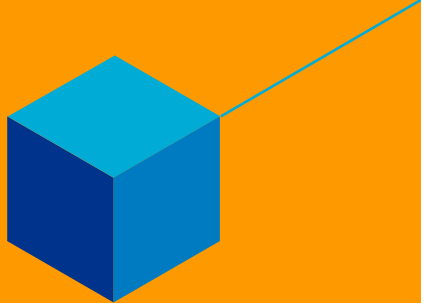
HELECLOUD

# Breaking BlockChain's Chains Part 1

## A Way for Designing Scalable Blockchain Solutions

Doichin Yordanov,
CTO Cloud Applications,
HeleCloud
07.2021



Blockchain's highly resilient mechanism for distributed consensus ensures the development of applications that manage reliable, trusted and secure peer-to-peer distributed state. This is at the heart of the incredible success of cryptocurrencies, however it is also its pitfall. The problem of energy overconsumption for Bitcoin mining, highlighted recently by Elon Musk's stark comments, is emanating from the problem of scaling blockchain's consensus mechanism on a global level. In this series of articles, we will review the relevant theory behind the problems of scalable thrusted distributed systems, will discuss their impact on blockchain architectures and devise a way of achieving scalable yet thrusted blockchain solutions.

# Part 1

## Consensus in Distributed Systems

Internal synchronisation between the nodes in a distributed system is critical for the consistency of its data. Consensus mechanism is an algorithm that works to achieve consistency among the majority of nodes. Distributed systems are posed to experience interruptions in internal communications, partitions caused by network or node failures due to incidents, attacks, equipment malfunction, bugs, etc. Their ability to cope with partitions dictates much of their performance characteristics.

The Consistency Availability Partition **(CAP)** Theorem classifies distributed systems according to how they behave when there are challenges to reaching to a consensus. It dictates that when a partition occurs during synchronisation, the system cannot reach uniform global consensus about given state, so it has two choices: to be unavailable (to a certain degree) waiting until the partition resolves; or to go with partial consensus with the caveat that the two sets of nodes, separated by the partition, may reach separately to conflicting consensuses. In other words, given a Partition tolerance, a distributed system cannot have both Availability and Consistency and needs to compromise one of them. It's never black and white, rather a spectrum of possible CA combinations, some worse, some good enough in practice. Amazon, for example, chose high availability with weaker eventual consistency, in order to enable clients to always make an order.This does not mean that the majority of orders are missed, however sometimes an order may be eventually canceled if conflicting concurrent orders grabbed all the items in stock.

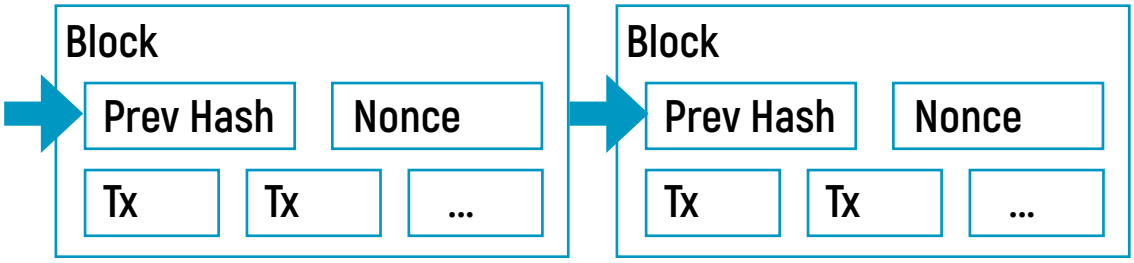
The **Paxos** algorithm is a key algorithm for consensus in distributed systems. It is named after the Greek islands when in ancient times the people dispersed on the separated lands, came with a clever sequence of steps needed to exchange messages with boats, so that all participants could agree on the timing of the next important democracy voting. The Paxos algorithm greatly increases the chance of a distributed system to reach a global consensus, eventually. Many modern distributed systems rely on Paxos or its variants. The **FLP** Theorem (In 1985, Michael J. **F**isher, Nany A. **L**ynch and Michael S. **P**aterson published a seminal paper referred to as **FLP**) proves, however, that despite all efforts a distributed system cannot be guaranteed to reach a global consensus all the time, i.e., there is always at least one sequence of events that can prevent it from reaching to a global consensus.

*The battle at Messembria, 812ac, the Bulgarian Khan Krum defeats the much larger Byzantine army due to its poor synchronization.*

The resilience of the consensus mechanism against malicious actors (and nodes) is a key characteristic of the overall reliability and security of a distributed system. A notorious problem for the Byzantine army was the consensus about the attack tactics during battle. Smart adversaries, such as the Bulgarian Khan Krum, used to bribe and/or eliminate just enough Byzantine generals who were responsible for synchronising an attack/retreat decision. Often even a single traitorous general was enough to throw the entire Byzantine army into chaos and lose the battle. **Byzantine Fault Tolerant** systems are those that can cope with a certain amount of missing or traitorous leader nodes for reaching the right consensus. Signing and stamping orders as they go through the chain of command was one way to overcome traitorous commanders and is still used today as a way for achieving eventual consistency in distributed systems, e.g Vector Clocks.

## Blockchain Architecture

Let's quickly review the basic blockchain terminology relevant to our topic.




Blockchain as the name suggests, stores transaction data in a chain of blocks. Each **block** contains transaction data and is stamped with a hash code, called **Nonce.** Given block points to the previous one by incorporating the hash code of the previous block as part of the current data to be hashed.

The chain is a form of Merkel hash trees that ensures an **immutable chain** of data. If the data in a block is altered, its hash is invalid and since the hash is replicated in the next blocks as part of their data, all the data in the subsequent blocks will be invalid. Essentially, a change in a block will require Nonce rehashing and renegotiation of all blocks down the chain, which is practically not feasible, unless the offending author owns all the nodes in the blockchain.

A blockchain system is a distributed system comprised of three distinct types of nodes; miner nodes, full nodes, and light nodes. **Miner** nodes can propose blocks and have the complete history of the blockchain. Full nodes have the complete history of the blockchain but cannot propose new blocks and light nodes rely on full nodes for blockchain's history. **Mining** is the process of producing new blocks. Miners compete with each other to bundle pending transactions in new blocks, calculate the Nonce so that it is smaller than the given system-wide threshold value and try to negotiate all other nodes to accept the new block as part of the chain.

**Smart Contract** is a block of code for reading, processing and adding blockchain data along with a trigger that determines when the code is executed. Smart Contracts are encoded and stored in the blockchain itself, the same way as transaction data is.

## Blockchain's Chains

The blockchain requirement for thrusted peer-to-peer state dictates an architecture heavily inclined towards high consistency, as CAP suggests transactions may be blocked until nodes in the system agree to add a new block. Furthermore, the consistency consensus should be reached in a Byzantine fault tolerant way, which imposes additional challenges to the system's availability as compared to the more light-weight Paxos consensus algorithm.

Let's drill down a bit into the consensus problem in blockchain – it is mining. A miner competes with other miners in the system: to first solve the nonce computational riddle while packing a bunch of pending transactions in a new block; then to negotiate the nodes in the system to accept its block, not the ones from the competitors. Checking a Nonce is a fast and cheap rehash operation, however, finding a Nonce has a random nature as it is computationally hard, hence it is unreasonable for an attacker to put enough computational power to solve the Nonce problem and repeatedly mislead the consensus and overall system's consistency. Solving the Nonce problem is called **Proof-of-Work** (PoW). The miner proves that it has put a legit amount of computational power to mine the block. Essentially mining ensures distributed transaction processing and is incentivised through awards. It is the original consensus mechanism in Bitcoin's blockchain.

The more computational power a miner has, the higher the chance to solve the Nonce, lead the consensus and get the reward. In effort to maximise profits, farms of mining nodes consume **large amounts of power.** Hence Musk's stark comments on the greenhouse emissions and Bitcoin's efficiency.

The **throughput** of a blockchain is another problematic area related to mining and computational heavy consensus. It is measured by the number of blocks it can add. As CAP dictates, the high consistency, achieved through computationally complex PoW for thrusted consensus, limits the availability of the system. The maximum throughput of Bitcoin is 1 block per 10 minutes and Ethereum improves that to 1 block in 15 seconds. Still, this is inadequate throughput for scalable global systems that need to handle millions of transactions per minute.

Another issue is the total **amount of data** that a blockchain system can store. The larger the blockchain system is and the higher the transaction throughput is, the higher the number of miners to cope with the incoming load. But the larger the number of miners, the harder it is to achieve consensus. Smart Contracts do not alleviate the situation, as they represent large amounts of data that need even more blocks to be added within the blockchain.

## Summary

In this first article, we reviewed the relevant theory behind the problems of scalable distributed systems and in particular thrusted peer-to-peer architectures. Then discussed their impact on blockchain.

In the next part, we will review ways to alleviate these problems; some are incoming as readily available blockchain offerings and some are up to the software engineers to tilt the software architecture for better CAP balance.