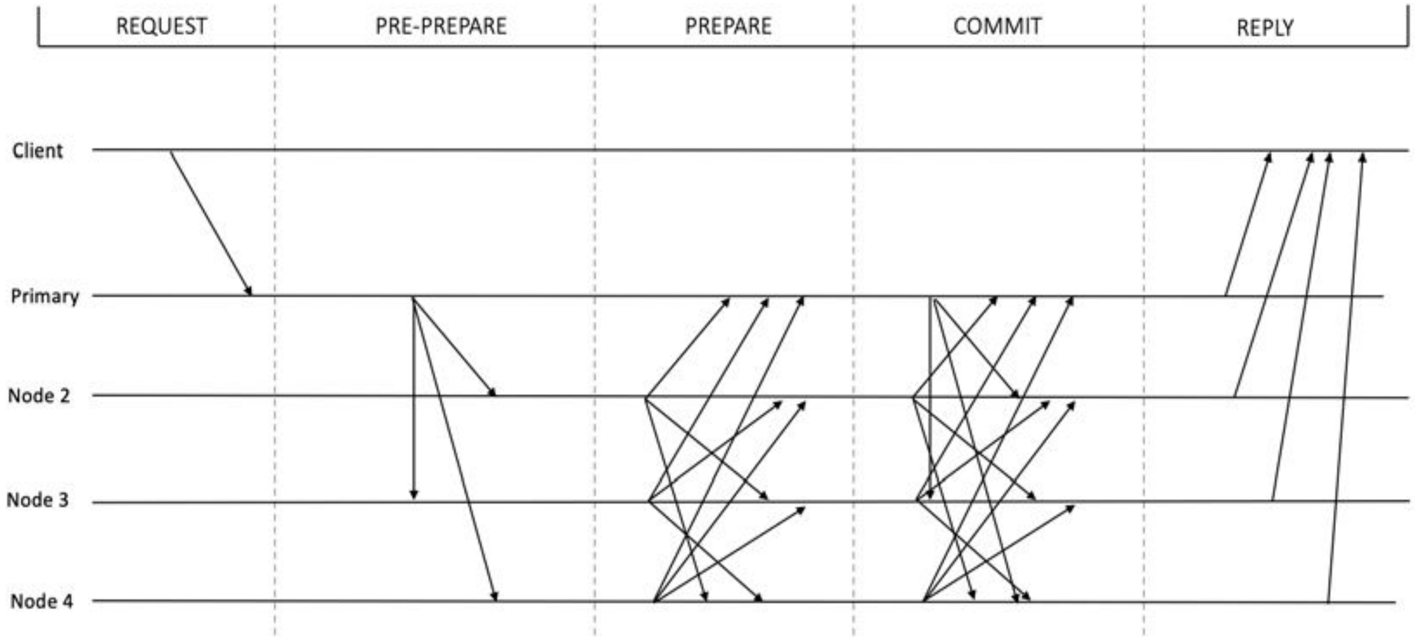
1. A client sends a request to invoke a service operation in the primary.

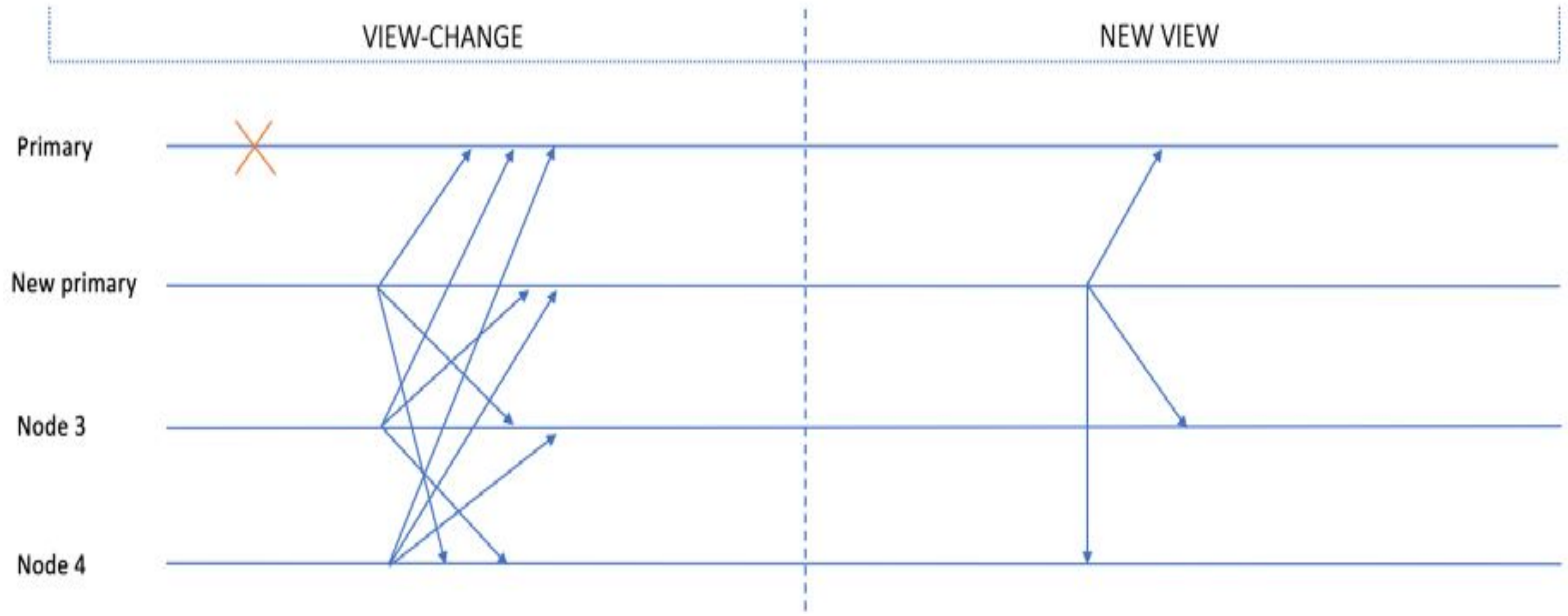
2. The primary multicasts the request to the backups.

3. Replicas execute the request and send a reply to the client.

4. The client waits for replies from different replicas with the same result; this is the result of the operation.

PBFT- Practical Byzantine Fault Tolerance




VIEW-CHANGE
NEW VIEW
Primary
New primary
Node 3
Node 4

Variants of PBFT

1. PBFT Hotstuff
2. PBFT-BigchainDB
3. Simplified PBFT
4. Tendermint

These variants of PBFT are designed to address different concerns or use cases, such as scalability, efficiency, or simplicity, while still providing the same level of fault tolerance as the original algorithm.

IBFT- Istanbul Byzantine Fault Tolerance

IBFT was developed by AMIS Technologies as a variant of PBFT suitable for blockchain networks.

It was presented in EIP 650 for the Ethereum blockchain.

IBFT- Istanbul Byzantine Fault Tolerance

How IBFT works: The model is composed of at least 3F+1 processes (standard BFT assumption), a partially synchronous message-passing network, and sound cryptography.

IBFT- Istanbul Byzantine Fault Tolerance

1. The protocol starts with a new round. In the new round, the selected proposer broadcasts a proposal (block) as a pre-prepare message.

2. The nodes that receive this pre-prepare message validate the message and accept it if it is a valid message. The nodes also then set their state to pre-prepared.

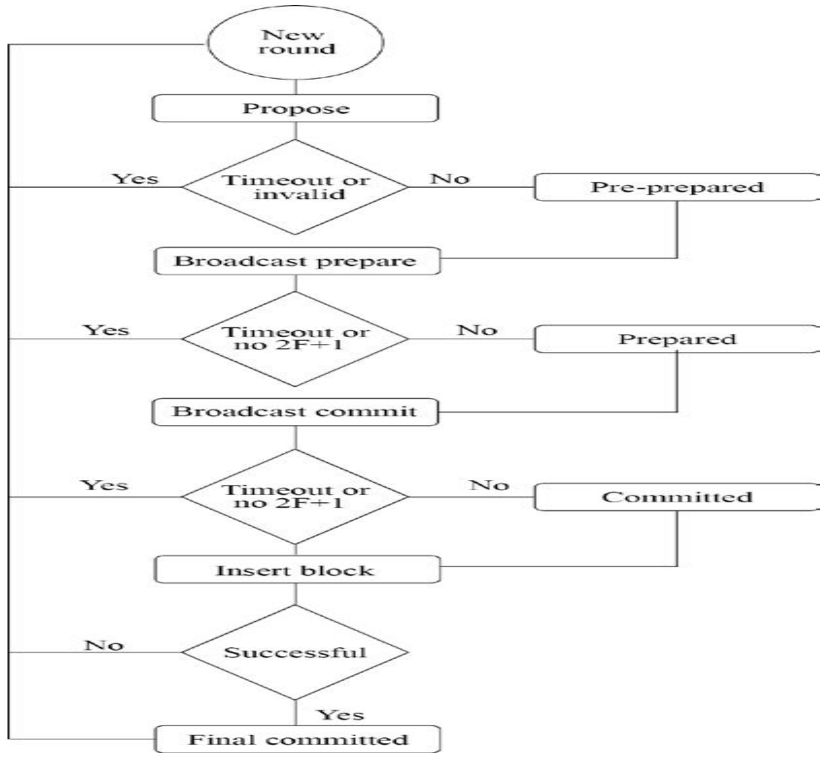
3. At this stage, if a timeout occurs, or a proposal is seen as invalid by the nodes, they will initiate a round change. The normal process then begins again with a proposer, proposing a block.

4. Nodes then broadcast the prepare message and wait for 2F+1 prepare messages to be received from other nodes. If the nodes do not receive 2F+1 messages in time, then they time out, and the round change process starts. The nodes then set their state to prepared after receiving 2F+1 messages from other nodes.

5. Finally, the nodes broadcast a commit message and wait for 2F+1 messages to arrive from other nodes. If they are received, then the state is set to committed, otherwise, timeout occurs and the round change process starts.

6. Once committed, block insertion is tried. If it succeeds, the protocol proceeds to the final committed state and, eventually, a new round starts. If insertion fails for some reason, the round change process triggers. Again, nodes wait for 2F+1 round change messages, and if the threshold of the messages is received, then round change occurs.

IBFT- Flow chart



Difference between PBFT and IBFT

• There is no distinctive concept of a client in IBFT. Instead, the proposer can be seen as a client, and in fact, all validators can be considered clients.

• There is a concept of dynamic validators, which is in contrast with the original PBFT, where the nodes are static. However, in IBFT, the validators can be voted in and out as required.

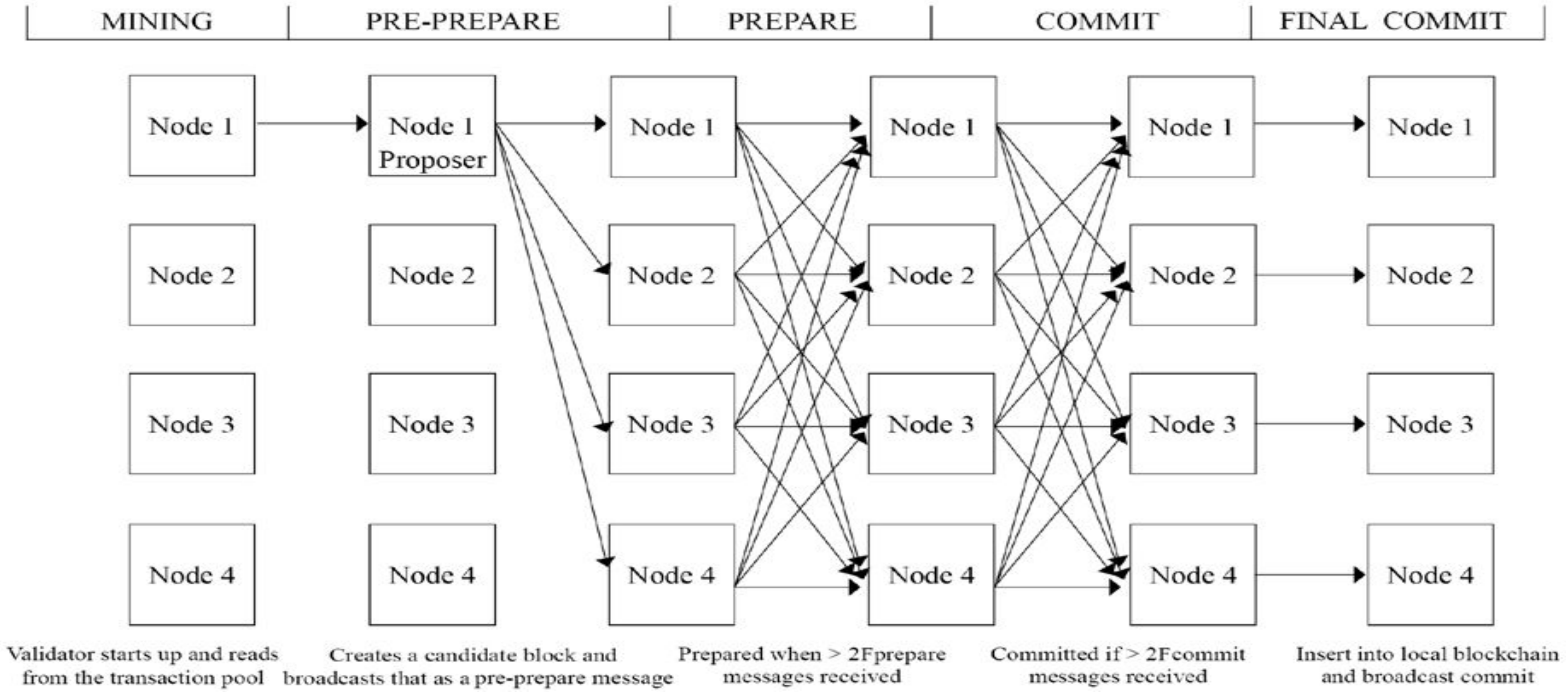
• There are two types of nodes in an IBFT network, nodes and validators. Nodes are synchronized with the blockchain without participating in the IBFT consensus process. In contrast, validators are the nodes that participate in the IBFT consensus process.

• IBFT relies on a more straightforward structure of view-change (round change) messages as compared to PBFT.

• In contrast with PBFT, in IBFT there is no concrete concept of checkpoints. However, each block can be considered an indicator of the progress so far (the chain height).

• There is no concept of garbage collection in IBFT.

IBFT Flow


MINING
PRE-PREPARE
PREPARE
COMMIT
FINAL COMMIT
Node 1
Node 2
Node 3
Node 4
Node 1 Proposer
Validator starts up and reads from the transaction pool
Creates a candidate block and broadcasts that as a pre-prepare message
Prepared when > 2Fprepare messages received
Committed if > 2Fcommit messages received
Insert into local blockchain and broadcast commit

Conclusion

- ❖ We discussed some of the most prominent protocols in blockchain and traditional distributed system consensus.
- ❖ We covered several algorithms, including **Proof of Work, Proof of Stake, traditional BFT protocols.**
- ❖ Distributed consensus is a very ripe area of research and academics and industry researchers are participating in this exciting subject.
- ❖ It is also a deep and vast area of research, therefore, it is impossible to cover every protocol and all dimensions of this gripping discipline in a single topic..
- ❖ The protocols and ideas presented in this topic provide **solid ground for further research and more in-depth exploration.**

References

[1] Chapter 5-Consensus Algorithms, Mastering Blockchain 4 edition, Imran Bashir, Packt Publications

[2] Chapter-3 From Byzantine Consensus to Blockchain Consensus by Miguel Correia ,Essentials of Blockchain Technology by Kuan-Ching Li et al. Pages: 41-72

[3] Chapter-15 from DISTRIBUTED SYSTEMS Concepts and Design Fifth Edition by George Coulouris et al, Addison Wesley Pages: 645-687