3.2 Types of Smart Contracts

Following are the categories of smart contracts using natural language for encoding.

- The contracts which are entirely in code.
- Contracts embedded in code using distinct version of the natural language.
- "Split" natural language contract with encoded performance.
- The contracts with encoded payment scheme. Encoding is in natural language. Here, the automation is offered.

Smart contracts can be used in many industries with several benefits. The different domains where they are used are : banking, health, finance, insurance, energy, e-governance, media, telecommunication, education, manufacturing and many other domains. The smart contracts can be classified into 4 types on the basis of applications.

1. Smart legal Contracts	2. Distributed Applications (DApps)
3. Decentralized Autonomous Organizations (DAO)	4. Smart Contracting Devices (SCD)

1. Smart Legal Contracts

- These are the most common type of smart contracts. They are with legal contract templates. They involve legal requirements such as mutual agreement which expresses a valid offer and acceptance, acceptable consideration, capacity and legality. These requirements are similar to their traditional counterpart.
- These are created to hold concerned parties responsible for fulfilling their end of an agreement. When set up correctly, a smart contract is lawfully enforceable and requires the parties to fulfil their duties. If parties fail to fulfill obligations in the contract then a legal action is triggered by the smart contract automatically against the party that is in breach.
- The laws and its contexts are different in different countries. Hence, they lack adequate support for smart and automated contracts on the blockchain and their legal status is unclear. However, once the laws are created, smart contracts can be created to ease processes which presently involve strict regulatory oversight such as transactions in the financial and real estate market, government subsidies, international trade etc.

2. Decentralized Applications (DApps)

- Smart contracts and DApps are closely related with each other. DApps interacts with the blockchain and controls the state of all network members. Smart contracts demonstrate the core logic of a decentralized app. DApps have unlimited number of members or participants in market.
- The user interface (UI) of DApps is similar to a typical website or mobile application. DApps front end represent the user's interaction, and its back end is implemented for all business logic. The business logic of DApps is depicted by one or more smart contracts communicating with the underlying technology of blockchain.

3. Decentralized Autonomous Organizations (DAO)

- DAO are the communities that exist on the blockchain. These communities can be defined by a set of stipulatory rules which are implemented via smart contracts. Every participant and their actions are bound by the community's rules with the task of imposing these rules. These rules are constituted from many smart contracts and collaborate to watch over activities in the community.
- The Genesis DAO was created by Ethereum members in May 2016. The main aim of community was for crowdfunding and as a venture capital platform. In an unexpectedly short time, they succeeded to raise an astonishing $150 million. However, hackers got some loopholes in the system and succeeded to steal about $50 million dollars' worth of Ethers from the crowdfund investors. The hack and its consequences resulted in a fork of the Ethereum blockchain into Ethereum and Ethereum Classic.
- So, DAO is the organization that is represented by logic implemented in transparent manner that is controlled by shareholder community without any central authority. DAOs are sophisticated form of smart contracts. There are some activities that needs manual intervention to execute certain tasks which cannot be executed automatically. These activities are :
  - **Tokens :** Tokens are used to reward certain activities which are funded directly upon creation of organizations.
  - **Autonomous :** The DAO code after deployment, cannot be controlled by its creator or any outsiders. Hence, it is transparent and incorruptible.

- **Agreement :** DAO's stakeholder should agree to withdraw or move funds. The number of stakeholders to agree is already mentioned in DAO code.
- **Contractor :** To elect contractor for carrying out product and hardware building, and coding as well, voting of token holders is necessary. This is because, the DAO cannot perform these tasks and contractor is appointed to achieve the gaols defined.
- **Proposals :** Used only for decision making in DAO. DAO need monetary deposit to avoid network overloading because of many proposals from many people. This discourages people from spamming the network.
- **Voting :** Once the proposal is submitted, voting takes place. DAO permits participants to exchange economic value with any person in global network without any intermediatory.

4. Smart Contracting Devices (SCD)

- Internet of things (IoT) in combination with the blockchain, talks about Application logic contacts (ALC) which contain an application-based code that remains in-step with other blockchain contracts. They allow communication across various devices and enable the merging of the Internet of Things (IoT) with blockchain technology.
- ALCs are an essential piece of multi-function smart contract and typically work under a managing program. Such smart contracts comprise application specific code that function in combination with other smart contracts and programs on the blockchain. They assist in communicating with and validating communication between IoT devices.

3.6.5 Arrays : Fixed and Dynamic Arrays

Array data structure stores a fixed-size sequential collection of elements of the same type. It is more useful to think array as a collection of same type variables. There is no need to declare individual variables, for example, number0, number1, ..., and number99. Instead of this, it is sufficient to declare one variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A particular element in an array is directly accessed by an index. In Solidity, two types of arrays are supported. One is compile-time fixed size and another is of dynamic size. In case of storage array, different types of elements can be stored. For memory array, element type cannot be mapping and in case it is to be used as function parameter then element type should be an ABI type. All arrays comprise contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element of an array.

Fixed Size Array

In Solidity, to declare fixed size array, programmer has to specify the type of the elements and the number of elements in an array.

```
type arrayName [ arraySize ];
```

This is a single-dimension array. The arraySize must be an integer constant greater than zero and type is any valid Solidity data type. For example, to declare a 20-element array called account of type uint, use this statement –

```
uint account[20];
```

Following program demonstrate the use of fixed size array.

```
// Solidity program to demonstrate creating a fixed-size array
pragma solidity ^0.8.2;
// Creating a contract
```

```
contract Types {
  // Declaring state variables of type array
  uint[6] data1;
  // Defining function to add values to an array
  function array_example() public returns (
  int[5] memory, uint[6] memory){
    int[5] memory data = [int(50), -63, 77, -28, 90];
    data1 = [uint(10), 20, 30, 40, 50, 60];
    return (data, data1);
}
```

Output:

```
"0": "int256[5] : 50, -63, 77, -28, 90",
"1": "uint256[6] : 10, 20, 30, 40, 50, 60"
```

Dynamic Arrays

- In Solidity, to declare an array of dynamic size, the programmer has to specify the type of the elements as follows.

```
type[] arrayName;
```

- Dynamic memory arrays are created using new keyword.

```
uint size = 3;
uint account[] = new uint[](size);
```

- Following single statement initializes Solidity array elements.

```
uint account[3] = [10, 20, 30];
```

- Following is an example to assign a single element of the array :

```
account[2] = 5;
```

- The above statement assigns element number 3rd in the array a value of 5.

```
uint salary = account[2];
```

- The above statement assigns the value of third element in array to salary variable.

Example:

```
// Solidity program to demonstrate creating a dynamic array
pragma solidity ^0.8.2;
// Creating a contract
contract Types {
  // Declaring state variable of type array. One is fixed-size and the other is dynamic array
  uint[] data
  = [10, 20, 30, 40, 50];
  int[] data1;
```

```
  // Defining function to assign values to dynamic array
  function dynamic_array() public returns(
    uint[] memory, int[] memory){
      data1 = [int(-60), 70, -80, 90, -100, -120, 140];
      return (data, data1);
  }
}
```

Output :

```
"0": " int256[] : 10, 20, 30, 40, 50",
"1": "int256[] : -60, 70, -80, 90, -100, -120, 140"
```

Members

- **Length :** It returns the size of the array. It can be can be also used to change the size of dynamic array.
- **Push :** push is used to append an element to a dynamic storage array at the end. It returns the new length of the array.
- Declaration, assignment and accessing the arrays is demonstrated in following example.

Example

```
pragma solidity ^0.8.2;
contract test {
  function testArray() public pure{
    uint len = 7;
    //dynamic array
    uint[] memory a = new uint[](7);
    //bytes is same as byte[]
    bytes memory b = new bytes(len);
    assert(a.length == 7);
    assert(b.length == len);
    //access array variable
    a[6] = 8;
//test array variable
    assert(a[6] == 8);
    //static array
    uint[3] memory c = [uint(1) , 2, 3];
    assert(c.length == 3);
  }
}
```

4.2 Ethereum

- Ethereum, founded in 2015 by Vitalik Buterin, is a unique blockchain with a unique token known as Ether.
- While the ownership of digital currency is verified by Bitcoin via the blockchain, the Ethereum platform enables developers to create fully distributed apps and does away with the "centralized" client-server internet architecture.
- The key feature of Ethereum is its programmability, which enables the development of applications.
- Ethereum is a blockchain-based open software platform that allows developers to create and deploy decentralized applications.
- It is more than simply a platform for cryptocurrencies; it has the ability to perform tasks and develop distributed applications.
- Ethereum is a DIY platform for decentralized applications known as Dapps (decentralized apps). It is the global infrastructure for operating Dapps.
- The Ethereum network is completely decentralized since it is run by thousands of separate computers.
- Once the program is deployed to the Ethereum network, these computers also identified as nodes will, be assured that it execute as written.
- It was the first technology to implement a smart contract. It permits participants to run decentralized blockchain applications, which are called Dapps.
- Ethereum blockchain is an immutable public database that stores a record of every transactions that has ever taken place.
- Ethereum is a Turing complete system.
- In its virtual machine, Ethereum can run programmes and read and write data to memory. Any programme that a Turing complete machine can compute may be done by this machine.
- Ethereum's major point of differentiation is how it mixes blockchain technology and computing power. When the programmability feature is paired with all of the blockchain's distinguishing characteristics, Ethereum becomes a practical technology.
- It is referred to as ETH and has its own currency called Ether.

What is Ether?

- Ether (ETH) is the native cryptocurrency of Ethereum.
- It is the second largest cryptocurrency around.
- The goal of ether is to create a market for computing.
- The Ethereum network and smart contracts are powered by ether, which is used as fuel (or gas).
- The Ethereum network's users utilize ether to pay for services rendered by the network.
- Ethereum allows us to send ether which is equivalent to Bitcoin.

4.3 Ethereum Components

Ethereum's components are as follows :

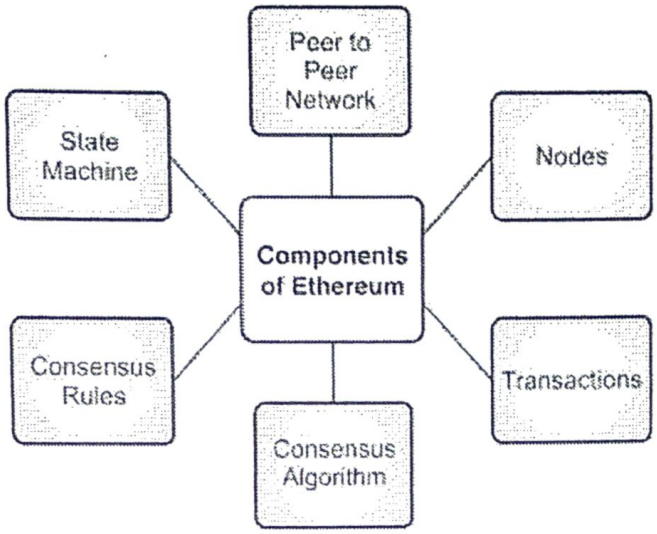


Fig. 4.3.1 : Components of Ethereum

- Ethereum is a peer-to-peer network of computers named Ethereum nodes or clients, each of which runs a compatible software. Anyone can run an Ethereum node and there are no special prerequisites or requirements.
- At the moment, there are thousands of Ethereum nodes that communicate with each other. In Fig. 4.3.2 and 4.3.3 shows, statistics about the Ethereum mainnet.

Fig. 4.3.2 : Ethereum latest blocks and transaction

Fig. 4.3.3 : Ethereum Mainnet Statistics

- There are more ways for a user or application to communicate with the network. Web 3.js is a collection of libraries that allows web applications to interact with an Ethereum node. using HTTP , IPC or WebSocket. A normal website cannot communicate with a Ethereum blockchain so we will need Web3.js
- Now let us discuss components of Ethereum.

1) **Peer-To-Peer (P2P) network :** Every computer or node on the Ethereum blockchain is linked to every other node. Ethereum operates on the Ethereum main network, which is accessible through TCP port 30303, and employs the ÐΞVp2p protocol.

2) **Consensus rules :** The set of rules used to determine if a transaction is legitimate

3) **Transactions :** On a Ethereum network, transactions are the messages that are sent and received. A value, a sender, a receiver, and a payload of data are all components of a transaction.

4) **State machine :** The state machine known as the Ethereum Virtual Machine, or EVM for short, processes Ethereum transactions and state changes. The Ethereum virtual machine executes the program in a machine language (bytecodes). These programmes are created in a high level language, such as Solidity, and are then converted into bytecode via a compiler. The program written in a high level language is called a smart contract.

5) **Consensus algorithm :** The distributed Ethereum network's consensus algorithm is the process used to reach consensus on the longest chain. Proof of work and proof of stake are the two consensus methods for Ethereum that are most often utilized.

... Data that can be sent. This field is op

4.6.2 Accounts

- A crucial part of the Ethereum system is an account.
- It is the primary component and has an ether (ETH) balance.
- A user's account can transmit transactions through the Ethereum network.
- Transactions are produced by the interactions between several accounts and are kept in the distributed database.
- Sender, recipient, and contract accounts are all involved in every transaction.
- An account can be managed by a user or by a programme running on the Ethereum network.
- Ethereum accounts are divided into two types:
  - Externally owned accounts (EOA)
  - Contract accounts

Externally owned accounts (EOA) :

- EOA is controlled via private keys from a user and the contract accounts are controlled by their contract code.
- The term "Externally Owned Account" is referred to as EOA. This is the type of account that is owned by people on the Ethereum network.
- A public-private key pair is produced for an account once the user creates one on Ethereum. The name or any other personal information of the user does not identify the account.

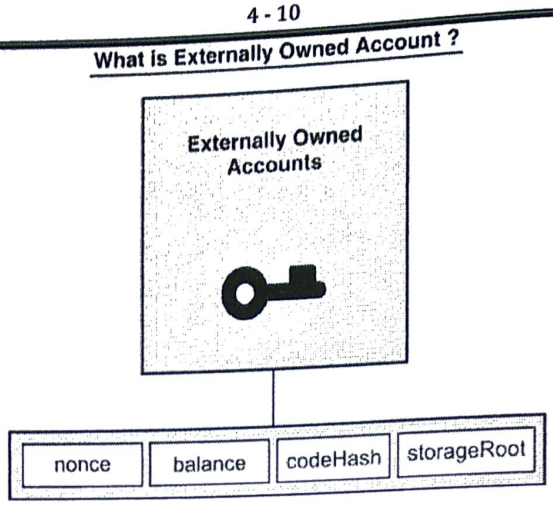


Fig. 4.6.3 : Externally Owned Account

- Externally owned accounts do not contain code; it has balance in ether and can send messages by generating and signing a transaction.
- It is used for holding, sending and receiving ETH and for interacting with smart contracts like deployment and calling functions etc.
- Access and control over one's contracts and assets are provided through the EOA private key. The user maintains security for the private key.
- Although, as the name indicates, the account's public key is accessible. The identity of the account is represented by this key. Through the use of a one-way cryptographic function, the public key is created from the private key.
- If you register for an Ethereum account, you will keep the private key to yourself while sharing the public key.
- Public keys are used for account-to-account transactions. An Ethereum account's public key is what creates the account's address.
- Other externally owned accounts and contract accounts can communicate with this kind of account.

Contract Accounts :

- A contract account has a code for a smart contract.
- This account doesn't have a public or private key; the public address of the contract account is used to identify it. Private keys are not owned by this account.

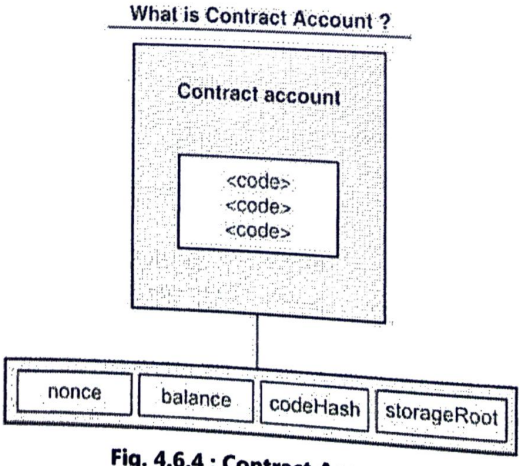


Fig. 4.6.4 : Contract Account

- Functions to run and state variables are present in this code.
- This account is managed by the smart contract's logic. The owner of the contract's account is the smart contract that is implemented on the Ethereum blockchain.
- The code for the contract is executed on the Ethereum virtual machine when an externally owned account begins a transaction with a contract account.
- When a contract account receives a message, it activates its code. This might imply that it reads and writes to its storage, transmits messages, or generates new contracts.
- In addition to ether, the transaction also contains information on which functions should be executed and what arguments should be sent to those functions.
- There are four fields in the account state: nonce, balance, storageroot, and codehash.
  - **Nonce -** The nonce serves as a counter, ensuring that each transaction is only completed once.
  - **The current ether balance -** The amount of ether the user presently holds. (in wei)
  - **Codehash or Contract code -** If the account is EOA, this field is empty. This field includes the code that will be done when a message is sent to the account if the account is a contract account. This is immutable EVM bytrecode.
  - **Storage -** The storage is initially empty, but it may be filled and read from through transactions and contract codes. (only for contract account)

Externally Owned Account	Contract Account
Owned by some external entity (person, corporation etc)	Owned by contract
This account does not have a code.	A code is assigned to each contract account.
A transaction can be initiated by an externally owned account with another externally owned account or a contract account.	But contract account cannot initiate transactions.
Externally owned accounts have a private key associated with them	Contract accounts do not have a private key
It is free to open an externally owned account.	It costs money to set up a contract account.
The only type of transaction that may occur between two EOAs is the transfer of ether.	Any type of transaction with a contract account is permitted, including deploying or invoking the functionality of other contracts.

4.8 Comparison between Bitcoin and Ethereum

Bitcoin	Ethereum
In Bitcoin SHA-256 is used in the algorithm of the Blockchain	In Ethereum, Keccak-256 or Keccak-512 are used. Keccak is a variant of SHA3.
Assets : Bitcoin	Assets : Ether
Bitcoin is exclusively used as an online currency.	Ethereum is often used as an online currency, however its blockchain is also utilized for decentralized applications such as voting systems, file storage, and so on.
Data related to Bitcoin network transactions is frequently utilized simply to keep track of transactions.	On the Ethereum network, for example, transactions may include executable code.

Bitcoin	Ethereum
Simple and robust	Complex and feature-rich
Bitcoin consumes much more energy than Ethereum.	Ethereum is certainly consuming an increasing amount of energy, but this will likely decrease significantly after it has transitioned to Proof-of-Stake.
Stack based primitive scripting language, not Turing-complete	Turing-complete scripting language
Key lengths Public key = 512 bits (compressed: 257 bits) Private key = 256 bits	Key lengths Public key = 512 bits Private key = 256 bits Address = 160 bits
Mining Proof-of-work	Mining Proof-of-Work to Proof-of-Stake
Signature size 512 bits (64 bytes)	Signature size 520 bits (65 bytes)
Energy consumption ≈ 3, 2 million households	Energy consumption ≈ 1, 1 million households
Transaction time Bitcoin is known for its slow and expensive transactions. It takes around 10 mins to complete a Bitcoin transactions.	Transaction time Ethereum transaction takes only 12 seconds.
Block time 10 minutes	Block time 12 seconds
Mining reward 12,5 Bitcoin per block	Mining reward 3 Ether plus some other rewards
Consensus time ≈ 1 hour	Consensus time ≈ 3 minutes

4.9 Types of test-networks used in Ethereum

- A version of a project is deployed to an Ethereum Test Network ("testnet"), which replicates Ethereum, before it is launched on the Ethereum blockchain (or before modifications are made to the blockchain itself).
- This allows developers and the community to test the project before actual assets are engaged.
- On a testnet, ether and tokens are simple to get and have no real-world value.
- There are presently three testnets in use, and each performs identically to the main blockchain where you're real Ether and tokens reside.

Ropsten

- The Ropsten test network is an Ethereum Proof-of-Work testnet. One can mine on the Ropsten network to obtain ETH.

Kovan

- The Parity team created the Kovan test network as a Proof-of-Authority testnet for Ethereum.

Rinkeby

- The Rinkeby test network is an Ethereum Proof-of-Authority testnet that was founded by the Geth team. At Rinkeby, one may obtain ETH by requesting it through a faucet.

Making a connection to a testnet

- Ethereum addresses and private keys that operate on Ethereum work on each testnet.
- Create a new wallet specifically for use on testnets.

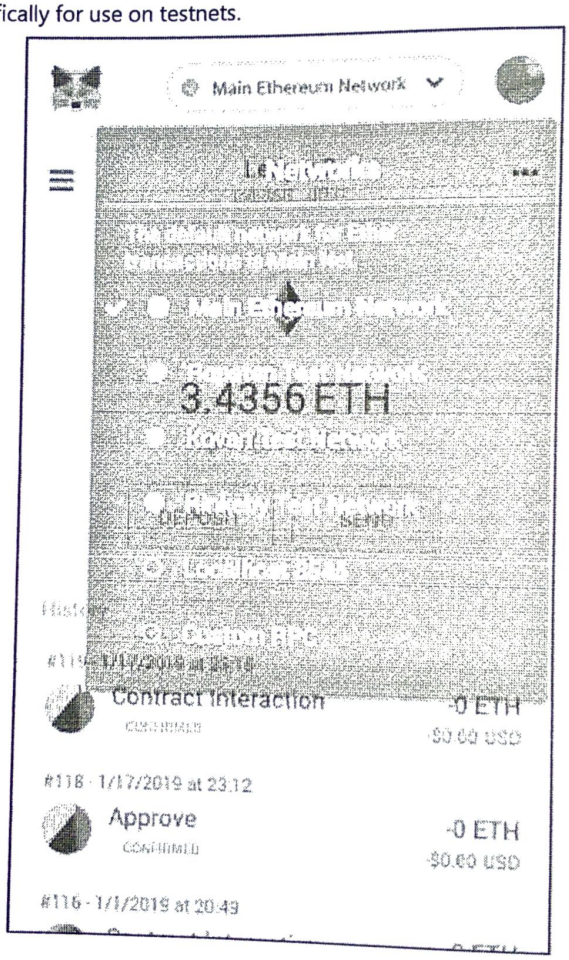


Fig. 4.9.1 : Metamask Plugin

- Using MetaMask to transmit Ether and tokens on a testnet is simple; simply choose an Ethereum network in the top-left corner of MetaMask.
- When you switch from the Main Ethereum Network to Rinkeby or another testnet, your balance and transaction history should change to reflect the network you've chosen.
- When you make a transaction in MetaMask, it will now be sent to the network you specify.

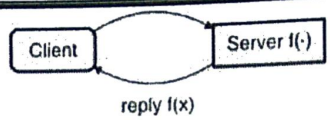


Fig. 5.5.1

- But there can be a single point failure.
- Here comes the need of state machine replication.

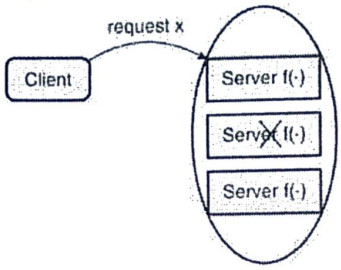


Fig. 5.5.2

- State machine replication gives safety and liveness by creating replicas of state of servers.
- It is an interactive protocol among servers.
- Replicas maintain the same state
  - replicas start in the same state.
  - Replicas execute operations in the same order.
- Replicas send replies to clients.
- Clients vote on replica replies.
- **The replication of state machines assists us in reaching a consensus in a permission model.**
- A smart contract does not have to be executed on each node.
- Instead, the chosen subset of the contract executor executes it and propagates it with further nodes to make sure that all nodes in the network are informed of the contract's state consistently and in real time.
- Consensus is ensured in a permission blockchain context via the distributed state machine replication technique.

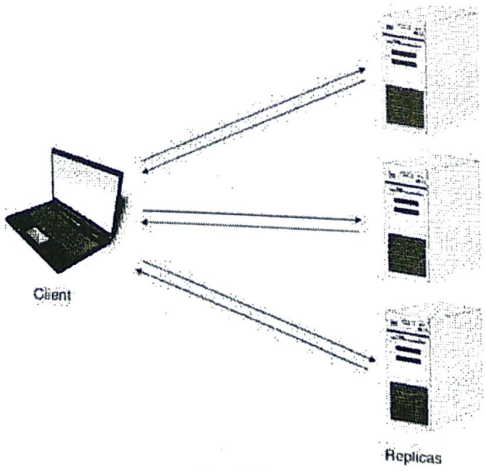


Fig. 5.5.3

- What is a state machine ?
  - A state machine executes a sequence of commands.
  - Transforms its state and may produce some output.
- Outputs of the state machines are solely determined by the initial state and by sequence of commands that it has executed.

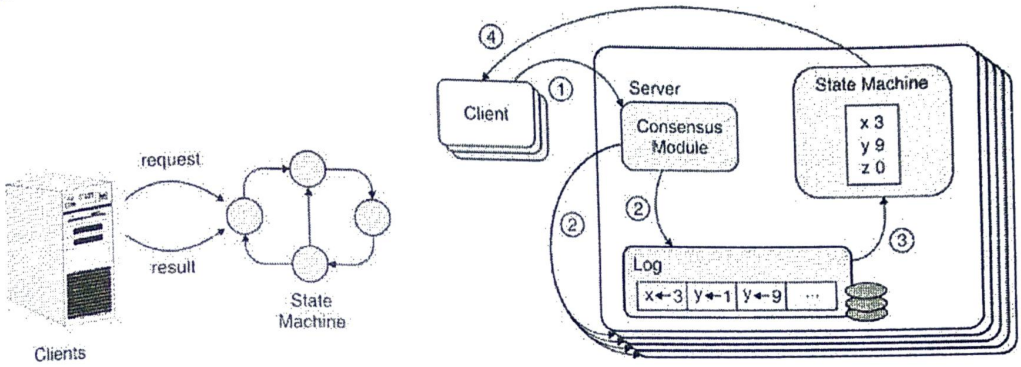


Fig. 5.5.4

- Typically, replicated logs are used to create replicated state machines.
- Upon receiving commands from clients, the server's consensus module adds them to its log..
- Whenever commands have been correctly replicated, the state machine on each server executes them in log order.
- Let us look at a scenario of a replicated state machine.
- We have a number of clients, and services are replicated into three machines. So the replication degree is three.
- We have a consensus module, and when a client wants to issue a command, it sends a command to one of the state machines. The Consensus Module makes an attempt to add these commands to a log.

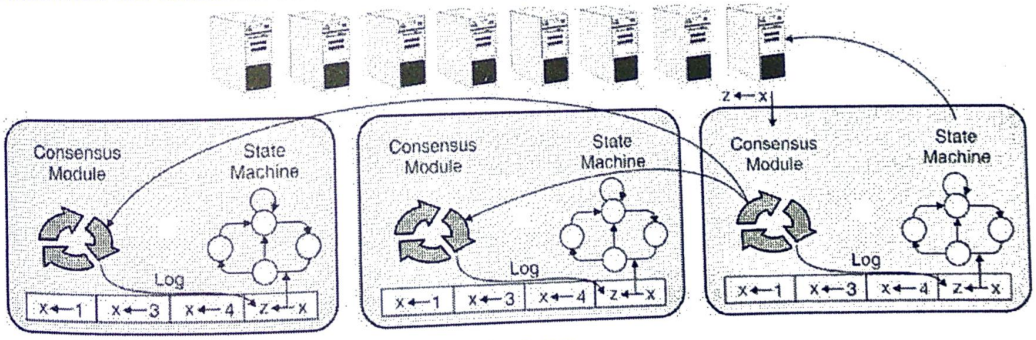


Fig. 5.5.5

- Replicated log ensures state machines execute the same commands in the same order.
- Consensus module guarantees agreement on command sequence in the replicated log.

Summary :

a) Implement service as a deterministic state machine

b) Replicate

c) Provide all replicas with the same input

5.6 Consensus Algorithms for Private Blockchain –PAXOS and RAFT

5.6.1 PAXOS Consensus Algorithm

- L. Lamport introduced it as the first consensus algorithm in 1989.
- The goal was to select a single value under the network or crash problems. Paxos lets all nodes agree on the same value despite node failures, network failures and delays.
- By delivering them to the other nodes, certain nodes offer values. The acceptance or rejection of those values must be decided by each node.
- But concurrent processes and uncertainty of timing, order of events and inputs can be a problem
- And also failure and recovery of machines or processors, of communication channels will be a major concern.

Consensus Problem :

- Consensus is impossible to solve in asynchronous systems means it is impossible to distinguish a failed process from one that is just very very slow. Hence the rest of the alive processes may stay unsure when it comes to deciding.
- But consensus is important since it maps to many important distributed computing problems. So can we solve the consensus problem ?
- Paxos does solve the consensus problem. It is the most popular consensus solving algorithm.
- The only entirely safe and mostly live agreement protocol known.
- Paxos is a family of distributed algorithms used to reach consensus.
- We know consensus is agreeing on one result. Once a majority agrees on a proposal, that is the consensus. The reached consensus can be eventually known by everyone.
- Paxos provides safety and eventual liveness.
  - Safety- consensus is not violated. Only the value that has been proposed may be chosen. And also no two correct nodes choose different values.
  - Eventual liveness - If things go well sometime in the future, there is a good chance consensus will be reached. But there is no guarantee.
- Paxos General approach
  - A node or more nodes select a leader.
  - An idea is put out by the leader, who then seeks support from others.
  - Leader either declares the outcome or tries again.

Paxos : Types of nodes

- Paxos defines three roles: proposers, acceptors, and learners.
- Proposer, who proposes values to reach consensus.

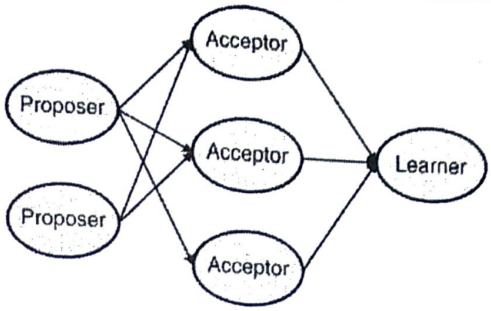


Fig. 5.6.1

- Acceptors, who contribute to reaching the consensus itself; they form the consensus and accept the values. The acceptors have the option of accepting or rejecting a proposal whenever they hear it from the proposer.
- Learners, who learn the agreed upon value. This learner will identify which value has been selected by each acceptor and agree to accept it as a group.
- In practice, Paxos nodes can take multiple roles, even all of them. So a single node, a single server can send proposals to other nodes, they can contribute to reaching consensus, and they learn the final agreed upon value so that you can query them for it later.
- Example, we are at the college, and following courses, we plan to hang out as a group. Pizzahut and Starbucks are our two options for meeting up with classmates after courses. So, depending on the group vote, we can either go to starbucks or pizza hut after classes, but everyone will go there. There is no centralized leader in this situation. The only method to make a group choice is for a couple of them; in this situation, the maximum number may be the two students to present an alternative i.e Starbucks or Subway. The consensus will ultimately be determined by how many people accept or reject that suggested choice, with the outcome being determined by the final majority count.
- Paxos nodes must know how many acceptors a majority is. Paxos nodes must be persistent; they can't forget what they accepted, even though the communication channel may be faulty.
- A Paxos run aims at reaching a single consensus. Once a consensus is reached, it cannot progress to another consensus. In order to reach another consensus, a different Paxos run must happen.

Paxos Algorithm

- We have a Proposer and three acceptors, so the majority of the acceptors would be two. Each node runs as a proposer, acceptor, and learner.
- Paxos has rounds and each round has a unique ballot ID.
- These rounds are asynchronous. If anybody is in round 1, and hear a message from round 2, abort everything and go to round 2.
- Each round has 3 phases

  Phase 1 : A leader is elected (election)

  Phase 2 : Leader proposes a value (bill), processes acknowledgements

  Phase 3 : Leaders multicast final value (based on consensus)

Phase 1 :

- Proposer proposes a value and solicits acceptance from acceptors. A specific value is what the proposer intends to propose. It notifies all Acceptors of the proposal number n, which is PREPARE ID. The proposal number must be unique.
- Acceptors respond to highest PREPARE Id. If an acceptor receives a prepare request with number n greater than that of any prepare request it saw, it responses YES to that request with a promise not to accept any more proposals numbered less than n and include the highest-numbered proposal (if any) that it has accepted.

  <prepare ,n>

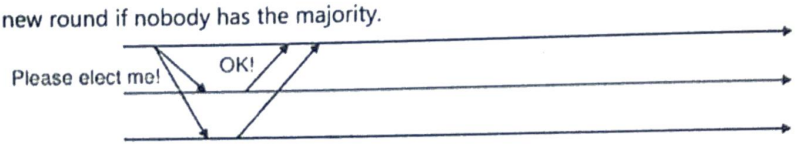
  if n< np

  reply prepare reject (as already saw a higher proposal number), where np is previous proposal

  else

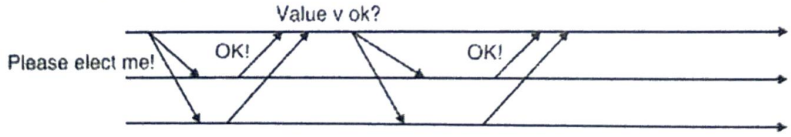
  np = n

  reply prepare accept , (this node will not accept any lower than n proposal.
- Acceptors record the PREPARE ID they got on disc as well. Node is the leader if the majority of responses are positive. Start a new round if nobody has the majority.



Phase 2 :

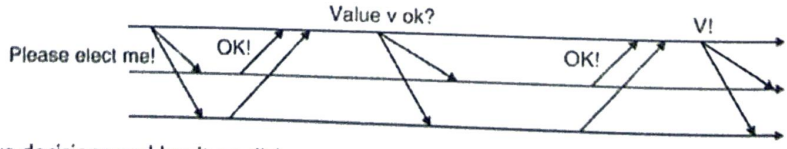
- If the majority of the acceptors respond YES to the proposer's prepare requests, the proposer will issue accept requests for each of those acceptors for proposals with numbers n and v, where v is the value of the proposal with the largest number among the responses.



- Recipient log on disk and responds OK.
- Delay the procedure and restart Paxos if the leader is unable to get the support of the majority.

Phase 3 :

- If leader hears OK from majority, it lets everyone know of the decision (learning).
  - send (decide, v) to all nodes



- Recipient receive decisions and log it on disk.
- Delay the procedure and restart Paxos if the leader is unable to get the support of the majority.

- Over distributed networks, it is possible to reach consensus, but it is not an easy task. This issue gave rise to the concept of Byzantine fault tolerance.
- Lamport, Pease, and Shostak proposed the Byzantine Generals Problem in 1982 to guarantee that malfunctioning components providing conflict information are managed inside a system.

What is the Byzantine Fault Problem / Byzantine Generals Problem?

- This problem represents an issue of computer networking over an unreliable link.
- A theoretical problem known as The Byzantine Generals' Problem illustrates how several Byzantine generals might find it difficult to communicate while deciding on their next course of action.
- The Byzantine Generals problem demonstrates the value of blockchains and explains why they are important. It will aid in understanding why a blockchain operates as it does.
- First we will discuss the unsolvable Two Generals Problem. Following that, we'll expand on that to the Byzantine Generals' Problem and talk about Byzantine Fault Tolerance in distributed and decentralized systems.

Two Generals Problem

- This problem was first published in 1975.
- In this problem we will call these two nodes as generals. In this situation the generals in their armies have come upon a city that they wish to attack.
- Each general's army on its own is not enough to defeat the enemy's army successfully, thus they need to cooperate with an attack at the same time the problem arises because two generals are separated by a significant distance.
- So think of this as if they are like on opposite sides of the city.
- General 1 is considered as a Leader and the other is considered the Follower. In order for them to communicate and decide on a time of attack, General 1 has to send a messenger across the enemy's camp that will deliver the time of attack to General 2.
- But this could mean that the messenger could get captured and the message would be lost forever.
- This will result in General 1 attacking while General 2 and his army hold their grounds assuming the first message goes through.
- General 2 must still acknowledge that he received a message so he sends a messenger back.
- Thus repeating the previous scenario where the messenger can get caught.
- This opens up an infinite loop where both parties are sending acknowledgements back and forth with neither General knowing whether or not the message was received.
- If General 1 receives a message how General 2 will know that the message was received without an additional acknowledgement message.
- General 1 presents a repeating problem because General's problem is currently unsolvable.
- And still a newer problem was iterated off of this one, which is Byzantine Generals Problem.

Byzantine Generals Problem

- We have already learned about this problem in Module 1.

- This problem was introduced in 1982 in a paper written by Leslie Lamport, Robert Shostak and Marshall Pease which were a group of researchers.
- It's a special case of the Two Generals Problem that has been generalized.
- Similar to the Two Generals problem, the Byzantine Generals problem deals with reliable computer systems handling malfunctioning components that give conflicting information to different parts of the system to ensure consensus within the distributed computing system.
- The difference now is that more than two generals need to agree on a time to attack their common enemy.
- The same situation is described, but this time, more than two generals are required to agree on when to launch an attack against their common enemies.
- One or more of the generals may be a traitor in this situation, which would allow them to lie about their decision, adding another layer of complexity (e.g. they say that they agree to attack at 08.00 but instead they do not).
- An arrangement of commander and lieutenant replaces the leader-follower paradigm described in the Two Generals Problem.
- The commander and each lieutenant must reach consensus here in order for the action to be taken.
- The issues that could develop when communicating a decision from one general to another are while conveying the message, the messenger could be arrested; What if the message was altered by a forger? ; How can a general be certain that he received the message from the expecting general? ; What if other generals turn traitor and tell their troops to strike but then they flee?
- How can the system be confident that every general will launch a simultaneous attack from their predetermined location? Is there any other option but to have complete trust in one another?
- With its Byzantine fault tolerance (BFT) consensus method, blockchain appears to solve this issue.

Byzantine Fault Tolerance (BFT) Algorithm

- BFT algorithm is based on state machine replication.
- In distributed systems, ensuring that computers can establish consensus on specific facts in a reliable manner is a very real difficulty that byzantine fault tolerance seeks to address.
- An method that resists a system to enter the Byzantine Generals' problem is known as Byzantine Fault Tolerance (BFT).
- BFT system is able to continue operating even if some of the nodes fails communicate who acts maliciously.
- The general's team needs an algorithm that can follow the following guidelines in order to guarantee the success of the generals' team: The next action of the plan must be approved by all troop generals ; the generals must be dependable and committed to the establishment ; the impact of network traitors on generals must be avoided ; they must adhere to the system's algorithm ; no matter what the traitors do, the generals must agree on something or make a choice ; at no point during the action, the system or network should result in a 51 percent attack.
- The basic operation of a BFT algorithm is :
  - **Phase 1 :** Client sends a request to the primary. The primary can then validate the message and propose a sequence number for it.

- **Phase 2 :** Primary node sends pre-prepare message to all backup nodes. This allows the backup nodes to validate the message and receive the sequence number.
- **Phase 3 :** All functioning backup nodes sends prepare message to all other backup nodes. This allows replicas to agree on a total ordering.

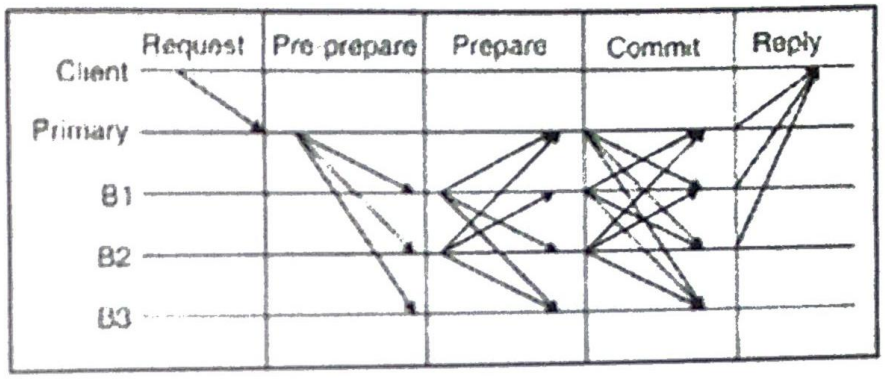


Fig. 5.7.1 : Working of BFT

- **Phase 4 :** All replicas multicast a commit. The replicas have agreed on an ordering and have acknowledged the receipt of the request.
- **Phase 5 :** Each functioning replica sends a reply directly to the client. This bypasses the case where the primary fails between the request and reply.

- The amount of network communications among participants in the BFT-based consensus method grows as the number of participants rises. This is so because each transaction's processing requires participation from every participant in the BFT-based consensus method as shown in Fig. 5.7.1.1, and the process itself is made up of a variety of steps. Due to the fact that the completion of the consensus process slows down as more participants join, this feature of the BFT-based consensus method may cause performance and scalability issues.
- Limitations of BFT :
  - If malicious clients have the necessary access, they may corrupt the system state.
  - No consideration is given to fault-tolerant privacy.

5.8 Introduction to Hyperledger

- In order to improve cross-industry blockchain technology, the LF produced the open source initiative known as Hyperledger.
- "Hyperledger is an open sourced community of communities to benefit an ecosystem of Hyperledger based solution providers and users focused on blockchain related use cases that will work across a variety of industrial sectors." – Brian Behlendorf, Executive Director of Hyperledger
- Hyperledger is neither a Blockchain nor a company, nor is it a cryptocurrency. It is a piece of software used to build one's own custom blockchain service.
- A cross-industry open source collaborative project called Hyperledger was developed to advance blockchain technologies.
- Global leaders in banking, supply chains, technology, IOT, and finance are participating in the collaboration with Linux Foundation, which is also being hosted by the Linux Foundation.
- According to the Hyperledger philosophy, multiple private chains will manage multiple markets around the world.
- Since each business is distinct in its own right, applications that cater to these businesses should be created using individualized rules.
- Unlike Ethereum, which frequently forces programmers to create their applications using generic protocols.
- In late 2015, a small group of developers launched the Hyperledger project.
- These developers were from a range of industries, including data science, manufacturing, banking, etc., and they shared the objective of making blockchain technology more usable to developers and enterprises.

Why is Hyperledger necessary ?

- Public blockchains do not support private and confidential transactions.
- Public blockchain is having issues with scalability.
- The developers discovered during extensive testing that blockchain networks, where each peer must simultaneously run consensus and validate each and every transaction, suffer greatly in terms of scalability.
- On top of that, due to the extensive measures that are taken to ensure a transaction's integrity, transactions with a degree of confidentiality and privacy attached to them cannot be executed on public blockchains.

Difference between popular and emerging blockchain platforms

Platforms	Ethereum	Hyperledger Fabric	R3 Corda	Ripple	Quorum
Public/Private					
/Permissioned	Public Blockchain- No permission is needed to access network content.	Private Blockchain- Network is limited to people with permission.	Private Blockchain- Permission is needed to access network content.	Public Blockchain- Permissionless network content	Both Public, Private permissioned network content
Smart Contract	Solidity programming language	Golang programming language	Kotlin programming language	C++ programming language	Solidity and Vyper
Governance	Carried out by developers (DAO)	Linux foundation incharge	R3 company in charge	Ripple Labs	Etherum developers and JP Morgan Chase
Consensus	Relies on Proof of Stake for decision making	The consensus process does not require participation from every node in a network.	Making decisions is limited to those parties engaged in a transaction.	Participating nodes verify the authenticity of a transaction by conducting a poll.	Multiple voting based consensus mechanisms which do away with the current proof of work consensus mechanism used today by the public Ethereum blockchain among many others

Platforms	Ethereum	Hyperledger Fabric	R3 Corda	Ripple	Quorum
Consensus Protocol	PoW/ PoS	Puggable	Notary-based	Probabilistic voting	Pluggable
Currency	Ether	No native currency	No native cryptocurrency	XRP	ETH
Use Case	Popular with generalized applications and mostly used for P2P and B2C operations.	the preferred platform for B2B operations, mainly used in enterprise.	Runs on a specialized distributed platform for the financial industry needs.	Runs on a specialized distributed platform for the financial industry needs.	Popular with generalized applications and mostly used for P2P and B2C operations.