Distributed Consensus Algorithms to the Blockchain Consensus Mechanism

This article looks into the theoretical problems and challenges of distributed consensus and corresponding consensus algorithms.

A **consensus algorithm** is a process in computer science used to achieve agreement on a single data value among distributed processes or systems. **Consensus algorithms** are designed to achieve reliability in a network involving multiple unreliable nodes.

A distributed consensus ensures a consensus of data among nodes in a distributed system or reaches an agreement on a proposal. This topic may be very familiar to any technicians that work with distributed systems such as HDFS, MQ, ZooKeeper, Kafka, Redis, and Elasticsearch. With the rapid development and the increasing complexity of distributed networks, developers have always been exploring possible solutions to solve this persistent problem in both theory and practice.

Next, with the rise of blockchain technology, especially public blockchains in open networks and private blockchains in permissioned networks, this consensus problem has once again received much attention and needs to be considered from a new perspective.

In this article, we will look into the problems and challenges of distributed consensus and corresponding consensus algorithms. We will also briefly analyse the applicability and limitations of these consensus algorithms and discuss the combination of these traditional consensus algorithms and new blockchain technologies. Later, this article concentrates on the consensus algorithm and mechanism in the public blockchain field from the perspective of the reliability of human beings. This article also considers the association between the distributed consensus algorithms in traditional computer science and the consensus mechanism in blockchain and shows how new consensus ideas can be seen in the public blockchain field.

Problems and Challenges of Distributed Consensus

To fully understand distributed consensus, we need to first build an understanding of the features of a distributed network. What are the main features and characteristics of a distributed network? Or what are some possible problems involved with a distributed network? Let's look into some of these questions in this section of this article.

Crash Fault

First, let's consider cash faults. A crash fault in a distributed network often may be related to one of the following issues:

- Nodes or replicas may experience downtime at any time, stop running for a short time and recover later.
- The network may be interrupted at any time.
- A sent message may be lost during delivery and cannot be received.
- A sent message may be delayed and received after a long time.

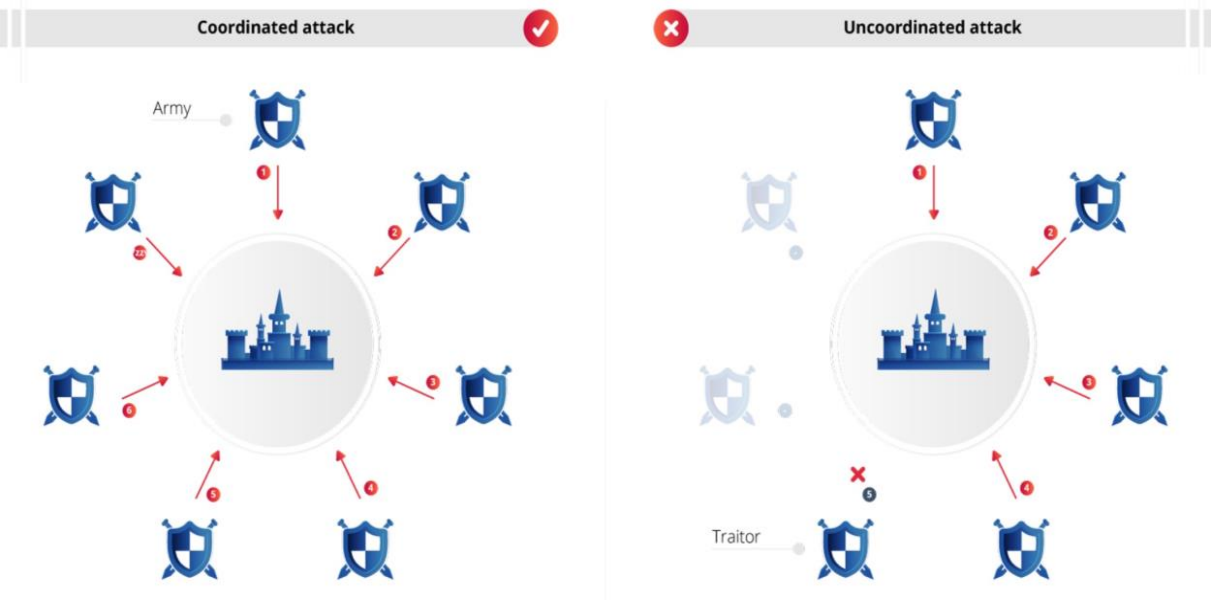
- Messages may experience the out-of-order problem during the delivery process.
- The network may be divided. For example, due to poor communication between clusters in China and the US, the entire network may be divided into two sub-networks for the China clusters and US clusters, for instance.

These above problems are common in distributed systems. They are essentially the inevitable risks caused by unreliable and unstable physical hardware in distributed systems. For example, networks, or communication channels, cannot always be stable and reliable. Disks on physical machines or CPUs will not always be of good condition. Therefore, it is safe to say that crash faults are the most basic and common type of faults to be solved in distributed systems.

The Byzantine Fault

The crash faults are based on a simple assumption: Either nodes do not work or respond normally, or although they work and respond normally, they cannot implement inconsistency, that is to say, being idle is okay for them, but they cannot commit some errors. Malicious nodes in networks may change and forge data at any time, making it harder to solve the consensus problem. These troublemaking problems that may change and forge data or response information are often refer Byzantine faults. The crash fault is called a non-Byzantine fault.

Byzantine originated from Lamport's paper. It is no exaggeration to say that Byzantine fault tolerance (BFT) is the most complex and rigorous tolerance model. By analogy, some number of generals plan an attack on a castle together and each general can choose to start the attack or retreat. However, to successfully take the castle, all the generals must act synchronously. Next, given that the generals are too far from each other to use direct communication, messengers are used to carry messages. However, messages are not reliable. They may successfully delivery messages after a very long time, fail to deliver messages or even change with messages. The generals may not be reliable, either, for example, one of them may be a traitor who does not act in accordance with the plan. The messengers in this story represent communication channels in distributed networks and the generals represent nodes.



[Byzantine fault tolerance]

Fault Tolerance

The most critical problem that distributed consensus algorithms need to solve is the question of how to implement the certainty and consensus, so that reliable consensus results are returned across the entire distributed network, which can be full of risks and uncertainties. Naturally, it is relatively easy to solve crash faults. Algorithms used to solve this type of faults are called crash fault tolerance (CFT) algorithms or non-Byzantine fault tolerance algorithms. Byzantine faults may cause unauthorized changes, have higher complexity and are more difficult to solve. Algorithms for solving these problems are called Byzantine fault tolerance algorithms.

What is the boundary between the two types of fault tolerance algorithms? In what scenarios do these two types of faults occur? Is it really necessary to take unauthorized changes into consideration? The answers to these questions may depend on the actual network environments and business scenarios.

Crash Fault Tolerance

Generally, if a system is in a reliable internal network, we only need to consider the issue of crash fault tolerance (CFT). For example, for distributed components in many companies such as distributed storage, message queue, and distributed services, we only need to consider CFT. The reasons for this is as follows: The entire enterprise network is closed and protected by multiple firewalls, making external access and attacks unlikely. Individual nodes are deployed in a unified manner, and it is very unlikely that the machines and running software gets changed without proper authorization. The distributed network is relatively "pure" at this point, and we only need to pay special attention to the communication network and machine hardware. We need to consider network latency and instability as well as downtime and faults that machines may experience at any time.

Byzantine Fault Tolerance

Then, there's Byzantine fault tolerance (BFT), which related to the entire distributed network being evaluated in a larger environment. In addition to physical hardware, it is also necessary to take some "man-made" factors. After all, it is specific persons instead of machines that perform misconduct. Assume that a distributed network is relatively open, for example, a private network of tens of companies in a specific industry. Or assume a completely open network, for example, a network that anyone has access to. Node machines and software on these machines are deployed by individual companies or individuals themselves. If the benefit is tempting enough, a person may launch DDoS attacks on one of these nodes, making authorized, often malicious changes to software code and to the code execution logic, or even the data that is persisted on disks in the network. In such case, we face bigger challenges. In addition to the unreliable communication networks and machine hardware, we need to consider and deal with the "troublemakers" in the system.

Impossibility Triangle

To solve these problems encountered in actual scenarios, many computational scientists have conducted lots of theoretical studies. These theoretical studies may seem too abstract and cumbersome for engineering technicians, and some of these studies are about boring mathematical issues. However, these theories can provide significant guidance on how to solve these problems. Also, these theories show what the theoretical limitation of possible solutions are and which direction can be explored and which direction cannot work. Standing on the shoulders of giants, we do not have to spend all the energy making a "perpetual motion machine". Since most of you have some knowledge of these theories, let's just do a brief review.

Fisher, Lynch, and Paterson (FLP) Impossibility

As early as 1985, Fisher, Lynch, and Paterson published the impossibility theorem of distributed consensus. Earlier, we also have shown that a natural and important problem of fault-tolerant cooperative computing cannot be solved in a totally asynchronous model of computation. That is to say, in asynchronous networks, it is impossible to implement consensus algorithms that tolerate even a single node fault. Building on this, Byzantine failures are not taken into consideration in this theorem. It is also assumed that the network is very stable and that all messages are delivered correctly and exactly once. In this paper, we show the surprising result that no completely asynchronous consensus protocol can tolerate even a single unannounced process death. We do not consider Byzantine failures, and we assume that the message system is reliable—it delivers all messages correctly and exactly once.

Of course, this is only theoretical. It shows the theoretical limitations of solving these problems, but it does not mean that these problems cannot be solved in practice. If we are willing to relax restrictions and make some sacrifice, we can find practical and feasible solutions in engineering.

The most basic prerequisite of the FLP impossibility theorem is the asynchronous network model. What are the characteristics of an asynchronous model and a synchronous model respectively?

- In an asynchronous model, the message latency from one node to another node is finite but can be unbounded. This means that if a node does not receive a messages, it cannot judge exactly whether the message has gotten lost or just delayed. In other words, we cannot determine whether a node has experienced failures or not based on timeouts.
- In a synchronous model, the latency in message delivery is finite and bounded. This means that we can accurately estimate the possible maximum message latency based on our experience or the sampling to determine whether messages are lost or whether nodes experience failures based on timeouts.

Fortunately, our real network environment is more similar to a synchronous model. Therefore, we can determine the maximum timeout based on experience or sampling. For example, you mailed a book to one of your friends. However, after three days, the book is still not delivered to your friend. At this point, your friend may find it hard to determine whether the delivery has been delayed or whether the book has gotten lost during the delivery process. However, if the book is still not delivered to your friend after one month, you and your friend can basically come to the conclusion that the book has gotten lost during delivery. This conclusion is based on our experience and statistics: An item can usually be successfully delivered within one to two weeks. An asynchronous model reflects the worst cases and the extreme situations of the inter-code communication. An asynchronous model has something in common with a synchronous node: a consensus protocol that works in an asynchronous model also works in a synchronous model. A synchronous model has modifications and restrictions on an asynchronous model so that the synchronous model is closer to the real scenarios and it is possible to solve the consensus problem in practice.

In addition, even in an asynchronous network model, FLP does not indicate that consensus is unreachable, just that it's not always reachable in bounded time. In practice, it is still possible to find solutions if the restrictions on the bounded time are relaxed.

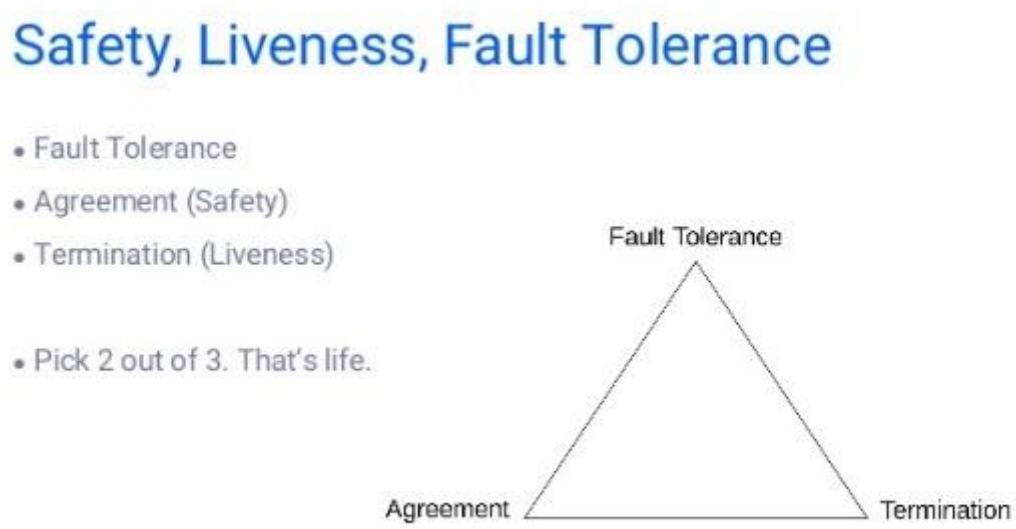
According to research into DLS, consensus algorithms can be divided into three major types by network model.

- Consensus protocols in partially synchronous models can tolerate up to 1/3 of any failures. In a partially synchronous model, the network latency is bounded, but we cannot know the boundary in advance. This type of fault tolerance also contains Byzantine faults.
- Deterministic protocols in an asynchronous model cannot tolerate faults. As mentioned before, in an asynchronous model, the network latency is unbounded. This conclusion is actually what the FLP impossibility theorem implies: Deterministic protocols in a completely asynchronous network cannot tolerate errors in even a single node.
- Protocols in a synchronous model can surprisingly support 100% fault tolerance, although they will limit node behaviors when the number of faulty nodes exceeds 1/2 of the total nodes. In a synchronous model, the network latency is bounded (smaller than a known constant).

From a different perspective, FLP actually covers three properties of a distributed system: safety, liveness, and fault tolerance.

- Safety means that the values reached across nodes in a system are consistent and valid. Safety is the most basic requirement to ensure system consistency. The core of safety is to ensure that it cannot do something bad.
- Liveness indicates that individual nodes in a system must reach an agreement (in bounded time), that is, the system must move forward and cannot always be in the inconsistency state. Liveness is actually a higher requirement. It means that you cannot do something bad but also cannot always do nothing. You must do something good, that is, to make the entire system run smoothly and normally.
- Fault tolerance requires that a protocol must also effective in case of node failures.

FLP impossibility means that, in an asynchronous network, no distributed consensus protocol can meet the three properties at the same time. In a distributed system, node failures are almost inevitable. Therefore, fault tolerance must be taken into consideration. FLP impossibility means that any consensus protocol can only have either liveness or safety in addition to fault tolerance. In practice, we can often make some sacrifices. For example, we can sacrifice a certain degree of safety, which means that the system can always reach an agreement quickly but the agreement is not very reliable. We can also sacrifice a certain degree of liveness, which means that the system can reach a very reliable agreement but this process takes too long or an agreement can never be reached due to relentless debate. Fortunately, many practical scenarios show strong robustness, making it very unlikely for events to invalidate a consensus protocol.



FLP impossibility diagram

In addition, FLP does not exclude randomized algorithms like Las Vegas. Many consensus algorithms adopt the Las Vegas algorithm to avoid the limitations imposed by FLP impossibility on deterministic and asynchronous networks. These non-deterministic consensus algorithms involve the Las Vegas rule: A consensus is always reachable in a network, but the time needed to reach consensus may be unbounded. When these type of algorithms are used, it is likely to reach a consensus decision in each round. The probability (P) of reaching consensus within T seconds will increase exponentially with the increase of T and will get closer and closer to 1. In fact, this method has been adopted by many successful

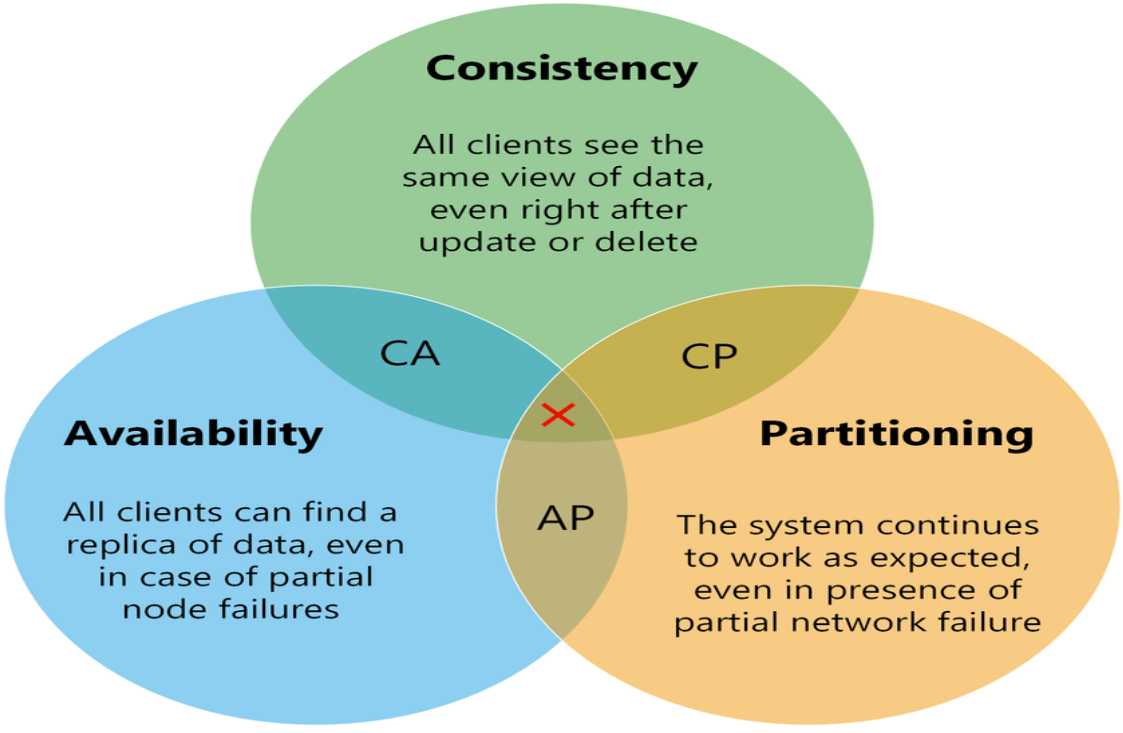
consensus algorithms and is an escape hatch within the scope of the FLP impossibility. The bitcoin consensus mechanism described later in this article also adopts this method.

Consistency Availability Partition tolerance (CAP) Theorem

The well-known Consistency Availability Partition Tolerenace (CAP) theorem states explicitly from a different perspective that "Of three properties of shared-data systems (data consistency, system availability and tolerance to network partitions), one can only achieve two at any given time". CAP is very similar to FLP, but they are not exactly the same. They focus on different perspectives. Even very similar concepts do not have the exactly same meanings. For example:

- FLP focuses on the distributed consensus problem, while CAP focuses on data synchronization and replication in a distributed network.
- FLP states that all the three properties of FLP cannot be implemented in an asynchronous network model, while CAP states that all the three parts of CAP cannot be implemented in all scenarios.
- Liveness in FLP emphasizes the internal property of a consensus protocol. The availability part of CAP emphasizes the external property of a consensus protocol.

Theoretically, only two of the CAP parts can be implemented. However, the boundary selection is not binary. The whole spectrum in between is useful, as it mixes different levels of Availability and Consistency usually yields a better result.



CAP theorem diagram

In practice, we often need to make some trade-offs based on actual business scenarios. For example:

- Many traditional relational databases (for example, MySQL) often use ACID (atomicity, consistency, isolation, and durability) and ensure strong consistency through synchronous transaction operations. The availability is generally not very good because the number of nodes is small (only primary and secondary nodes in most cases). The relatively simple network topology also reduces the partition tolerance.
- NoSQL storage systems (for example, HBase) usually adopt BASE (basically available, soft state, eventually consistent) and ensure high availability through multiple nodes and replicas. The number of nodes is larger, the network environment is also more complex, and the partition tolerance is also taken into account. However, BASE can only implement weak consistency and ensure the eventual consistency.

Comparison between ACID and BASE

Of course, these statements are not the final conclusion. Since individual systems are evolving constantly, a correct conclusion today may be not true tomorrow. To become better, a system has to keep exploring scenarios that are suitable and find an optimal balance point.

The Distributed Consensus Algorithm

To process a variety of real and complex problems and challenges in distributed systems, many solutions have been developed based on theoretical guidance. This article does not describe the implementation details and specific differences of these algorithms. Instead, only a general introduction is given to make a comparison on the whole from a broader perspective.

The Paxos Algorithm

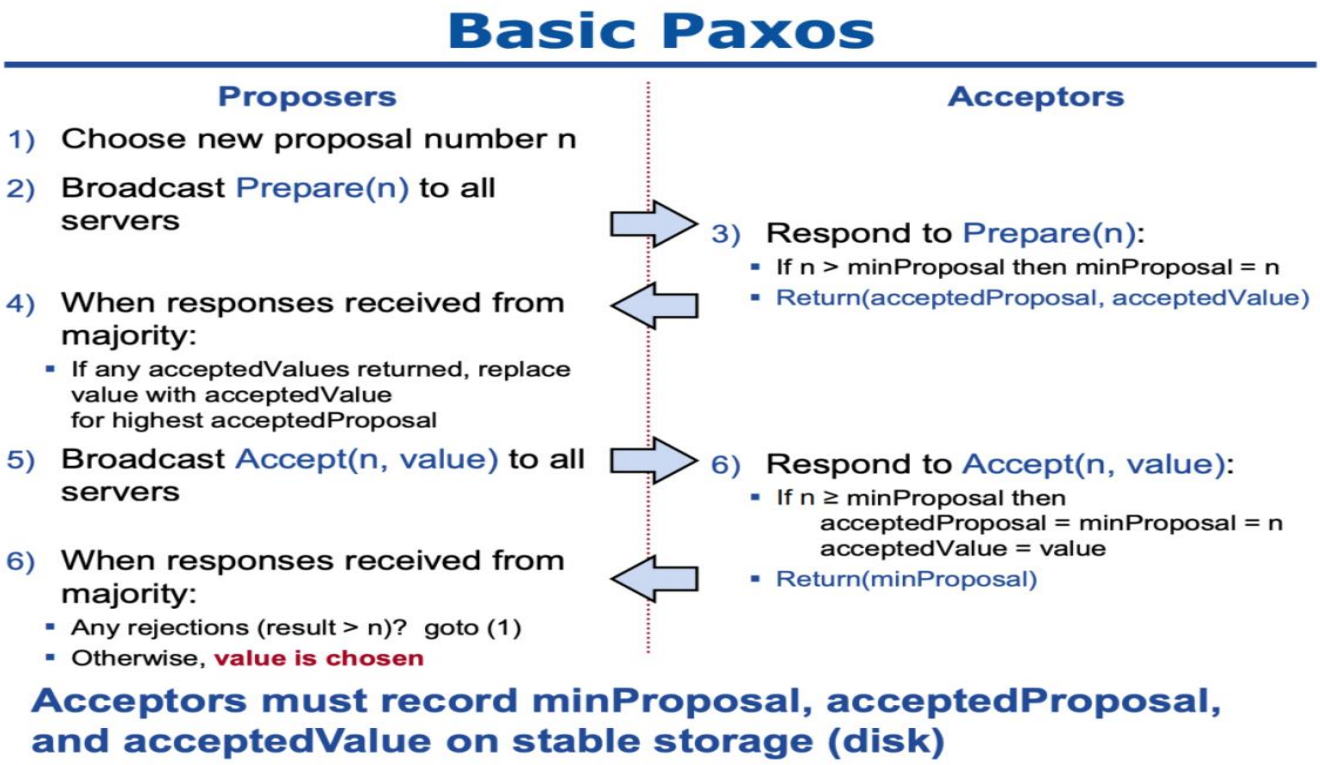
One of the most famous distributed consensus algorithms is Paxos suggested by Lamport, though its complexity is also "notorious". Lamport proposed this creative mechanism that is practical and can be implementable through engineering and can ensure the consistency of distributed systems to the maximum extent. Paxos is widely used in many distributed systems, including Chubby and ZooKeeper. Basic Paxos (single decree, that is, to only agree on a value each time) has two roles: A Proposer can process client request and actively propose a proposal value. An Acceptor passively responds to the information sent by a Proposer, votes on proposals made, and persists values and states during the decision-making process. To simplify the model, the Learner role can be ignored. This does not affect the decision-making in the model.

As shown in the figure, the consensus decision-making process uses the two-phase commit protocol:

- In the first phase, broadcast the Prepare RPC command to find the final value decided by the protocol and block unfinished old proposals.
- In the second phase, broadcast the Accept RPC command to require Acceptors to accept a specific value agreed on. Multi-Paxos consists of multiple instances of Basic Paxos and can decide on a series of values.

The implementability of Paxos in practice is also based on many assumptions and restrictions. Paxos can only be used to process CFT and cannot be used to process Byzantine failures. Therefore, it is a non-Byzantine fault tolerance algorithm. From the perspective of FLP, Paxos implements fault tolerance and safety and abandons liveness (safe but not live). That is to say, this algorithm may never end or reach consensus, though this case is very unlikely. From the perspective of CAP, Paxos only ensures C (consistency) and P (partition tolerance), but weakens the level of availability. To improve the availability of Paxos systems, we can increase the number of Learners.

Despite these shortcomings, Paxos is still reliable, effective and well-tested in practice. In essence, Paxos is a (dominant) distributed consensus protocol in asynchronous systems. The inventor of Chubby even said that "there is only one consensus protocol, and that's Paxos"—all other approaches are just broken versions of Paxos. Why Paxos is effective in practice is that conditions that may affect the liveness and availability of Paxos systems are usually not easily triggered. If those conditions do occur, the impact is also not very unacceptable.



Basic Paxos RPC communication and decision-making process

The Raft Algorithm

Due to the complexity of Paxos, Ongaro presented a simpler algorithm in 2014—Raft. Raft is much easier to understand and implement in engineering. This is also the initial objective of Raft. Many easy-to-understand design details have been made on the condition that the functionality is not affected.

The Raft algorithm is a leader-based asymmetric model. A node in a system can only be in one of the three states at any point in time: leader, follower, and candidate. In the initial stage, all the nodes are followers. To become the leader, a node (follower) must become a candidate and launch a round of electoral votes. If the node does not receive enough votes, the node becomes a follower again. However, if it receives a majority of the votes, the node becomes the leader. If the leader encounters failures and finds that a new leader is elected after it recovers from failures, the original leader automatically goes back to the follower state.

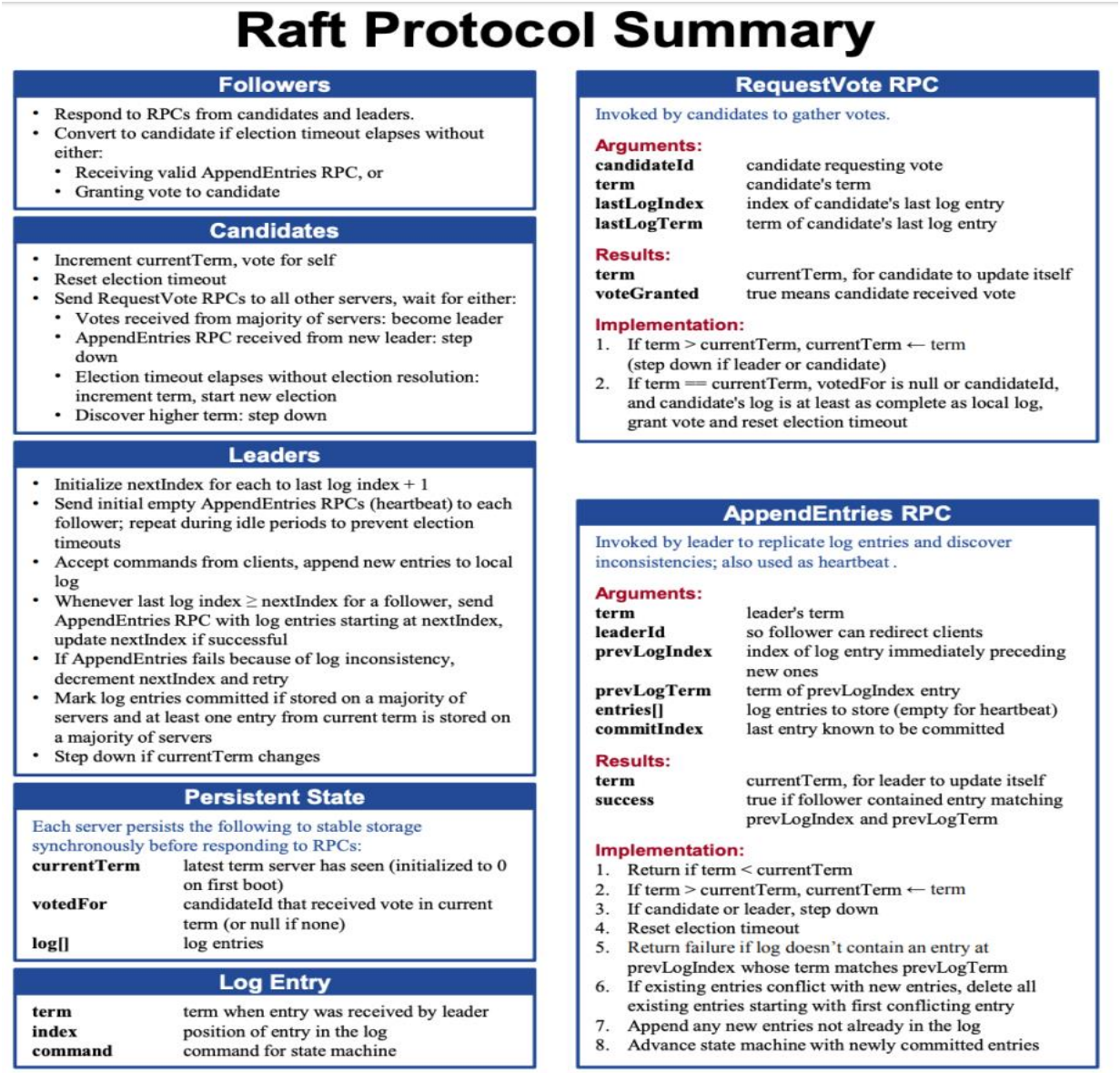
Raft also introduces the Term concept to identify expired information in a timely manner. A term is similar to an epoch file in ZooKeeper. A term number increases monotonically over time and at most one leader can be elected in a given term. If the logs have a last entry with different terms, then the log with the later term is more up-to-date.

Raft also introduces the heartbeat packet and timeout. To maintain its authority, an elected leader must continuously send a heartbeat packet to the other nodes in the cluster. If a follower does not receive the heartbeat packet during a given election timeout, the leader is considered to have crashed and the follower changes its status to candidate and starts a leader election.

The leader election in Raft is implemented through the heartbeat and random timeout. Log replication is implemented through strong leadership: The leader receives the client command, appends it to its log and replicates the log to other followers. Raft ensures safety by only allowing a leader to decide whether to commit a log or not.

Election and replication will not be described here in detail. For more information about election and replication in Raft. Note that the leader election and the normal operations orchestrated by the leader are relatively simple. The leader change process is actually a bit more complex in Raft.

However, although the principle/mechanism of Raft is not exactly the same as Paxos, the problems that they solve and the trade-off policies that they adopt can be considered similar. That is to say, Raft can only solve crash faults, emphasizing fault tolerance, safety, and consistency and weakens the level of liveness and availability.



Raft overview

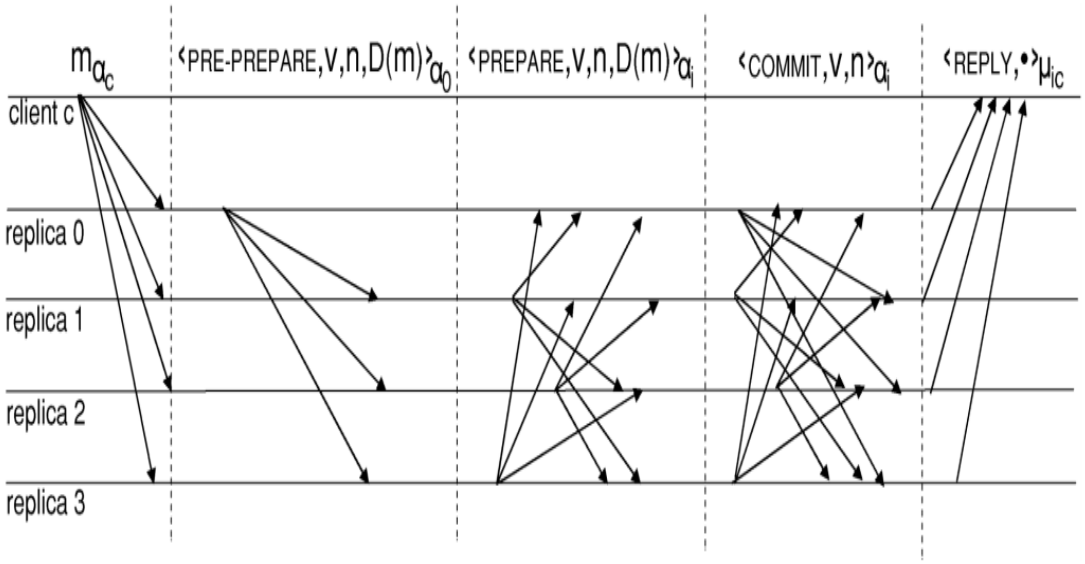
Practical Byzantine Fault Tolerance (PBFT)

Although many discussions on BFT solutions have been conducted since the Byzantine Generals' Problem raised by Lamport in 1982, many solutions for these problems are inefficient, slow and complex. The situation improved in 1999 when Castro and Liskov presented the Practical Byzantine Fault Tolerance (PBFT) algorithm. PBFT is the first algorithm of its kind with the complexity reduced from the exponential level to the polynomial level. PBFT enables several thousand TPS and feasible solutions to nodes acting maliciously in practice. It is proven that the PBFT algorithm will work normally if the number of malicious nodes in a system is no more than 1/3 of the total nodes.

All the nodes in a PBFT system are ordered sequentially with one node being the leader node and others considered as backup nodes. All the nodes in a system communicate with each other and reach consensus based on the majority principle. Each PBFT consensus round is called a view. The leading node is changed during every view and can be replaced with a protocol called a view change if a certain amount of time has passed without the leading node broadcasting the request. This replica timeout mechanism ensures that the crashed or malicious leader can be detected and that a new view starts by re-electing a new leader.

As shown in the figure, five phases are experienced from the client launching requests to receiving responses. The consensus process adopts the three-phase protocol. The following content briefly describes the five phases:

1. **Launch**: The client (client c) initiates the service request m to the cluster.
2. **Pre-prepare**: The leader node (replica 0) verifies the validity of the request message m, assigns the sequence number n to the request m in the view and broadcasts the assigned pre-prepare message to all the backup nodes (replica 1-3).
3. **Prepare**: The backup nodes verify the validity of the request message m and accept the sequence number n. If a backup node accepts the assignment scheme, it broadcasts the corresponding prepare message to the other nodes. In this phase, all the replicas are required to reach a globally consistent order.
4. **Commit**: Once the assignment agreement message is received from the cluster, all the nodes (primary and secondary nodes) broadcast the commit message to all the other nodes. In this phase, all the replicas have agreed on the order and confirmed the received request.
5. **Execute and reply**: After receiving the commit message from the cluster, the nodes execute the request m and send replies to the client. The client awaits the same reply from f+1 different nodes and considers that the request has been executed successfully. f represents the maximum number of potential faulty nodes in the clusters. That all the nodes directly return messages to the client is also to prevent the primary node from having problems during the request.



PBFT normal operations

PBFT implements safety based on an asynchronous network model, but it depends on message timeouts to perform periodic synchronization. Because the leader-plan solution is adopted, the message synchronization process is very fast and the sequential writing is also implemented. However, the leader re-election is difficult. A malicious leader may start to send a message when the time is very close to the timeout window, causing the system to be seriously slow. This disadvantage can be used to launch attacks against the network and make correctly running nodes look incorrect, causing endless leader election.

Compared with Paxos and Raft, PBFT can process more problems: In addition to crash faults, it can process Byzantine problems that may cause troubles and unauthorized changes. However, from the perspective of the trade-off policy adopted in PBFT, PBFT is still similar to Paxos and Raft. From the perspective of FLP, PBFT also emphasizes the fault tolerance and safety and weakens the level of liveness. From the perspective of CAP, PBFT emphasizes the tolerance to network partition faults and consistency and weakens the level of availability.

Despite these shortcomings, PBFT is still effective and feasible in practice if the number of faulty or malicious nodes is no more than 1/3 of the total nodes. PBFT is not the only BFT algorithm. Other BFT-like algorithms are also evolving, for example, Lamport once suggested BFT Paxos (an enhanced version of Paxos) to process Byzantine faults. Recently, the BFT Raft algorithm was presented based on the combination of PBFT and Raft. However, from the perspective of the problem scope and mechanism, these algorithms are still similar to previous ideas and frameworks. Introduction to these algorithms will not be described in this article.

Applicable Scenarios

From Paxos, Raft, and PBFT to various variants of Paxos and Raft and new BFT-like algorithms, distributed consensus algorithms have been developing, improving and evolving. Many big companies have also developed distributed consensus algorithms that meet their business scenarios. Although these algorithms are not very perfect, they play an important

role in specific business practices. Then what are the application scenarios of these algorithms? What are the limitations of these algorithms?

Non-BFT algorithms such as Paxos and Raft can only process machine hardware faults and cannot handle situations where malicious nodes are present. These non-BFT algorithms can only run in very reliable network environments, for example, the internal network of a company. In such relatively closed networks, access requires strict authorization, ensuring that the identities of individual nodes are known and reliable. This serves to eliminate malicious nodes, and to allow the algorithms to run effectively.

BFT algorithms do have very strict requirements on the network environment. Even if malicious nodes are present, the entire system is still secure only if the malicious nodes are no more than 1/3 of the total nodes. However, this brings about new problems. How do you know exactly the number of malicious nodes in the network? What is the proportion of the malicious nodes to the total nodes? If access to the network requires permissions, it is relatively easy to solve this problem. For example, in a private network consisting of 10 affiliates, only the 10 authorized companies can access the network. Even if some companies (less than 3 companies) act maliciously and attempt to change data without authorization, the whole system is still reliable and secure. In such a permissioned network, the number of nodes that may act maliciously has been estimated. When some nodes do act maliciously, their real identities can be quickly located. This indirectly improves the safety of the network.

Limitations

However, BFT algorithms may cause problems in a permissionless (open permission, without permission control) open network. If a distributed network is open and can be accessed by anyone and the cost of the network access is low, it is unknown how many potentially malicious nodes may be in the network. Even when some nodes act maliciously, determining their identities is also a difficult issue. A typical attack scenario is a Sybil attack where attackers can forge identities to control a large number of nodes and then the entire distributed network.

In addition, the biggest limitation of BFT algorithms is that they can only coordinate a small number of nodes (for example, no more than 100 nodes). If the nodes are in the thousands, the system shows very poor performance or even fails to reach consensus, affecting the liveness and availability of the system. You may have noticed that multicast is required throughout the three-phase protocol of PBFT: In the pre-prepare phase, the primary node broadcasts the request to all the secondary nodes; in the prepare phase, the secondary nodes broadcast to all other nodes; in the commit phase, individual nodes (primary and secondary) broadcast to all other nodes. From this process, we can know that the number of communication times is the number of nodes squared. When a system has a large number of nodes, this broadcast mechanism will be a disaster. The system is almost unable to reach consensus in a short period of time.

From the preceding content, we can draw a conclusion that traditional distributed consensus algorithms such as Paxos, Raft, and PBFT are generally applicable to reliable distributed networks that require permission control and have a small number of nodes.

Application in the Private Blockchain

In fact, these traditional consensus algorithms also gain new vigor in the age of blockchain: They are further understood and used. These consensus algorithms are widely used in private blockchain scenarios, where the network environment is relatively reliable. The application of a private blockchain is prospective due to the following characteristics:

- **Access authorization**: A private blockchain is not fully open and generally consists of several or tens of enterprises. Only authorized companies or organizations can join the network (they usually need to complete the real-name authentication before joining the network).
- **Data protection**: Information and data in a private blockchain is not completely open and is visible to authorized parties only. This is especially important for industry or enterprise data security. For example, transaction information about cross-border transfers is very crucial in the banking industry and tax information in the taxation systems on the chain is also very sensitive.
- **Supervision**: Generally, supervision and observation nodes can be put in place in a private blockchain to monitor and audit sensitive information, so that the compliance requirements are met.

In the current phase, private blockchains can be considered a good choice to achieve quick solution implementation and solve pain points in a specific industry. The application of private blockchains in the blockchain industry also indicates the further exploration of future blockchain development. Because authorization is required to join a private blockchain, a certain level of trust has been established in advance, and the network environment is relatively reliable. The probability of malicious behaviors and attacks in the network is extremely low and even if they should occur, responsibilities can be easily and quickly determined. Therefore, traditional consensus algorithms can also be applied in these scenarios. See the following examples:

- HyperLedger Fabric v1.0 leverage Solo and the Kafka pub/sub system to perform the ordering; v1.4 also introduces the Raft algorithm. Currently, all these adopted algorithms are CFT algorithms. Raft mainly paves the way for subsequent support for BFT algorithms. (Raft is the first step toward Fabric's development of a byzantine fault tolerant (BFT) ordering service. As we'll see, some Raft development decisions were driven by this.)
- R3 Corda also adopts the pluggable consensus design. It allows both the Raft algorithm, which implements high speed and requires highly reliable environments, and BFT algorithms, which implement relatively low speed and require less reliable network environments. See this document for more information. .
- The Enterprise Ethereum Alliance (EEA) also supports BFT algorithms, Raft, and PoET. See this document.
- The Ant Financial Blockchain BaaS platform also adopts the PBFT algorithm.

Challenges in a Permissionless Network

Can the entire system reach consensus within a limited period of time if the network is completely open and permissionless and can be accessed by anyone at any time? How can we

coordinate all the nodes in a network, for example, containing 10 thousand nodes instead of only tens of nodes?

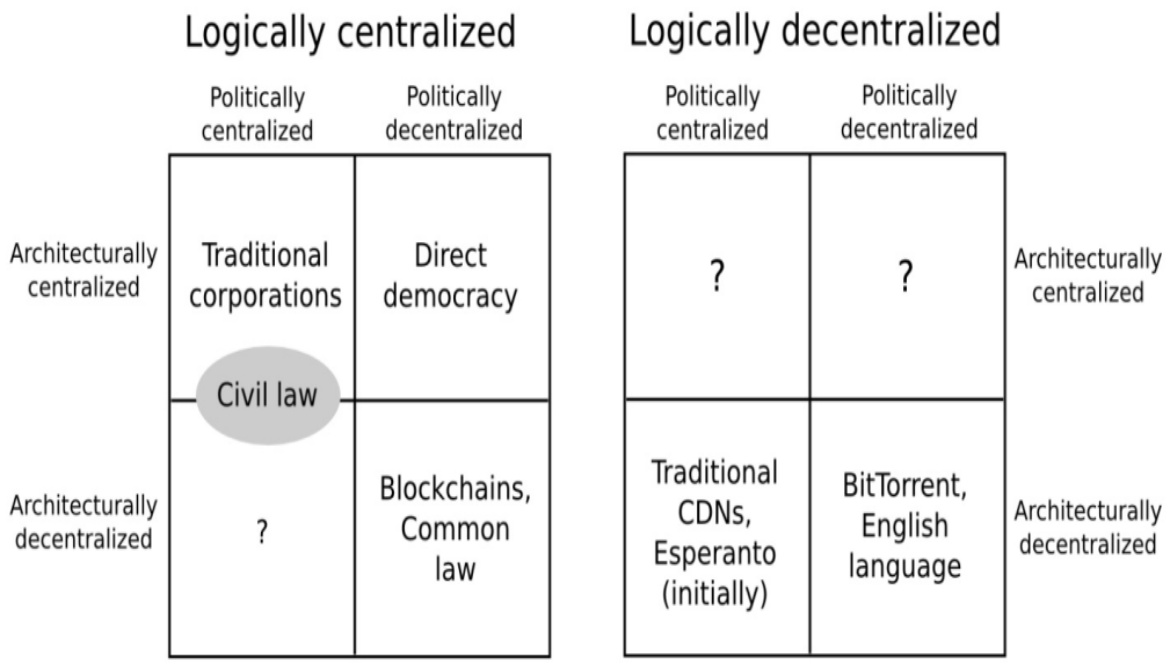
Before answering the preceding questions, you actually need to ask yourself the following questions: Why does a network need to be fully open and permissionless? What kind of scenarios will require 10 thousand nodes? Does this node requirement really exist in actual scenarios? The answers to these questions are directly related to public blockchains in the blockchain. To answer these questions, we need to review the goals of distributed systems.

Significance of Decentralization

Why do we need a distributed system? This question is not difficult to answer. Generally, a distributed system can increase fault tolerance. After all, a distributed system depends on many different nodes, and the probability of failures occurring on all the nodes at the same time is much lower than that on a single node. In addition, a distributed system also supports attach resistance. Attacking or destroying many nodes is much more difficult than attacking a single node.

However, the preceding content is still limited to the physical hardware. These advantages can reduce the probability of failures in the physical machine hardware. However, man-made factors are not taken into consideration. If a system is important enough (for example, an electronic money system), in addition to machine failures, we need pay more attention to man-made factors. Will the person who deploys nodes intentionally act maliciously? How can we prevent corruption and collusion among nodes in the system?

Vitalik Buterin, the founder of Ethereum, mentioned the significance of decentralization, as shown in the following figure. A traditional distributed system implements architectural decentralization from the perspective of fault tolerance and attack resistance (how many physical machines are in the system and on how many machines can the system allow simultaneous failures). Similarly, now we need to consider how to implement political decentralization and collusion resistance. How many people or organizations ultimately control nodes in the system? How can we prevent corruption and collusion? If we say that a traditional distributed system focuses on the reliability of networks and machine hardware, then what we need to consider now is the "reliability": Can we find an effective technique to prevent man-made malicious actions? How can we ensure that the majority of the nodes in an important network are not maliciously controlled by one person or organization?



Three dimensions of decentralization

It is worth mentioning that this question is still highly controversial. Many people have never thought of corruption and collusion or think that it is completely unnecessary to consider them. Perhaps they think that there is nothing they can do with this technical problem. After all, this question exists far away from the real world that we live in. We live in a world where centralized platforms receive a good reputation, provide credit endorsement and control all the rules and processes. For example, few people worry that banks will intentionally make false accounting and misappropriate their assets stored in the banks. After all, banks are generally considered reliable. If banks are not reliable, people may not be able to carry out any business activity.

However, banks being reliable is just our assumption. We are left with the options "trust" and "suspect" and we have to choose the former because we cannot carry out business activities and the economic development will also stagnate if we do not trust banks. However, no practical methods can prove that banks are very reliable.

If reliability is actually necessary and meaningful, can you find a solution to make this world more reliable? Can you prove that a stranger that you're doing business with is reliable instead of having to believe that stranger is reliable? Don't trust; please verify. You do not need to trust that stranger and do not have to trust that stranger, either. You just need to verify that stranger.

To solve this problem, all people must be equal. Everyone can participate in the decision-making process equally and freely. Everyone can freely enter and exit the "council". This is actually technical democracy, which includes the following technical elements: The network must be permissionless and anyone can enter or exit the network at any time; nodes must be peers and can communicate directly; no intermediary or centralized authority exists (completely peer-to-peer); each node may become a bookkeeper.

Because the network is permissionless, fully open, transparent and democratic, the number of participating nodes may be very large and the probability of malicious nodes is also very

high. Then, how do we coordinate the behaviors of nodes through a certain mechanism to ensure the consistency of the entire system, in this permissionless distributed network environment with a large number of nodes and a high possibility of malicious action? As mentioned before, consensus algorithms cannot do this. We need to seek new solutions.

In addition, decentralization may be the most controversial in the blockchain field. Some people think that decentralization is the value of blockchain, and the soul and premise of the existence of public blockchain, and the degree of decentralization of the system should be ensured as much as possible. While, others think that full decentralization is too ideal and unlikely to be achieved, so weak centralization or multi-centralization should be considered in combination with the actual situation, while giving consideration to the efficiency. Regardless of the value judgment, purely from a technical perspective, the higher the degree of decentralization, the higher the security of the system. Therefore, in the public blockchain system design, the degree of decentralization of the system should be ensured as much as possible. However, in conjunction with Vitalik Buterin's interpretation of the meaning of decentralization, in the process of pursuing decentralization, we should not stay at seemingly decentralized, but should comprehensively consider all dimensions of decentralization, and make the necessary trade-off according to the actual situation.

The Proof-of-Work (PoW) Mechanism

The innovative solution to distributed consensus in open networks is the Proof-of-Work (PoW) mechanism in Bitcoin.

Bitcoin

On October 31, 2008, Satoshi Nakamoto published the Bitcoin white paper "Bitcoin: A Peer-to-Peer Electronic Cash System", which miraculously provides a creative solution to such problems, making it possible to coordinate thousands of nodes in a complex network environment. In fact, Satoshi Nakamoto did not publish the white paper to solve this technical problem. On the contrary, Satoshi Nakamoto thinks bigger. He creatively invented Bitcoin, a completely peer-to-peer electronic cash system, to eliminate the trusted third-party middlemen that traditional payment needs to rely on. In the process of implementing the system, the problem of consistency among multiple nodes in the open network happened to be solved. It can also be said that, the core problem Bitcoin solves is the Double Spending problem of electronic currency in peer-to-peer networks. However, the Bitcoin implementation mechanism is not only about distributed network technology, but also combines cryptography, economics, game theory, and other ideas, and achieves consistency among nodes in a non-deterministic probability mode. Therefore, simply calling it an algorithm is no longer able to accurately express its meaning. It may be more appropriate to call it a consensus mechanism, because its implementation indeed relies on a complete set of policies and systems. Here, we will not elaborate much on the ideological significance and implementation details of Bitcoin, but focus only on the implementation of its consensus mechanism.

A bitcoin is actually an electronic chain of digital signatures. The coin owner can transfer the coin by signing the hash value of the previous transaction and the public key of the next owner, and adding these to the end of the coin. The payee verifies the chain formed by the coin owner by verifying the signature. However, the problem is that the payee cannot verify whether the coins received have not been double spent, that is, the owner may have

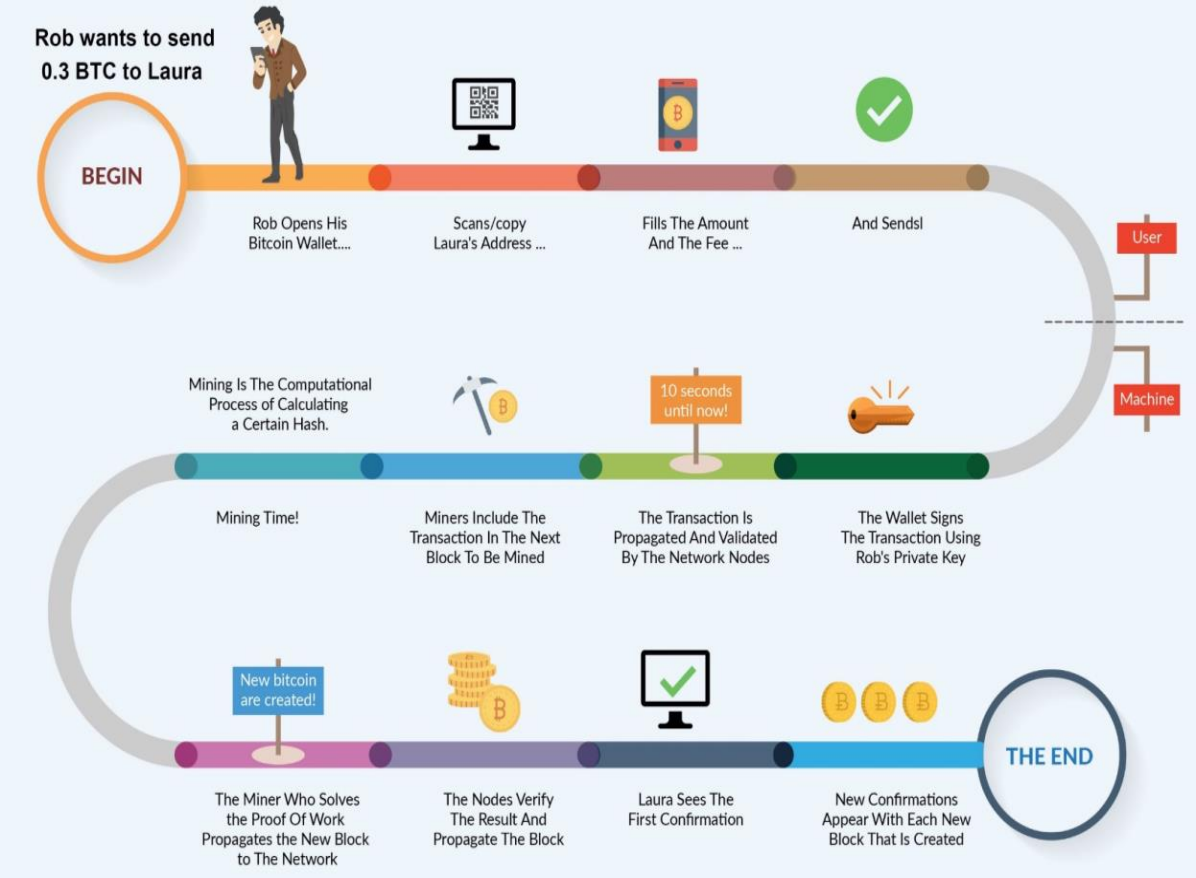
transferred the same coins to two people. Therefore, we need a mechanism for the payee to ensure that the previous coin owner did not transfer the coins to others before that. To ensure this, the only way is to let everyone know all transactions. In the absence of a trusted third party, to achieve this, all transactions must be broadcast to everyone. Therefore, we need a system where all participants agree on the order in which they receive the coins to form a unique sequence record history. This is actually a problem of distributed consensus.

The solution provided by the Bitcoin system is using a timestamp server composed of all nodes. The timestamp server timestamps the hash of transaction blocks, and broadcasts it. Each timestamp contains the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it. To implement a distributed timestamp server in a peer-to-peer network, Bitcoin uses the Proof-of-Work (PoW) mechanism. When a hash operation is performed, PoW needs to find a certain value so that the first few digits of the overall hash value are all zero, and the average workload increases exponentially as the number of zero digits increases. In addition, the hash has no rules. To ensure that the first few digits of the hash are zero, the only way is to force random trial and error over and over again. Once sufficient CPU computational power is consumed and the hash value that meets the condition is found, the block cannot be changed, unless CPU is used again to redo it.

In addition, PoW solves most decision-making problems. In Bitcoin, the longest chain represents most decisions. If most of the computational power is controlled by honest nodes, the honest chain will grow rapidly and surpass other chains. If an attacker wants to change a previous block without proper authorization, the attacker must redo the PoW tasks of the corresponding block and all its subsequent blocks, and then catch up with and surpass honest nodes. This is very difficult. Mathematically, it is not difficult to prove that with the increase of the number of nodes later, the probability for slower attackers to catch up with the honest nodes decreases exponentially. Generally speaking, it is almost impossible to catch up after 6 blocks. In addition, the difficulty of PoW tasks is not fixed, but it is dynamically adjusted using the Moving Average method, which mainly takes into account the increase of the hardware computing rate and the increase or decrease of the number of miners. If the computing rate is fast, the difficulty increases, the computing rate is slow, and the difficulty decreases. Through dynamic adjustment of the difficulty, the block generation time of Bitcoin is roughly stabilized at about 10 minutes.

The entire network runs as follows:

1. New transactions are broadcast to all nodes.
2. Each node packs the received transactions into a block.
3. Each node performs the PoW task by constantly changing the nonce for the block, to make the hash of the block meet the specified conditions.
4. Once a node completes the PoW task, it broadcasts the block to all other nodes.
5. After receiving the block, other nodes verify the validity of transactions within the block, and accept the block if the verification passes.
6. How does a node express its acceptance of the block? That is, when the next block is added, the hash value of the accepted block is taken as the previous hash value of the next block.



Bitcoin transaction process

Details of the transaction and mining are not described in detail here. But, in short, what we can say is that, in Bitcoin, the longest chain of information always prevails. If a node finds a chain longer than itself, it will automatically switch to the longest chain to work.

Since the cost of PoW is high, how can we encourage everyone to contribute their computational power and become nodes to ensure the safety of the entire Bitcoin network? Bitcoin provides two incentive policies:

1. A node digging out a certain block will obtain a certain amount of bitcoins, which is actually the only bitcoin issuing mechanism (primary market). All bitcoins can only be dug out through mining and then enter into circulation.
2. Miners can obtain a certain amount of service fees for processing the transaction information, which is actually the circulation of existing bitcoins (secondary market). When 21 million bitcoins are all dug up, users can only be motivated by service fees.

These incentive policies also implicitly encourage the node to be honest. If a greedy attacker has more than half of the CPU computational power, he has to make a choice: whether to change the transaction records and transfer back the bitcoins he has spent, or to dig for new coins and earn service fees honestly? It is very likely that it is more advantageous to mine honestly. After all, the coins that he can earn are more than the total coins earned by all other nodes. While, destroying the Bitcoin system will also undermine the effectiveness of his own wealth. If the bitcoin is no longer reliable, the value will collapse quickly. In addition, an attacker is not as free to manipulate, change without authorization, or forge transaction records as one might think. All he can do is steal back the bitcoins he has recently spent.

Why does PoW Work?

Bitcoin has been running steadily for 10 years without any organization or group to maintain it, relying only on voluntary maintenance of community volunteers. No major problems have ever occurred during this period. This is a miracle and is sufficient to prove the effectiveness of the consensus mechanism behind Bitcoin. Why can Bitcoin do this? Why is the consensus mechanism behind Bitcoin so effective? The Bitnodes data shows that the number of Bitcoin nodes exceeds 10 thousand (Bitcoin has many types of nodes, and the number may be different for nodes with different calibers. Only full nodes are considered here). Why can Bitcoin coordinate tens of thousands of nodes in a permissionless network environment?

In my humble opinion, the following reasons may apply:

- **Effective incentive policies**: The incentive policies effectively encourage more nodes to participate in the Bitcoin peer-to-peer network. The more nodes, the safer the Bitcoin network.
- **PoW**: Mining and generating blocks consume CPU computational power, which artificially creates obstacles and increases costs, thereby increasing the cost of attackers.
- **Game theory**: Incentive strategies also take into account the game balance, so that rational nodes benefit more from keeping honesty.
- **Communication efficiency**: The communication efficiency between Bitcoin nodes is not low. You may notice that the broadcast of transactions and blocks is also involved. However, such broadcasts are not broadcast between two nodes, instead, information is broadcast to all other nodes by a certain node (the node where the transaction occurred or the PoW is computed). In addition, the broadcast for transactions is not required to reach all nodes. As long as many nodes accept the broadcast, it will be packaged soon. In 2014, Miller and others (Anonymous Byzantine Consensus from Moderately-Hard Puzzles: A Model for Bitcoin) strictly proved that the message complexity does not increase with the network size, but is a constant. In addition, the broadcast for blocks also allows message loss. If a node does not receive a block, it will realize that it has missed the previous block when it receives the next block, and actively request the block from other nodes.
- **Probabilistic consistency**: Compared with other consensus algorithms, the Bitcoin consensus mechanism is most special in that it does not pursue deterministic consistency, but pursues probabilistic consistency. When a block is just dug up, the transaction information it contains is not confirmed by all nodes, and the data it contains is not the result of eventual consistency, which may still be changed by attackers. However, as the number of nodes increases later, the probability of authorized changes decreases exponentially, and the probability of eventual consistency increases significantly. Once the number of subsequent nodes exceeds 6 (that is, after about 60 minutes), the consistency can be regarded as deterministic and final.

The Bitcoin consensus mechanism is no longer confined to the level of distributed algorithms, but contains more ideas, such as economics, game theory, and probability theory. Therefore, it may be more appropriate to call it the consensus mechanism. However, we can still put the Bitcoin PoW consensus mechanism into the framework of consistency. From the perspective of FLP and CAP:

1. Bitcoin considers Crash Fault Tolerance and Tolerance of Network Partition to the maximum extent, which is also a necessary requirement for the network openness. The open network environment is extremely complex and open to everyone at any time, nodes are distributed all over the world, and machine faults, network differentiation, and system attacks may occur at any time, so fault tolerance must be considered. Using the PoW mechanism, Bitcoin can not only achieve Crash Fault Tolerance, but also can achieve Byzantine Fault Tolerance in combination with cryptographic asymmetric encryption technology, to defend against unauthorized changes and attacks.
2. Bitcoin ensures liveness and availability as much as possible. The block generating time of Bitcoin is always about 10 minutes, which means the system can always reach an agreement within 10 minutes. The Bitcoin network has never been down in the last ten years. From this perspective, it has indeed achieved the ultimate availability. However, it must be pointed out that the availability of Bitcoin is quite different from the availability of the Internet field we generally understand. For the system availability in the Internet field, it is not only required that the system runs stably without downtime, but also has clear requirements for service experience, such as response time. If you use Alipay to transfer money, but it is not available at any time and the money does not arrive in 3 seconds, instead, you are told that the system is busy and needs to wait for 10 minutes or even 30 minutes, then the service will be considered unavailable. However, this phenomenon has been happening in Bitcoin. Bitcoin has a block every 10 minutes, and the block size is only 1 MB, which cannot hold too many transactions. If too many transactions exist at the same time, you can only wait until these transactions can be packed into the next block. Therefore, it may take 20 minutes, 30 minutes, or even longer. From this perspective, Bitcoin network actually relaxes the requirements for response time and achieves relatively basic availability: The read availability is extremely high, while the write availability is very low.
3. Bitcoin no longer pursues Determinacy for safety and consistency, but uses probabilistic assurance, which can basically be considered to ensure the eventual safety and consistency, but the "final" here is still time-conditioned and probability-based. For example, if I have just transferred you a bitcoin, no one can say that the result is a definite and eventual one. However, as time goes by, new blocks are dug out continuously, and the transaction information of "I transferred money to you" will also be confirmed by more nodes and reinforced by more subsequent blocks. The probability of certainty of this result is constantly increasing. Once enough time passes (for example, 1 hour), we can consider, from the probability perspective, the probability of the result being changed with is extremely low, and the probability of the system reaching the eventual consistency is extremely high. In practice, we can think that the system ensures the eventual consistency.

On the whole, the Bitcoin PoW consensus mechanism has achieved a good compromise and trade-offs under the limit of FLP and CAP. In practice, it does provide a feasible solution to the distributed consensus problem in open and complex networks. The stable and reliable operation of Bitcoin over the past decade has also proven this point.

In addition, Bitcoin PoW has been rigorously analyzed and proved in a study by Miller and LaViola to have the following characteristics:

- A Bitcoin network can be viewed as consisting of approximately infinite nodes. Each node contributes a small amount of computational power and correspondingly each node has a low probability of creating blocks.
- The PoW mechanism relies on the synchronous network model. In this model, if the network latency is 0, the mechanism can tolerate 50% faults. However, judging from the actual observed network latency, Bitcoin can tolerate 49.5% faults. If the network latency is equal to the block generation time (10 minutes), only 33% of faults can be tolerated. If the network latency is close to infinity, the fault tolerance of the mechanism also approaches 0.
- The Bitcoin PoW mechanism is scalable, because the consensus time and message complexity are not related to the network size (number of nodes in the network), but only to the computational power of the wrong node, which can be considered as a non-dimensional constant.

The PoW mechanism is not only reliable in practice, but also can be tested theoretically The PoW mechanism uses the synchronous model and random probability to circumvent the impossibility theorem of FLP deterministic asynchronous model. Compared with the complexity of PBFT algorithm O(n2), the scalability of PoW independent of network size has great advantages: The more nodes, the safer the system. And, the efficiency of the system has not decreased.

What Exactly Is PoW?

What is the magic of the PoW mechanism? In fact, as you may be aware, the idea of PoW is not profound, and in fact it was not first proposed by Satoshi Nakamoto. As early as 1993, this idea was proposed to combat junk mail (Pricing via Processing or Combatting Junk Mail). However, it was not widely used until Satoshi Nakamoto created Bitcoin. The essence of PoW lies in intentionally creating obstacles and increasing the cost of participants, so as to minimize the malicious attempts of participants. For example, the requester is required to do some additional work to detect DDoS attacks and junk mail. For another example, it is very common that a verification code need to be entered when logging on to a website, which is also to increase the login cost and prevent the website from being attacked. The core features of such tasks are non-symmetric: For service requesters, it must be difficult to complete the task. For service providers, the verification task must be simple and fast. For Bitcoin PoW, it is non-symmetrical: It requires a lot of computational power to find a nonce (random number) that makes the hash conform to the condition, after continuous trial and error, while verifying whether the found nonce conforms to the condition only requires a simple hash operation verification.

Bitcoin PoW is essentially one-CPU-one-vote. Why CPU instead of IP address? This is still based on the difficulty of the task. If one-IP-one-vote is used, the system can be easily controlled by people with a large number of IP addresses (such as IP providers). Relatively speaking, at least at the time when ASIC or FPGA is unavailable, the CPU was still relatively expensive hardware, and it was not easy to have a large amount of computational power (CPU + Electric power). This actually implicitly provides a real-world anchor for the value of Bitcoin: The virtual currency system finds the value anchor of the real physical world through computational power, although it seems to many people that this consumption of computational power is meaningless and a waste of energy.

Many people are thinking about how to reduce Bitcoin mining costs. This kind of thinking certainly has positive significance. The cost of PoW needs to be appropriate: If the difficulty and the cost are both too high, it indeed will waste more energy, but the security of the Bitcoin network has also been improved. If the difficulty and the cost are both too low, the purpose of preventing attacks will not be achieved, and the security of the Bitcoin network will be reduced. This is actually a trade-off issue and also a subjective value judgment, depending on the public's understanding and positioning of Bitcoin. Value judgment is always full of subjective prejudice. At present, the debate on Bitcoin is so great, because the public has not yet reached a consensus and is yet to come up with a common vision for the future of Bitcoin.

In short, Bitcoin PoW is a complete set of mechanisms, including technical trade-offs, and economic and gaming considerations, which jointly ensure the security and reliability of the Bitcoin network.

Limitations of the PoW Mechanism

Everything is not perfect, and the PoW mechanism also has its limitations without exception. In fact, we can know a thing or two from the many criticisms of Bitcoin. Generally, the PoW mechanism is considered to have the following limitations:

1. **High costs and energy waste**: Critics of Bitcoin's waste of energy are endless. According to Digiconomist data, the annual electricity consumption of Bitcoin is basically the same as that of New Zealand, and is also equivalent to 1/5 of the power consumption of Australia. The cost of each bitcoin transfer transaction is three times that of every 100,000 visa transfer transactions. Although this comparison is sometimes unfair (bitcoin transactions are liquidation, while visa transactions have additional liquidation costs in addition to transaction costs), many people disagree. As mentioned earlier, this is also a subjective value judgment, but it is, after all, a point of view and sometimes a real pain point. For example, I'm afraid no one is willing to buy a cup of coffee with bitcoins, because the service fee may be higher than that of coffee. The "culprit" is the CPU computational power consumption required by the PoW mechanism. Therefore, some people constantly try to improve, and even put forward new solutions.
2. **Low efficiency**: We are used to the convenience of the Internet, and get used to second-level transfers and million-level TPS. However, for bitcoin transactions, we may have to wait for tens of minutes, and only 7 transactions are supported per second. We are not very satisfied. This comparison is also not fair. The banking system has only a few data centers and a maximum of hundreds of machines in the background, and the transaction only enters one of the machines, so the eventual consistency is ensured in the liquidation process afterwards. While, Bitcoin has no single point. It coordinates tens of thousands of machines, and the transaction is the liquidation. However, this inefficiency is indeed a fact, and some people are constantly trying to improve it. For example, the size limit of each bitcoin block is increased, to allow each bitcoin block to pack more transactions. This is what Bitcoin Cash does. For another example, the block generation time of Bitcoin is shortened to allow faster block generation. This is what Litecoin does. Even so, due to the huge PoW cost required by the PoW mechanism to ensure network security, improving the network efficiency is difficult.

3. **Centralization risk**: With the emergence of specially crafted mining chips, such as ASIC and FPGA, it is almost impossible for ordinary personal PCs to dig up bitcoins. Mining is increasingly concentrated among giant companies with the ability to research and develop chips, and the emergence of the mining pool (in order to smooth the earnings, a large number of nodes form an alliance to jointly mine and share the benefits equally) has also intensified this trend. In addition, the increase of the bitcoin block size limit will also lead to the need for a large storage space to run all the Bitcoin nodes, so that Bitcoin cannot run on an ordinary PC, but only on a special large computer. These centralization tendencies undoubtedly damage the security of the Bitcoin network. After all, the security of the Bitcoin network consisting of ordinary PCs around the world is much higher than that of the Bitcoin network directly or indirectly controlled by several giant companies. Although the dispute over this issue is even greater and different people have different views, many people are still trying to find new solutions.

PoS

Among these new solutions, Proof-of-Stake (PoS) undoubtedly attracts the most attention. Also facing the consistency issues in open and complex networks, it proposes a brand-new solution.

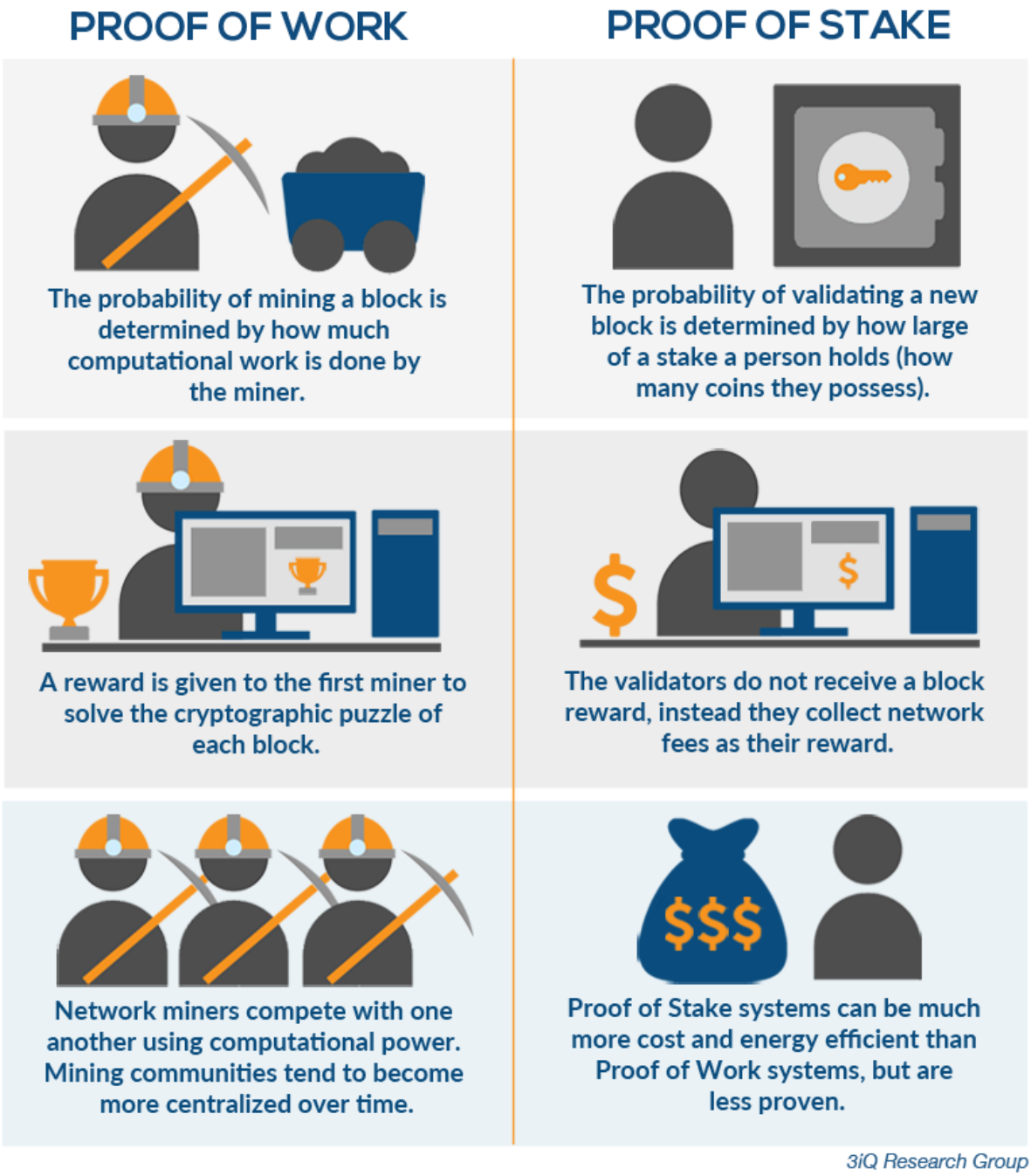
Basic CcON

In 2011, a user named QuantumMechanic took the lead in putting forward the idea of proof-of-stake in BitcoinTalk Forum. After that, the idea has been continuously developed and perfected, and has been trusted by more and more people.

The basic concepts of PoS are as follows:

- All nodes are no longer competing for mining at the same time, but each time only one node is used as the verifier: In the Bitcoin network, all nodes need to perform PoW tasks, that is, all nodes need to perform complicated hash operations, and consume a large amount of CPU computational power, while only the node that finds the answer first can be rewarded. This simultaneous competition between all nodes will undoubtedly consume a large amount of resources. So, can only one node work at a time? If so, how will the lucky one be chosen? In PoS, mining or miners are no longer needed. Instead, only one node needs to be selected at a time as the validator to verify the validity of the block. If a node is selected as the validator to verify the next block, it verifies whether all transactions in that block are valid. If all transactions are verified to be valid, the node signs the block and adds it to the blockchain. In return, the validator will receive transaction fees related to these transactions. In PoS, only one node works in each consensus, and the work is very easy, thus achieving the purpose of saving resources.
- To become a validator, a deposit must be provided: To prevent the validator from acting maliciously, a node must deposit tokens into a designated account in advance as the security deposit or mortgage guarantee to become a validator. Once the node is found to be acting maliciously, the security deposit will be confiscated, thus encouraging honest work. As long as the benefits from malicious actions do not exceed the deposit limit, the node will be honest.

- Being elected as the validator is not completely random, but is proportional to the deposit amount provided. For example, Alice provides a deposit of 100 coins, while Bob provides a deposit of 500 coins, then the probability of randomly selecting Bob as the validator to produce the next block is 5 times greater than Alice. This is actually similar to a joint-stock company, which divides rights, such as the right to speak and the beneficial right, according to the proportion of capital contribution. Major shareholders contribute more, bear more responsibilities, and have greater corresponding returns.



[PoW and PoS comparison diagram]

It is easy to see that PoS also adopts ideas from economics and game theory, and ensures the security and reliability of the network through incentive policies and punishment mechanisms.

Why does PoS Work?

The PoS protocol is still in line with the conclusions of the traditional Byzantine Fault Tolerance (BFT) algorithm. Currently, the research on PoS can be divided into two main lines: a line for the synchronous network model, and another line for the partial asynchronous network model. The chain-based PoS algorithm is almost always dependent on the synchronization network model, and its effectiveness and security can be strictly proved like the PoW algorithm. For reference, see this document.

In addition, from the CAP perspective, the chain-based PoS algorithm is similar to the PoW algorithm, which also achieves fault tolerance as much as possible, and availability is more ensured between availability and consistency.

If the traditional consensus algorithms (Paxos, Raft, and PBFT) implement deterministic finality or consistency, then PoS is similar to PoW, and seeks probabilistic eventual consistency instead. From the perspective of traditional CAP, this is actually a weakening of consistency. However, from the perspective of practical feasibility, it is also a brand new thinking and breakthrough.

From the design strategy of PoS, it can be divided into two types. For reference see, this paper.

- The first type is the chain-based PoS mentioned earlier. It mainly imitates the PoW mechanism, and simulates the mining process by pseudo-randomly assigning the block creation right to stakeholders. Typical representatives include PeerCoin, and Blackcoin. Its security and effectiveness can be viewed from the analogy of PoW.
- The other type is BFT-based PoS, which is based on nearly 30 years of BFT consensus algorithm research. The idea of designing PoS based on the BFT algorithm was initially proposed in Tendermint. Casper in Ethereum 2.0 also followed this tradition and made some modifications and improvements. The security and effectiveness of this type of PoS algorithms can be seen by referring to the BFT algorithm. For example, it can be proved mathematically that as long as more than 2/3 nodes of protocol participants follow the protocol honestly, the algorithm can ensure that no conflicting blocks exist in the final state regardless of the network latency. However, such algorithms are not perfect either, especially for 51% attacks, which has not been completely solved. Currently, this field is still in the open exploration stage.

Debate over PoS

The idea of PoS is not complex, and what is more easily criticized is precisely the system that is similar to the real world and obtains income according to the proportion of capital contribution. People are already wary of the Matthew Effect in the real world. This system leads to the result that the rich get richer and the poor get poorer: People with more tokens will have more opportunities to become the validator, thus participating in the network and earning more.

However, the views on this issue are very controversial, and many people have put forward completely different views, believing that PoS is fairer and more conducive to countering the centralization trend than PoW. The main reason is that PoW mining relies on physical hardware and electric power resources in the real world, which easily leads to Economies of Scale. Companies that buy 10,000 mining machines have more bargaining power than individuals who buy 1 mining machine, and even can develop their own mining machines with lower costs. And, mines with 10,000 mining machines have higher bargaining power over electricity charges, and can be migrated near power stations in countries and regions with low electricity charges, or even build their own power stations with lower costs. The consequence is that the larger the organization, the lower the comprehensive mining cost, which is exactly what has really happened in the real world. In contrast, PoS does not need to rely on hardware in the real world, and has no Economies of Scale advantage. If price manipulation is not taken into account, the price of 1 coin and the price of 10,000 coins increase linearly. From this perspective, PoS may be fairer and more conducive to decentralization.

Another concern for PoS is its security. After all, PoS no longer performs complex CPU operations like PoW to prove itself. In PoW, if an attack is to be launched, 51% of the computational power needs to be controlled (recently, some studies have found that only 25% of the computational power is required to make the attack possible), which means that most mining machines and computational power resources are required. In PoS, if the whole system is to be controlled, 51% of all tokens are required. Which one is safer? In fact, it is not easy to say, but from a real-world example, if the Bitcoin algorithm is switched to PoS, it takes about half of the Bitcoin market value to control the Bitcoin system, which is about 40-160 billion US dollars (the price range of a bitcoin: 5,000-20,000 US dollars). This figure is far higher than the cost of mining machines. It is almost impossible to launch an attack with such a large amount of money. In this regard, PoS may be safer.

In addition, due to the low cost of PoS deployment (low hardware requirements), tokens can easily be forked in the real world, resulting in a pile of altcoins. This problem does not exist in PoW. PoW relies on hardware for mining, so it is easy to change a parameter of Bitcoin. However, if you really want to run it, it requires a lot of computational power and the support of a large number of miners. For example, forking Bitcoin Cash from Bitcoin has experienced twists and turns. PoS has no such concern at all. Anyone can download the open source code and change it at will. By winning the support of a few nodes, they can claim that they have created a brand new token. For example, dozens or hundreds of altcoins can be easily forked from EOS (a token name), each of which claims to be unique. This is indeed a fact, but it is not easy to say whether it is good or bad.

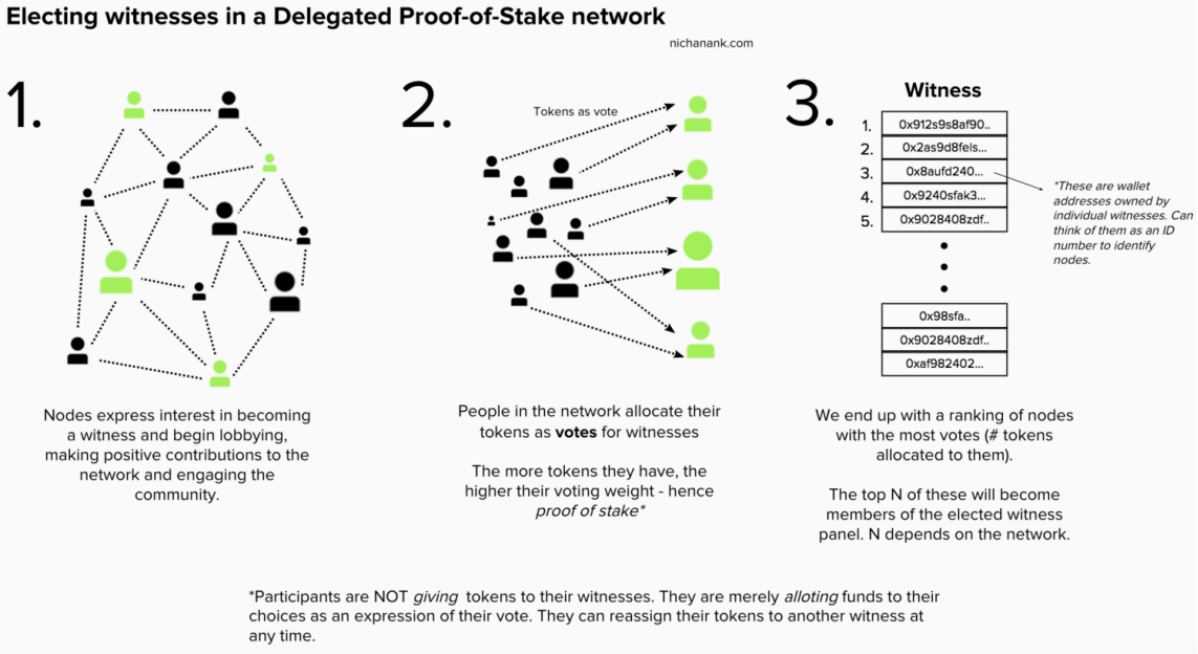
PoS: Improvement and Optimization

The most important part of the PoS mechanism is the selection mechanism of the validator or creator in the next block. Who will be the lucky one? The above-mentioned selection according to the proportion of account funds and the probability is actually the simplest way. This method is indeed easy to lead the rich to obtain the income once and for all, thus compromising the enthusiasm of other participants in the network. Currently, many ideas are available to ameliorate this problem, among which the more interesting one is the coin age-based method. When the creator is selected, not only the amount of funds, but also the coin age will be considered. The so-called coin age refers to the retention time of the coins on an account. For example, if 1 coin is transferred to a specified account for 10 days, it can be

considered that the coin age is 10, and the coin age will be recomputed from 0 every time the account in which the coin is located changes. In this way, it is possible to restrict the nodes with large amount of funds from becoming the creator frequently. For example, it can be set that only nodes with a coin age of 30 will have the chance to become the creator, and the coin age will be cleared immediately after the node becomes the creator. This actually limits the interests of large participants, and provides more opportunities for small and medium-sized participants.

The famous solution improved base on PoS is Delegated Proof-of-Stake (DPoS), in which the proxy delegation mechanism is adopted. In DPoS, it is no longer possible for all nodes to become the creator. Instead, the nodes vote for each other, and only the nodes with the highest votes can participate in the block creation process. The details are as follows:

- The duties of proxies include ensuring the continuous operation of their own nodes, collecting transaction information and packaging it into blocks, verifying signatures and broadcasting blocks, and solving possible consistency problems in the network.
- For most DPoS chains, all token holders in the network can vote for the proxy, and the voting right is proportional to the number of tokens held. Token holders can also delegate their voting rights to others to vote on behalf of them instead of directly voting.
- Voting is dynamic and variable, which means that a proxy may be elected in or out at any time. Once a proxy is found to be malicious or fraudulent, he/she will lose the income and reputation. This serves as an incentive for the proxy to be honest and ensure network security. The proxy can distribute the received block rewards to the users who vote for him in proportion (this is actually equivalent to vote buying, which is not allowed in some scenarios).
- Unlike traditional PoS, proxies no longer need to hold large amounts of tokens, but must compete with each other to win votes from tokens holders.
- DPoS limits the number of verifiers in transaction blocks, which sacrifices the decentralization to a certain extent but increases efficiency, because the workload required to reach a consensus on the network is greatly reduced.



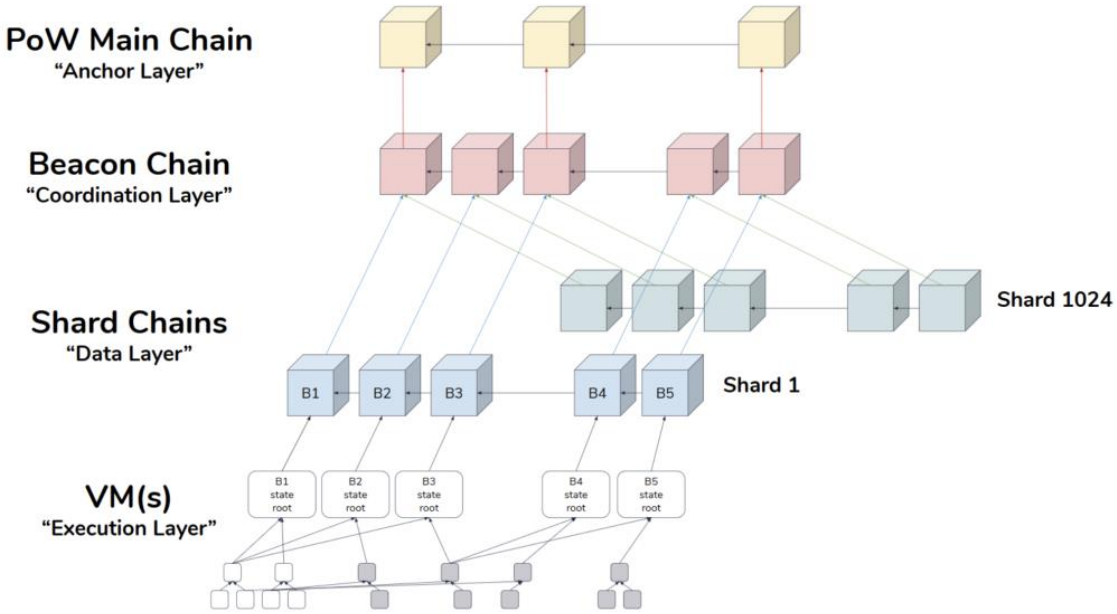
The process of electing the validator/witness in DPoS

It is not difficult to find that DPoS has ensured the extensive participation of nodes as much as possible by introducing the voting mechanism. And, the system is as efficient as possible by limiting the number of validators (generally 21-101). Despite the great controversy, DPoS is still a feasible solution, and more and more blockchain systems are also trying to improve and explore it.

Application in the Public Blockchain

In the public blockchain, many projects adopt the PoS mechanism. The famous ones are:

- Ethereum: At present, Ethereum still uses the PoW mechanism. However, Vitalik Buterin, the founder of Ethereum and a leader in the public blockchain field, is more favored for the PoS mechanism, and have also elaborated on the design philosophy of PoS (See this blog) and its advantages over PoW (See this article) many times. Ethereum is currently developing the PoS-based Casper protocol (See this paper for reference), which is expected to be released in the second half of this year. This shift from PoW to PoS also marks the beginning of the 2.0 era in Ethereum. As shown in the following figure, in the Ethereum 2.0 PHASE0, the PoS Beacon Chain using the Casper protocol will be released as the Coordination Layer (See this website for reference).



Ethereum 2.0 layers and phases

- EOS: Daniel Larimer, the initiator of DPoS, launched the EOS public blockchain project, in which many nodes will compete with each other, hoping to become one of the 21 supernodes with accounting rights. This design similar to the real-world Council system has aroused great controversy, and the supernode election may also contain huge commercial interests, which have gone beyond the scope of technical discussions and will not be discussed here.

Proof of X?

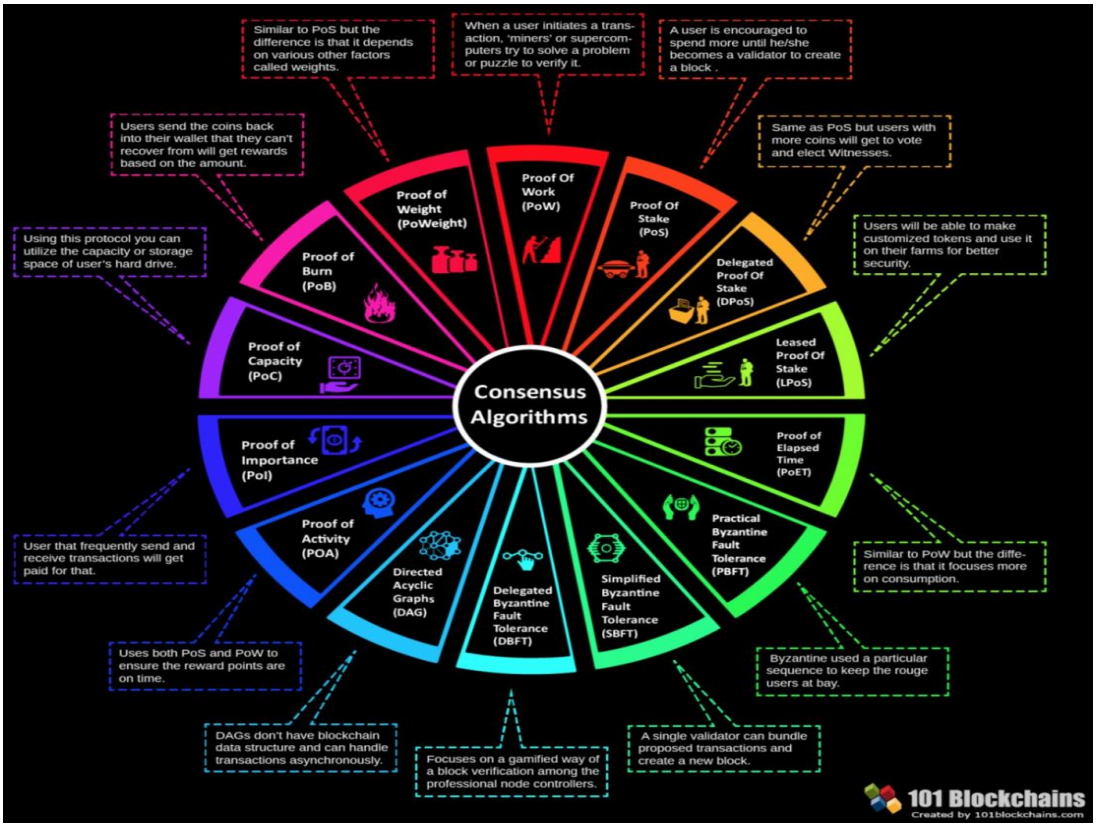
In fact, the rise of PoS mechanism is not only due to its own features, such as low cost, high efficiency and decentralization, but also because it opens the door to a wider array of techniques that use game-theoretic mechanism design in order to better discourage centralized cartels from forming and, if they do form, from acting in ways that are harmful to the network.

With the rapid development of blockchain, especially the public blockchain, in recent years, other Proof of X? Such mechanisms are also emerging. From the many mechanisms above, we can see the shadows of the PoS concept, that is, how to design a system from the economic and game theory perspectives to ensure decentralization, security and efficiency as much as possible. The following is a brief description of these mechanisms:

- **Leased Proof of Stake (LPoS)**: Many nodes with very few tokens can lease tokens to other nodes, thus forming a joint force and increasing the probability of becoming the validator. Once the node wins the election and obtains the reward, the service fees will be distributed proportionally, which is actually similar to the idea of the mining pool.
- **Proof of Elapsed Time (PoET)**: All nodes must wait for a certain amount of time to become the bookkeeper, and the waiting time is completely random. To ensure fairness, the two core questions are: how to ensure that the waiting time is indeed completely random? How to ensure that a certain node really waited for the specified time? The current solution relies on the special CPU hardware of Intel, the Intel SGX

system. Currently, it can only be applied in the permissioned network environment, such as the aforementioned Enterprise Ethereum Alliance (EEA).

- **Proof of Activity (PoA)**: PoA combines both PoW and PoS ideas. In PoA, the initial process is similar to that of PoW. Miners are still competing to solve the problem and dig the mine, except that the dug block only contains the header information and the address of the miner. Once a block is dug up, the system automatically switches to the PoS mode. The block header information points to a random stake holder, which verifies the pre-mined block.
- **Proof of Importance (PoI)**: In view of the fact that PoS mechanism tends to encourage people to hold coins instead of circulating them, and tends to lead to the problem that the rich get richer, PoI has included more dimensions in the importance of computing nodes to the system: In addition to the number of coins and the retention time of the coins on the account, counterparty (the more net transactions with other accounts, the higher the score), and the number and size of transactions in the last 30 days (the more frequent the transaction and the larger the amount, the higher the score) are also considered.
- **Proof of Capacity (PoC)**: It is also known as Proof of Space (PoS). The idea is similar to that of PoW, but it is measured by storage space instead of CPU computational power.
- **Proof of Burn (PoB)**: Miners must burn a certain amount of tokens, which means transferring a certain amount of tokens to an eater address (a Black Hole address, where tokens only go in but not out, that is, an address with an unknown private key) to prove themselves. Essentially, it is close to the idea of PoW. The difference is that PoW consumes computational power resources, while PoB directly consumes tokens.
- **Proof of Weight (PoWeight)**: PoWeight takes more weight factors into account based on the consideration of token amount in PoS. For example, FileCoin (the token on IPFS distributed file system) takes into account the size of IPFS data you have. Other weighting factors include but are not limited to Proof-of-Spacetime (PoSt), and Proof-of-Reputation (PoR).

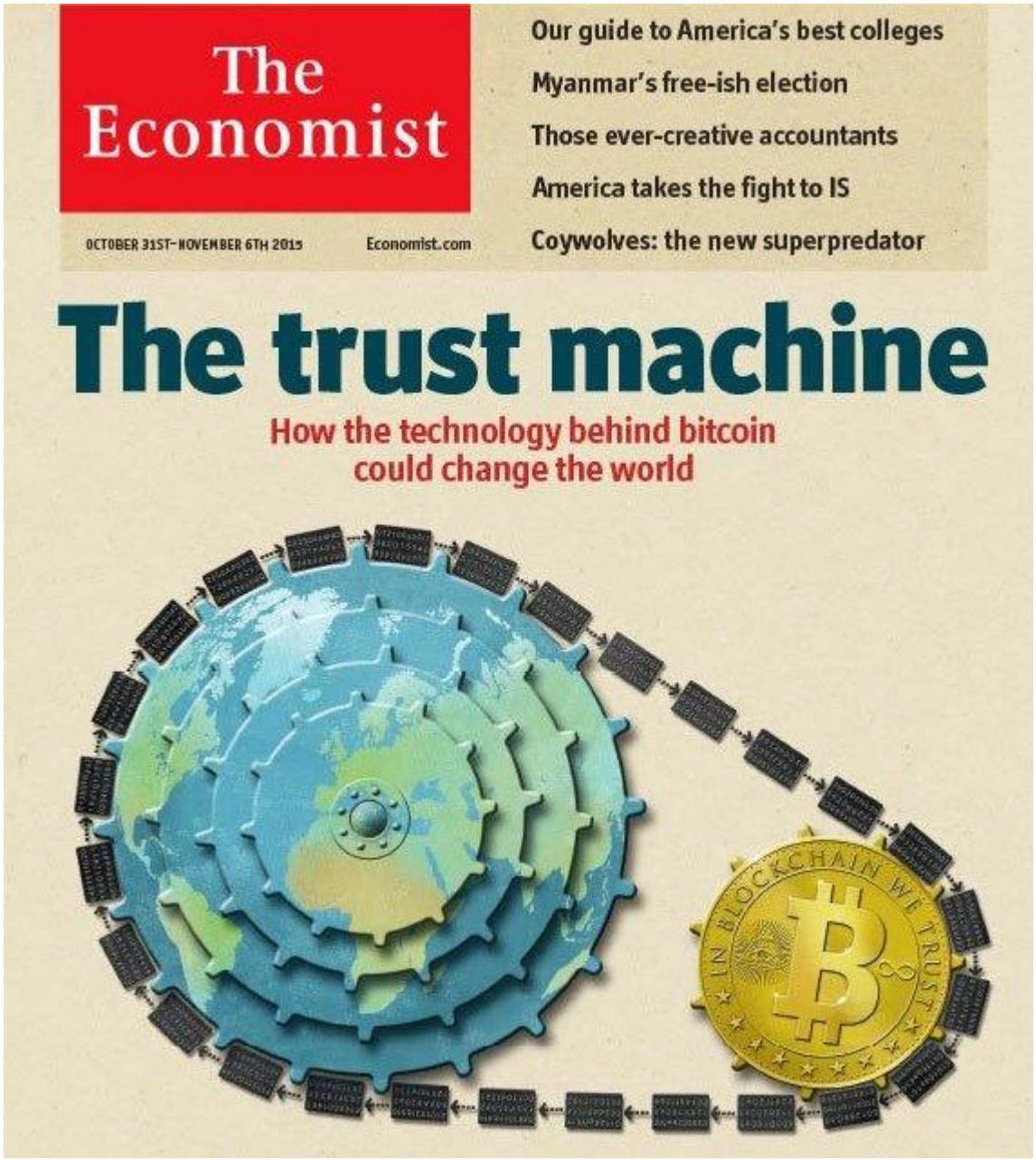


An Overview of consensus algorithms

It can be seen that although the Proof-of-X? Mechanisms emerge one after another and are different from each other, the core essential problem to be solved is the same: Who will be the lucky one to keep accounts? These mechanisms just adopt different policies to develop game rules, so that nodes can prove themselves as fairly as possible, and the lucky ones can be chosen fairly. All these policies, include CPU computational power, number of tokens held, storage space size, random wait time, number of tokens burned, node activity, and node contribution, are all explorations for consistency issues in open networks under specific scenarios.

All about Trust

From PoW to PoS, to Proof of "X (Everything you can think)", the issue of consistency in the permissionless network has been under exploration. The connotation of "consistency" is also changing, from how to prevent the faults of network and machine hardware to ensure the data consistency between network nodes, to how to prevent the malicious actions of people in the open network to ensure the true consistency of data between nodes. It can be said that we have moved from the reliability of hardware to the "reliability of people," and public blockchain technology is also regarded as the "trust machine." However, the issue of "reliability of people" is too complicated, and even beyond the technical scope. What can be done at the present stage is far from ensuring the "reliability of people." However, in most cases, it is still at the stage of people's trust in machines and in protocols. Fortunately, we finally took this step and began to face up to this thorny problem to explore innovative solutions.



[The trust machine]

Summary

The world is full of uncertainty, and the same is true for computer science. Since the emergence of computers, we have to face the uncertainty of machine hardware: problems that may arise from unexpected failures. And, since the rise of the Internet, we have to face the uncertainty of the network: the possible latency, disorder, and loss of communication messages. The most natural solution to the uncertainty problems is redundancy. A large number of nodes are used to ensure the overall security of the system, to avoid the single point of failure (SPOF) and enhance fault tolerance and attack defense capabilities. It is on this basis that large-scale distributed networks are booming. The way to find certain certainty between uncertain networks and nodes, and coordinate the consistency among many nodes, is exactly the problem that distributed consensus algorithms need to solve. The CFT algorithms capable of dealing with fault type errors include the most classic Paxos algorithm and the

simpler Raft algorithm. The effectiveness of the algorithm can be guaranteed when the normal nodes in the network exceed half. These algorithms are usually used in closed networks with trusted environment to coordinate the consistency between several to dozens of nodes, such as distributed storage, distributed service protocols, and distributed message systems within the company. In addition, they can also be applied to a private chain network composed of a few organizations that require authorization to access.

However, it is not only the network and the machine itself that are uncertain, but also the behavior of people controlling nodes in the network. The way to ensure the consistency of the distributed network under the condition that the attackers may change the data or attack the network without authorization, is exactly the problem that the BFT algorithm needs to solve. The most common BFT algorithm is the PBFT algorithm. The effectiveness of the algorithm can be guaranteed when the normal nodes in the network exceed 1/3. Even so, the ability of PBFT to deal with malicious behaviors in the network is still limited, and its performance will also decrease significantly as the number of nodes in the network increases. These limitations also lead to the PBFT algorithm can only be used in the permissioned network with more reliable environment, to coordinate the consistency between several to dozens of nodes, such as in the private chain scenario.

However, in a permissionless open network, the uncertainty problem is more severe, especially the uncertainty of the behavior of people behind the network nodes. The way to prevent the controllers in the network from forming a magnate through corruption and collusion, thus controlling more than half of the nodes in the network and achieving the purpose of controlling, damaging and attacking the network, is the problem to be solved in the open network. From this perspective, consistency in the open network also implies the premise of security: It requires not only that a consensus can be reached among nodes, but also that the consensus is indeed formed by the true expression of many node controllers. To achieve this consistency and security, it is not only necessary to realize the structural decentralization of physical hardware nodes, but also to ensure the decentralization of the actual controllers behind the nodes as much as possible. For this, it is necessary to: ensure that anyone can deploy and run the network protocol at any time to become a node in the network, and can access the network at any time; ensure peer-to-peer communication between nodes, without any centralized control nodes; and ensure that the roles of nodes are completely equivalent, and all nodes can participate in accounting fairly according to the rules. The way to coordinate the behaviors between tens of thousands of nodes in an open network to ensure the consistency and security of the network, is the problem to be solved by the public blockchain consensus mechanism. Among them, the most typical is the PoW consensus mechanism initiated by Bitcoin, and the subsequent PoS consensus mechanism. These consensus mechanisms are no longer limited to the technical consistency, but more of the ideas of economics and game theory are introduced to ensure the consistency and security of the network as much as possible from the perspective of economics and game theory.

From the consistency in the closed distributed network environment, to the consistency in the permissioned private chain scenario, and to the consensus mechanism in the permissionless public blockchain open network environment, the problems are becoming more and more complex, and the challenges are becoming more and more severe. From a purely technical perspective, the research on consensus is the same strain. These consensus algorithms or consensus mechanisms are also restricted by the FLP impossibility and CAP theorem in the traditional distributed consensus theory research. Paxos, Raft, and PBFT all emphasize fault tolerance and safety/consistency, and weaken liveness and availability. PoW and PoS

consider the problem from a brand-new perspective, ensuring fault tolerance, liveness and availability as much as possible, abandoning the pursuit of certainty for safety and consistency, and only pursuing the eventual safety and consistency in a probabilistic way.

In addition, the thinking on consensus is constantly deepening, from the simple data consistency between nodes to the emphasis on the consensus and identification between people behind the nodes, and from ensuring the reliability of the network and hardware to ensuring the reliability of the people behind the nodes that make up the network as much as possible. Although the reliability between people is very complicated and goes beyond the pure technical scope, it is gratifying that we are already on the road, and the ongoing innovative and active exploration in this field will certainly make the world more reliable.

References

1. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends
2. https://101blockchains.com/consensus-algorithms-blockchain/
3. Comparative Analysis of Blockchain Consensus Algorithms
4. https://draveness.me/consensus
5. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.67.6951&rep=rep1&type=pdf
6. https://dba.stackexchange.com/questions/18435/cap-theorem-vs-base-nosql
7. https://www.quora.com/What-is-the-difference-between-CAP-and-BASE-and-how-are-they-related-with-each-other
8. http://ug93tad.github.io/flpcap/
9. https://ramcloud.stanford.edu/~ongaro/userstudy/paxos.pdf
10. https://en.wikipedia.org/wiki/Paxos_(computer_science)
11. http://www.pmg.csail.mit.edu/papers/bft-tocs.pdf
12. https://www.youtube.com/watch?v=M4RW6GAwryc
13. https://medium.com/codechain/safety-and-liveness-blockchain-in-the-point-of-view-of-flp-impossibility-182e33927ce6
14. http://www.cs.utexas.edu/~lorenzo/corsi/cs380d/past/15S2/notes/week14.pdf
15. http://disi.unitn.it/~montreso/ds/slides17/10-pbft.pdf
16. https://eprints.soton.ac.uk/415083/2/itasec18_main.pdf
17. http://www.scs.stanford.edu/14au-cs244b/labs/projects/copeland_zhong.pdf
18. https://people.cs.umass.edu/~emery/classes/cmpsci691st/scribe/lecture17-byz.pdf
19. https://lamport.azurewebsites.net/tla/byzsimple.pdf
20. https://blockonomi.com/practical-byzantine-fault-tolerance/
21. https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274
22. https://bitcoin.org/bitcoin.pdf
23. https://en.wikipedia.org/wiki/Proof-of-work_system
24. https://bitnodes.earn.com/
25. Data Consistency and Blockchain
26. https://www.mangoresearch.co/understanding-blockchain-tech-cap-theorem/
27. https://cryptographics.info/cryptographics/blockchain/cap-theorem/
28. https://paulkernfeld.com/2016/01/15/bitcoin-cap-theorem.html
29. https://digiconomist.net/bitcoin-energy-consumption
30. https://www.economist.com/the-economist-explains/2018/07/09/why-bitcoin-uses-so-much-energy
31. https://www.youtube.com/watch?v=M3EFi_POhps
32. http://www.wisdom.weizmann.ac.il/~naor/PAPERS/pvp.pdf

33. https://bitcointalk.org/index.php?topic=27787.0
34. https://lisk.io/academy/blockchain-basics/how-does-blockchain-work/delegated-proof-of-stake
35. https://medium.com/loom-network/understanding-blockchain-fundamentals-part-3-delegated-proof-of-stake-b385a6b92ef
36. https://www.youtube.com/watch?v=Qfm26MX-Kdo
37. https://lisk.io/academy/blockchain-basics/how-does-blockchain-work/byzantine-fault-tolerance-explained
38. https://www.slideshare.net/oryband/the-stellar-blockchain-and-the-story-of-the-federated-consensusblockchain-academy
39. https://www.slideshare.net/sharkag/consistency-availability-partition-make-your-choice
40. https://fenix.tecnico.ulisboa.pt/downloadFile/1126518382178117/10.e-CAP-3.pdf
41. https://people.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
42. https://www.infoq.com/news/2008/01/consistency-vs-availability
43. https://www.giottus.com/Bitcoin
44. https://www.nichanank.com/blog/2018/6/4/consensus-algorithms-pos-dpos
45. https://en.bitcoinwiki.org/wiki/DPoS
46. https://cryptographics.info/cryptographics/blockchain/consensus-mechanisms/leased-proof-stake/
47. https://tokens-economy.gitbook.io/consensus/chain-based-trusted-computing-algorithms/poet
48. https://www.investopedia.com/terms/p/proof-elapsed-time-cryptocurrency.asp
49. https://www.investopedia.com/terms/p/proof-activity-cryptocurrency.asp
50. https://www.mycryptopedia.com/proof-of-importance/
51. https://en.wikipedia.org/wiki/Proof-of-space
52. https://en.bitcoin.it/wiki/Proof_of_burn
53. https://hackernoon.com/a-hitchhikers-guide-to-consensus-algorithms-d81aae3eb0e3
54. https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ
55. https://socrates1024.s3.amazonaws.com/consensus.pdf
56. http://groups.csail.mit.edu/tds/papers/Lynch/jacm88.pdf
57. https://en.wikipedia.org/wiki/Las_Vegas_algorithm
58. https://bitcoinmagazine.com/articles/selfish-mining-a-25-attack-against-the-bitcoin-network-1383578440/
59. https://medium.com/mechanism-labs/finality-in-blockchain-consensus-d1f83c120a9a
60. https://medium.com/@marchionnip/the-dlt-consensus-ecosystem-dff47d2cb926
61. https://pdfs.semanticscholar.org/da8a/37b10bc1521a4d3de925d7ebc44bb606d740.pdf
62. https://www.infoq.cn/article/5-consortium-blockchain-comparison
63. https://medium.com/@micobo/technical-difference-between-ethereum-hyperledger-fabric-and-r3-corda-5a58d0a6e347
64. https://medium.com/newcryptoblock/hyperledger-fabric-vs-r3-corda-7954035a4884
65. https://docs.corda.net/design/kafka-notary/decisions/replicated-storage.html
66. https://hyperledger-fabric.readthedocs.io/en/release-1.4/raft_configuration.html
67. https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf
68. https://medium.com/coinmonks/hyperledger-fabric-the-taste-of-raft-4f9f0df20b5e
69. https://medium.com/kokster/understanding-hyperledger-fabric-byzantine-fault-tolerance-cf106146ef43
70. https://fenix.tecnico.ulisboa.pt/downloadFile/282093452042936/alysson-bessani-2.pdf

71. https://hyperledger-fabric.readthedocs.io/en/release-1.4/orderer/ordering_service.html
72. https://entethalliance.org/wp-content/uploads/2018/05/EEA-TS-0001-0-v1.00-EEA-Enterprise-Ethereum-Specification-R1.pdf
73. https://medium.com/@VitalikButerin/a-proof-of-stake-design-philosophy-506585978d51
74. https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ
75. https://docs.google.com/presentation/d/1G5UZdEL71XAkU5B2v-TC3lmGaRIu2P6QSeF8m3wg6MU/edit#slide=id.g3aba11d29b_0_41
76. https://arxiv.org/pdf/1710.09437.pdf
77. https://arxiv.org/pdf/1607.01341.pdf
78. https://en.bitcoinwiki.org/wiki/DPoS
79. https://amplab.github.io/cs262a-fall2016/notes/21-Paxos-Raft.pdf
80. https://www.the-paper-trail.org/post/2012-03-25-flp-and-cap-arent-the-same-thing/