University of Milano-Bicocca
Department of Informatics, Systems and Communication
Computer Science Master Degree

Decentralized Wealth Distribution: A Simulation Study of Proof-of-Stake Algorithm Variations

Supervisor: Prof. Leporati Alberto

Master thesis by:
Poveromo Marco
Matr. 830626

Academic Year 2022-2023

Acknowledgments

Before proceeding with the discussion, I would like to dedicate a few lines to all those who have been close to me in this path of personal and professional growth.
I would like to thank my supervisor Prof.Alberto Leporati for his precious knowledge and availability.
I thank my university colleagues for accompanying me on this journey and for sharing great knowledge with me.
In conclusion, I cannot forget to thank my family and friends for always being there and supporting me unconditionally during these years.

Abstract

Consensus algorithms are fundamental to the security and integrity of blockchains, ensuring that all nodes in peer-to-peer systems maintain the same view to the blockchain ledger.
They determine how transactions are validated and how new blocks are added to the chain, agreeing to the current state of the blockchain ledger and the uniqueness of transactions.

Among the numerous consensus algorithms that have been proposed in literature, the so-called Proof-of-Stake is having a fair success, as it reduces the overall computational effort of the network (and therefore also the energy consumption) compared to the Proof-of-Work, currently used by most cryptocurrencies.

However, the Proof-of-Stake introduces its own set of challenges, including the 'rich getting richer' problem. This problem refers to the tendency for holders of larger amounts of resources (such as stake) to acquire even more resources, thereby increasing their influence and the risk of centralization.

Simulating these dynamics is important for understanding how parameters of the consensus algorithm impact the fairness of the process and for testing reward distribution strategies against the risk of centralization.

The purpose of this thesis is to study some possible variations of the Proof-of-Stake consensus algorithm, considering different possible initial wealth distributions and different possible rules for rewarding. In this regard, an agent-based python simulator of variants of Proof-of-Stake consensus algorithms has been developed by exploiting parallel computation techniques for better performance.

The goal is to identify variations that are sustainable in the long term, ensuring a certain level of democracy in the system and a fair distribution of wealth, using metrics based on the Gini coefficient and Lorenz curves.

Contents

List of Figures

Chapter 1

Introduction

The **blockchain** is a technology whose invention dates back to 2008, by the unknown author of the pseudonym Satoshi Nakamoto, who published a scientific paper entitled "Bitcoin: A Peer-to-Peer Electronic Cash System" [24]. Bitcoin was implemented the following year, the purpose of which is to store all cryptocurrency transactions of **Bitcoin**. The blockchain is therefore the technology on which Bitcoin is based, and which has revolutionised the financial sector.

Blockchain-based systems can be studied from different perspectives, one of which is that of **distributed systems**. Indeed, blockchain is a distributed system that inherits all the theory and concepts which have been obtained from decades of study in this area. It is important to note that blockchains do not automatically guarantee the data consistency, but rather fall under the AP (availability and partition tolerance) category of the CAP Theorem, in which nodes may not simultaneously see the same state of the blockchain.

A blockchain is a **distributed peer-to-peer** system that exploits decentralisation to enable participants to make decisions autonomously without the need for a central authority, such as a bank, that exercise centralised control over the network. The blockchain maintains a shared state through a distributed ledger consisting of a chain of blocks, where a block is linked with the previous through an hash reference of the previous block.

Participants in the blockchain network coordinate themselves through the use of a consensus algorithm, which allows nodes to agree on the state of the distributed ledger. Consensus algorithms are fundamental to the proper functioning of the blockchain and determine how transactions are validated and how a consensus is reached between nodes on the validity of a block. Two of the most popular consensus algorithms used in blockchains are **Proof of Work (PoW)** and **Proof of Stake (PoS)**. In PoW, miners must prove complex computational work to validate blocks and obtain block rewards, which represent an amount of cryptocurrency. In PoS, on the other hand, validators have to prove that they possess a quantity of cryptocurrency in order to obtain the rights to validate blocks and receive rewards.

Block rewards represent incentives for participants to contribute to the security and operation of the blockchain network. They encourage participants to invest resources, both in terms of computing power and cryptocurrency ownership, to play an active role in validating transactions and maintaining the network.

In conclusion, the blockchain is a distributed system that is based on decentralisation and uses consensus algorithms to reach agreement among participants on the validity of transactions. Consensus algorithms, such as PoW and PoS, are supported by miners or validators to obtain **block rewards**, which represent a form of economic incentive within the network. These elements contribute to the overall reliability and securty of blockchain.

This rest of this thesis is organized as follows:

- In **chapter 2** is described the **State of the art**, a literature review and analysis of the current state of knowledge of blockchain tecnology, consensus algorithms and block rewards redistribution in consensus algorithms.
- In **chapter 3** is shown the **Project**, the mathematical and theorical description of the model of the blockchain simulator, used for the block reward distribution analysis in several PoS variants.
- In **chapter 4** is shown the **Simulator**, the desing and the implementation of the blockchain parallel simulator coded in Python.
- In **chapter 5** is shown the **Results**, an analysis of the simulator outputs under various input conditions.
- In **chapter 6** are presented the **Conclusions**, with deductions that have been made on the results obtained.

1.1 Blockchain and distributed systems

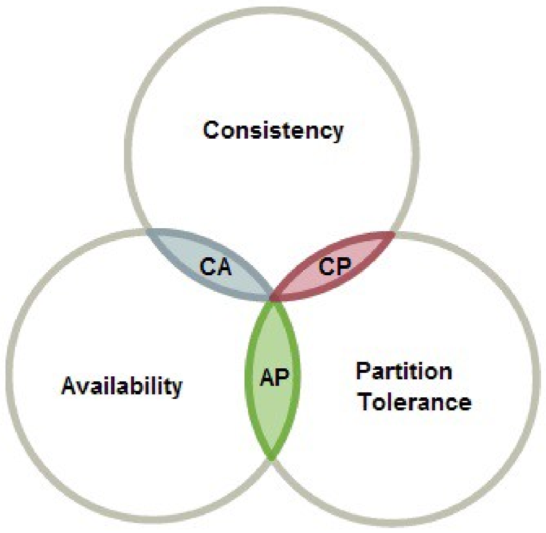
In order to introduce the description of a blockchain, it is necessary to introduce the concept of **distributed system**, since a blockchain is a distributed system.
A distributed system is a collection of independent computers that appear to the end user as one coherent system.

The aim is to offer a single view of the system that is actually composed of heterogeneous computers. The computers that are part of the distributed system are called **nodes**, and are able to communicate and coordinate by means of **message passing**, i.e. a type of communication between processes where there are no shared resources. Each node possesses its own memory and processor and may act honestly, they may be points of failure, or they may act maliciously. In the case of known not honest behaviour, they take the name **Byzantine nodes** whose origin is attributed to the Byzantine Generals problem [18].

Designing a distributed system poses many difficulties, one of the main challenges being to guarantee fault tolerance. In the event that one or more nodes fail or there is network partitioning, the distributed system should continue to function normally. However, the CAP theorem [5], introduced by Eric Brewer in 1998, shows that a distributed system cannot guarantee the following three properties at the same time: consistency, availability, and partition tolerance.

- **Consistency** is a property that ensures that all nodes in a system see the exact same copy of the data. In blockchains, data consistency is achieved through consensus algorithms, and is also called state machine replication.
- **Availability** is a property indicating that each node in the system is able to accept incoming requests and respond without failure when requested.
- **Partition tolerance** is a property that ensures that if one group of nodes is unable to communicate with another group of nodes (i.e. network partitioning occurs), the system should continue to operate correctly.

Figure 1.1: A Venn diagram of the CAP Theorem



The Figure 1.1 shows by means of a Venn diagram that a distributed system can only simultaneously satisfy at most two properties; according to the two properties it satisfies, it can be classified as AP, CA, or CP.

- A CP (consistency and partition tolerance) system does not guarantee availability.
- A AP (availability and partition tolerance) system does not guarantee consistency.
- A AC (availability and consistency) system does not guarantee partition tolerance.

Typically, blockchains do not guarantee consistency, and thus fall into the AP category of the CAP theorem. In this sense, data consistency is achieved over time (eventual consistency) and nodes may not see the same copy of the data at a given instant in time. This feature allows the blockchain to have temporary disagreement, which is however achieved over time through a process of **blockchain validation**.

1.2 Blockchain structure

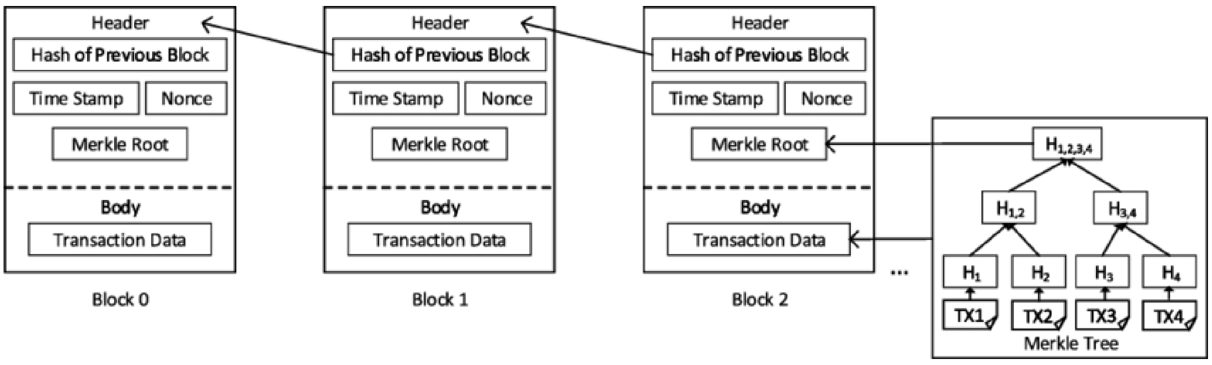
The **blockchain** is a technology whose invention dates back to 2008, by the unknown author of the pseudonym Satoshi Nakamoto, who published a scientific paper entitled

"Bitcoin: A Peer-to-Peer Electronic Cash System". [24].
Bitcoin was implemented the following year with the purpose of storing all cryptocurrency transactions of **Bitcoin** within a distributed ledger composed of a chain of blocks.

Technically, a blockchain is a peer-to-peer, distributed ledger that is cryptographically secure, append-only, immutable (extremely hard to change), and updateable only via consensus or agreement among peers (nodes).

A blockchain is therefore a distributed **peer-to-peer (P2P)** system, i.e. the nodes communicate directly with each other without being managed by a main controlling node. In financial terms, this makes it possible to exchange information about economic transactions directly between the users of the blockchain without going through central control entities, such as banks. Each node has a local copy of the **ledger**, which is distributed to all nodes in the network. Since each block of the ledger contains a cryptographic hash of the previous block, any change in one block would cause a change in all subsequent blocks, this is why, in practice, it is extremely difficult to modify a block of the ledger. A further characteristic of the ledger is that it is only editable by an **append** operation, i.e. it is only possible to add blocks at the end of the ledger. In order to append new blocks to the ledger, these must be validated by the protocol that controls the blockchain, and only after consensus has been found among all the nodes in the network can it be added to the distributed ledger.

Figure 1.2: General structure of a blockchain



Each blockchain technology has its own implementation, although common elements can be recognised. The generic structure of a blockchain can be seen as a chain of blocks as in Figure 1.2.

The first block of the blockchain is generally called *genesis block*, and is the block from which the entire blockchain is originated. Each blockchain has its own genesis block, which is created when the blockchain itself is created. The genesis block plays a crucial role in the identification and start-up of a blockchain, providing the necessary information for the start of the chain and establishing the foundation for the functioning of the blockchain system. This block is the only block that has no reference to predecessor blocks and typically contains an initial configuration needed to start the blockchain, such as the creation date, the digital currency symbol, and rewards for miners.

In general, a blockchain block is composed of a **header** and a **body**, which contains the following information:

- **Hash of previous block**: a reference to the previous block, typically the hash of the header of the previous block. Using this reference, a chain of blocks can be generated, each linked to the previous block.
- **Nonce**: the "number used once" is a random number generated and used only once. This value is only used in certain types of consensus algorithms, called proof-of-work. It is used by miners, who try a huge number of nonce values until they find a value that produces the required hash that satisfies a certain property. In other types of consensus algorithms, such as proof-of-stake, there is no concept of mining or solving a cryptographic problem. Therefore, nonce is not necessary in proof-of-stake.
- **Timestamp**: contains the date and time the block was created.
- **Merkle root**: contains the hash of all the nodes that make up a Merkle tree, i.e. the hash of the block's transactions. It is used because it is very efficient for verifying transactions, as verifying the Merkle tree is equivalent to verifying all the transactions in the block one by one.
- **Transactions**: this is data describing an event. They are typically used to store financial transactions of the cryptocurrency associated with the blockchain.

1.3 Generating a new block

The term *blockchain* can be used in a broad sense to describe a decentralised and distributed system that uses a chain of blocks to record and verify transactions. However, the concept of blockchain includes more than just a chain of blocks.

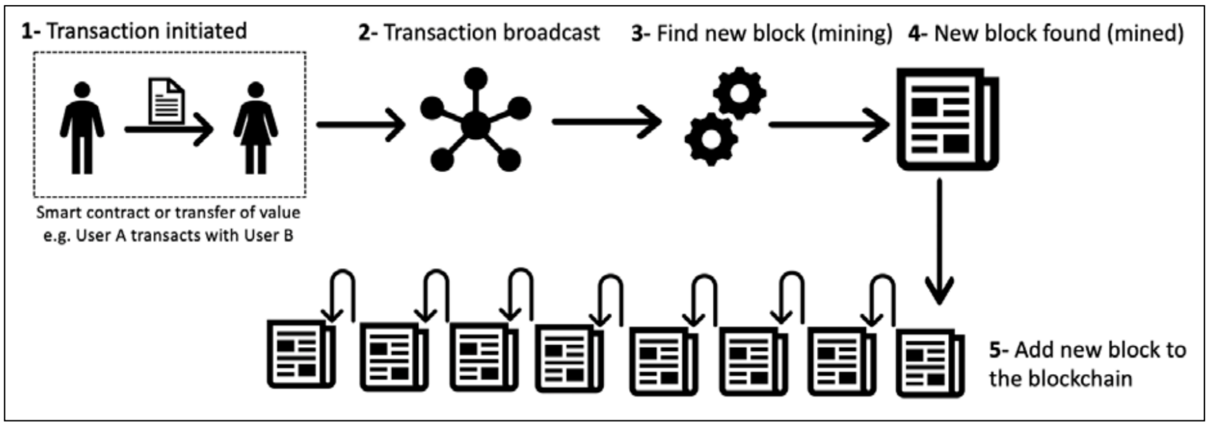
For the blockchain, a specific software implementation is required that allows nodes to communicate with each other, validate transactions, generate new blocks and maintain the integrity of the chain. In addition, an implementation of a consensus algorithm is required, through which the nodes in the network reach an agreement on the validity and state of the blockchain. This determines how the nodes that create a new block are selected and how conflicts are resolved if there are different opinions on the validity of the blockchain.

Nodes can typically be of two types, those that create new blocks (**miners or validators**) and those that validate created blocks (**block signers**). It is important to note that the terms used may vary depending on the specific PoS implementation. Each system may have its own unique definitions and terminologies for the participants that play the role of validating and creating blocks in the network.
The blockchain decides which node will be in charge of hanging the next block on the blockchain. This is the task of the consensus algorithm.

In general, a new blockchain creation process starts when a user creates a **transaction**. A transaction is a data structure that contains source and destination addresses

and amounts of cryptocurrency to be transferred. The transaction is then signed with the user's private key. Then the user that created the transaction sends it to a node that **distributes** it to the other nodes in the P2P network via broadcast protocols (e.g. Gossip protocol).
The transaction is validated by special nodes in the network that create new blocks, named *miners* in PoW and *validators* in PoS. Then they include it in the list of transactions and insert in the next block. At this point, the **block construction** process varies depending on the consensus algorithm used by the blockchain. In PoW blockchains, nodes compete to solve a cryptographic problem to create a new block and include transactions in it. In PoS blockchains, validators are selected proportionally to their stake to validate blocks. The first node to find the solution to the puzzle, in case of PoW, or the one selected as validator in PoS, receives a *block reward* and **appends the block** to the block chain. The transactions contained in the block are then confirmed.
The network nodes receiving the updated version of the blockchain can validate the newly created block, which at this point is in effect part of the ledger. The process of generating a new block is shown in Figure 1.3.

Figure 1.3: Block generation process


1- Transaction initiated
2- Transaction broadcast
3- Find new block (mining)
4- New block found (mined)
Smart contract or transfer of value
e.g. User A transacts with User B
5- Add new block to the blockchain

1.4 Consensus algorithms

A consensus algorithm is a mechanism that is used to reach an **agreement** on updating the system state. In blockchain, it serves to coordinate the nodes participating in the network so that they can agree on a shared state and build a consensus on the sequence of transactions. Specifically, it is a series of steps that all or most of the nodes in the network must follow to agree on the proposed state or value.

A consensus algorithm, to be defined as such, must fulfil certain **safety** and **liveness** requirements. The safety requirements include agreement, integrity, validity: they generally indicate that nothing bad should happen to the network. Liveness requirements include fault tolerance and termination and indicate that the algorithm should make progress even if the network and node conditions are not perfect.

- **Agreement**: All nodes acting honestly agree on the same status or value.
- **Integrity**: During the execution of a consensus cycle, a node may only make a decision once.

- **Validity**: The value agreed by all honest nodes must be the same as the initial value proposed by at least one honest node.
- **Fault tolerant**: The consensus algorithm should be able to function correctly even in the presence of Byzantine nodes.
- **Termination**: All honest nodes complete the consensus process and express their decision.

In client-server systems, the consensus algorithm is very simple, both nodes must agree on the state of the system, but the case where there are multiple nodes (**distributed consensus**), must handle the scenario where some nodes disagree or are byzantine nodes (faulty nodes or malicious nodes).

Consensus algorithms can be divided into two main categories, **traditional consensus (voting-based)** and **Nakamoto consensus (lottery-based)**.

Lottery-based consensus involves nodes competing to win a lottery. The winning node can propose a new value/state of the ledger with an extra block. These algorithms are based on the principle of proving a proof of some kind, e.g. proof of having done a job (proof-of-work) or proof of possessing a certain amount of tokens (proof-of-stake).

Traditional algorithms, such as Byzantine fault tolerance (BFT), where a small group of nodes, known as a committee or quorum, participates in the consensus process. This committee reaches agreement on the validity of transactions through a series of message exchanges and voting algorithms. Since it involves a limited number of nodes, the BFT mechanism can work quickly and efficiently, but has a limited scale. If the number of nodes in the network increases, it becomes more difficult to coordinate consensus and scalability suffers.

On the other hand, lottery-based consensus, such as the PoW, scales very well. In PoW, nodes compete to solve complex mathematical calculations in order to propose new transaction blocks and validate previous transactions. The node that solves the calculation first is selected as the leader and leads the consensus process. This mechanism can scale effectively, as the number of nodes in the network can be very large. However, the computational processing required to solve mathematical calculations is intensive and time-consuming. Therefore, PoW can be slow compared to BFT when it comes to reaching consensus and confirming transactions.

The choice of consensus depends on the specific needs of the blockchain and the desired level of decentralisation and security. Public blockchains, with broad participation and the goal of ensuring security and decentralisation, often opt for proof-based consensus, whereas permissioned blockchains are controlled by a restricted group of authorised participants and may adopt a more traditional consensus, such as Practical Byzantine Fault Tolerance (PBFT), to simplify implementation and meet the specific needs of the organisations involved.

The concept of proof-of-work (PoW) dates back to 1993 by authors Moni Naor and Cynthia Dwork [9] as a defence against denial-of-service attacks in order to require a proof-of-work from clients, thus proving that they have spent a certain amount of time in a computational intensive process.

The PoW mechanism was exploited and popularised in 2008 with the introduction of Bitcoin, in which special nodes called miners compete to create and hang a new block on the blockchain, where each miner has a probability of success proportional to the computational resources used to solve a mathematical puzzle.

Another popular consensus algorithm in blockchain is proof-of-stake (PoS), created by developer Sunny King in 2011 and introduced with Peercoin [30] in 2012, and based on proving possession of a certain amount of cryptocurrency.

Even today, the most popular distributed consensus algorithms in blockchains are **Proof-of-Work (PoW)** and **Proof-of-Stake (PoS)**. While both PoW and PoS aim to reach consensus on the state of the blockchain, they differ in their approach and have distinct advantages and disadvantages.

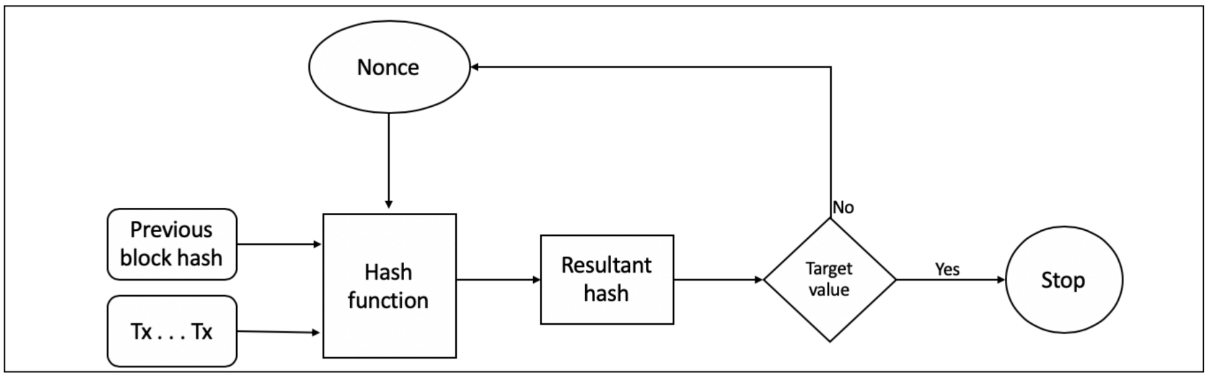
Both algorithms are effective in preventing **Sybil attacks**, in which a single malicious actor creates many false identities or nodes to influence the consensus of a distributed network. The main goal of these attacks is to gain disproportionate control over the network and compromise its security and integrity. When the network is under the control of a malicious node, then that node can influence the network's majority-based decisions, since the majority of decisions are controlled by him.

Both PoW and PoS attempt to mitigate Sybil attacks by introducing a 'cost' requirement to participate in the consensus process, although it is important to note that these consensus algorithms are not completely immune to this attack. Their goal is to provide a mechanism that makes it very costly and inconvenient to carry out these attacks.

Proof of Work

Consensus algorithms exploit a proof for which computational resources must be used to propose a new block, and thus a new blockchain state. This type of consensus is now used by popular blockchains such as Bitcoin and Litecoin and in many other cryptocurrencies.

Figure 1.4: Proof of Work consensus



The PoW process starts when new transactions are spread to all nodes in the network.

Each node collects these transactions to form a candidate block. The candidate block contains a list of transactions that need to be validated to be added in the blockchain. Miners, who are the participants trying to solve the PoW, propose new blocks. When a miner proposes a block, it must find a nonce such that H $<$ target, where H is defined as H $=$ (nonce || previous hash || Tx || Tx || ... || Tx). In this case the *target* is a value representing the difficulty in solving the puzzle.

If the resulting hash value is less than the target, the PoW is considered solved and the proposed block is accepted. Otherwise, the miner chooses at random another target and repeats the process. This cycle continues until a hash is found that meets the requirement of being below the nonce. Thanks to this process, visible in Figure 1.4, the PoW requires miners to dedicate significant computational resources to solve hash puzzles and propose valid blocks to the blockchain.

Proof of Stake

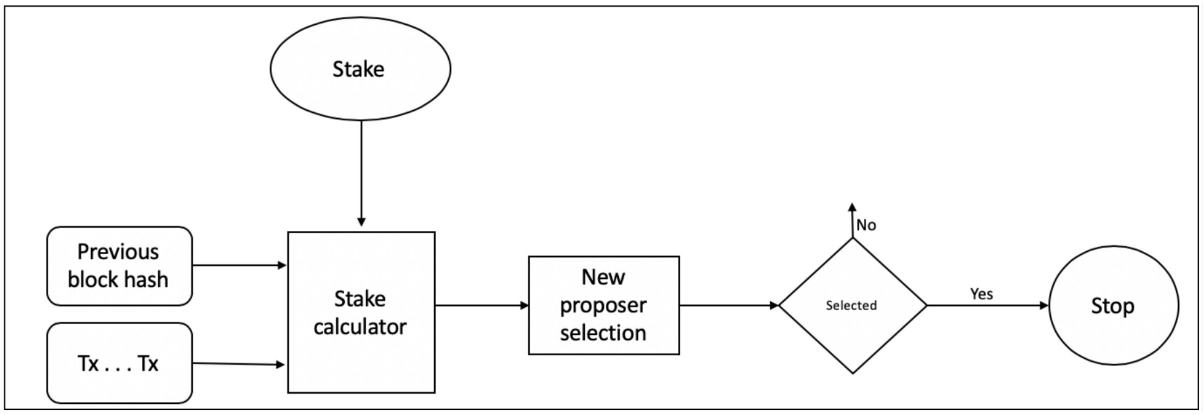
Algorithms of type **PoS** exploit a proof that a node must possess a certain amount of stake, typically a cryptocurrency associated with the blockchain. The idea is that the user must have invested enough in the blockchain so that any attempt at malicious behaviour would be punished with a reduction of the set stake. This type of consensus is used by popular blockchains including Ethereum (2.0) and Peercoin.

The key idea is that if someone has an interest in the system, they will not try to sabotage it. In general, the possibility of proposing the next block is directly proportional to the stake value of the node.

In PoS systems, there is no concept of mining in the traditional sense of Nakamoto consensus. A PoS miner is called a validator. The right to win the role of the next validator is usually assigned randomly, where the probability of being chosen as validator is proportional to the stake value. The proposers are rewarded with transaction fees or block rewards.

The PoS algorithm randomly selects a validator, who proposes the next block. The PoS process is shown in Figure 1.5. It is important to note that there are many variants and implementations of PoS, including Chain-based PoS, Committee-based PoS and Delegated PoS, but they all share the main mechanism just described. Furthermore, in some systems the amount of stake is used directly, while in others a function is used that considers other factors besides the stake, for example the time the stake has been stored by the node (**coinage**).

Figure 1.5: Proof of Stake consensus



1.5 Block rewards

A block reward is a reward given to miners (validators in PoS) who contribute to the creation of new blocks. The block reward usually consists of an amount of native cryptocurrency of the blockchain, which is awarded to the miner or validator who manages to solve the consensus puzzle and proposes the next block. This reward is an incentive for participants to spend their resources and work to maintaining the security and integrity of the network.

The main problem associated with block rewards in PoW blockchains, is the energy consumption and efficiency. Since PoW requires miners to solve complex cryptographic problems through computationally intensive processing, it requires a huge amount of computing power and energy. This leads to high energy costs and a large environmental footprint, as many miners use specialised hardware (ASIC) to gain a competitive advantage. In addition, increased competition between miners can lead to an arms race of ever more powerful hardware, which further fuels the overall energy consumption of the network.

In the case of PoS blockchains, the consensus mechanism does not require computationally intensive work as in PoW, in this sense , there is no real block reward as a reward for mining. However, in PoS, validators can be rewarded through transaction fees as block rewards. The amount of reward and its distribution may vary depending on the specific protocols used in the PoS blockchain.

Many blockchains, such as Bitcoin, are designed to gradually reduce the block reward over time. This mechanism, known as 'halving', reduces the amount of cryptocurrency issued as a reward for mining. The goal behind this gradual reduction is to control inflation and ensure that the supply of cryptocurrency increases more slowly than demand, contributing to its potential valuation over time.
In addition to block rewards, transaction fees are another form of incentive for miners or validators. Transaction fees are the fees paid by users to include their transactions within a block. In blockchain systems, miners or validators can select which transactions to include based on the fees offered. Therefore, transaction fees can become an essential part of the overall rewards for network participants.

The block reward must be designed in such a way as to provide an adequate economic incentive for participants to contribute to the security of the network. A reward that is too low may not be sufficient to cover operating costs or motivate participants, compromising the overall security of the blockchain. On the other hand, an excessively high reward could lead to a concentration of power in the hands of a few participants or encourage unwanted behaviour.

The reward system can influence the adoption and use of a blockchain. For example, a fair and well-balanced reward mechanism may attract more participants and promote economic activity on the blockchain. At the same time, an inappropriate rewards policy may discourage network membership or limit the interest of participants. If rewards are considered attractive and fair, users may be incentivised to become miners or validators, leading to greater decentralisation and security of the network. The ecosystem of participants contributes to the overall robustness of the blockchain. Proper block reward design can promote greater distribution of rewards within the network. This means that rewards should not be concentrated in only a few centralised nodes or entities, but should be accessible to a wider number of participants. The motivation is that a greater distribution can promote a more decentralized control and reduce the potential risk of centralisation and monopoly.

1.6 From Prof-of-Work to Proof-of-Stake

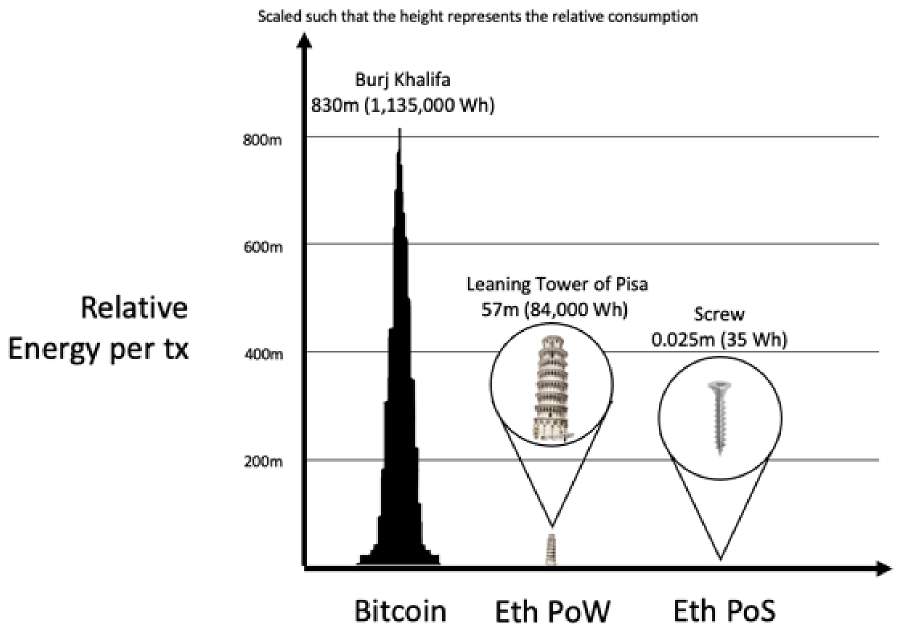
It is estimated that the Ethereum PoW miners consumed 44.5 TWh per year, or 5.13 gigawatts on an ongoing basis. PoS would be approximately 2000 times more energy efficient, resulting in a 99.95% reduction in energy consumption[10]. The Figure 1.6 shows a comparison in terms of electrical consumption bewtween the PoW (before the transition to PoS) and PoS.

The growing popularity of blockchains has led to more research into sustainability and scalability. For these reasons, many of the most popular blockchains have decided to adopt PoS as their consensus algorithm. Although each blockchain implements its own customized version of the consensus algorithms, they all have a common problem, i.e., how the **reward distribution** (or block selection) takes place and how many incentives the nodes of the blockchain receive.

In **PoW** based blockchains, the problem arises because only miners who have access to the most powerful mining devices have the opportunity to generate new blocks and receive the cryptocurrency reward. This leads to a concentration of mining power in the hands of a few individuals or organizations, who may have excessive control over the system.

In **PoS** based blockchains, the problem arises because only nodes with a large number of coins "staked" (i.e. locked as collateral for block generation) have a higher probability of being selected to generate new blocks and receive the reward. This can lead to a concentration of wealth in the hands of a few individuals or organizations, who may have, like in the PoW case, excessive control over the system.

Figure 1.6: Relative energy consumption per transaction



A node with excessive control over the blockchain can compromise the security, decentralization, transparency and reliability of the network. Therefore, it is important for blockchains to maintain a balanced distribution of the control to ensure a fair network.

One of the most famous cases that highlighted the problem of centralization was the case of **Ethereum** [33] when it switched from the PoW to the PoS algorithm on 15 September 2022. The news outlets published headlines such as *"Two Addresses Control Over 45% of Ethereum Validator Nodes Post Merge"* [31] and *"40%+ of Ethereum PoS nodes are controlled by 2 addresses, says Santiment data"* [1].

This strong dominance by a few addresses has led many users to criticize PoS, saying that this consensus model leads to centralization as validators are weighted based on the amount of ETH they have staked. Centralisation of control can lead to less security, greater risk of manipulation and potential abuse of power, while an important objective of a blockchain is to promote decentralisation to ensure greater democratisation and participation in the control of blockchain networks."

To solve this problem, some variations of PoS introduce features for the selection of block-generating nodes, to ensure greater decentralization and democracy in the system. For example, random or weighted selection mechanisms can be implemented to ensure a fairer distribution. Additionally, some PoS implementations include "slashing" mechanisms (punishment) for nodes that violate the system's rules, to prevent the misuse of staked coins.

The simulation-based study presented in this thesis report was born to deepen the critique of reward distribution in blockchain PoS networks, and examine the effects of variants of PoS algorithms in the redistribution of wealth and how these can contribute to the centralization effect.

Chapter 2

State of the art

In this chapter, the state of the art concerning blockchain technology will be presented, focusing first on **technical aspects** of the technology and how a network of nodes can achieve distributed consensus. In this sense, references will be made to the literature regarding two consensus mechanisms popular today, proof-of-work (PoW) and proof-of-stake (PoS), as well as on the energetic motivations that caused some significant blockchains to switch from PoW to PoS.

Secondly, will be shown references to the criticism made by the blockchain community towards some possible emerging issues in PoS, including the effect of **rich nodes becoming richer** and richer. Studies conducted in this context will be presented, analysing the problem of the inequality and wealth within a group of validators/miners of a blockchain.

The problem of inequality is theoretically linked to a centralisation effect of the blockchain, which could have **security implications** even in the PoS consensus.

Statistical indices will be considered, including the Gini coefficient and Lorenz curves, to measure **economic inequality** in a blockchain network and in particular to analyse wealth and income distributions for a set of validators of a consensus PoS.

Finally, some **variants** of PoS will be presented, which offer different incentives to the validator nodes of the network, focusing in particular on the Coin-Age PoS type.

2.1 Blockchain

The blockchain is a technology whose invention dates back to 2008, by the unknown author of the pseudonym Satoshi Nakamoto, who published a scientific paper entitled *Bitcoin: A Peer-to-Peer Electronic Cash System* [24]. Bitcoin was implemented the following year, the purpose of which is to store all cryptocurrency transactions of Bitcoin.

A blockchain [4] is a cryptographically secure, peer-to-peer distributed system in which blocks can be attached to the chain, which is immutable in practice (that is, extremely difficult to mutate) and uses a consensus algorithm to reach agreement on the state of the blockchain.

Typically, blockchains do not guarantee consistency, and thus fall into the AP category of the CAP theorem. Each node belonging to the blockchain network has a local copy of the distributed ledger, consisting of a blockchain, where each block contains information about the state of the system, for example financial transactions in the case of cryptocurrencies.

2.2 Consensus algorithms

A consensus algorithm is a mechanism that is used to reach an agreement on updating the system state. In blockchain, it serves to coordinate the nodes participating in the network so that they can agree on a shared state and build a consensus on the sequence of transactions. Specifically, it is a series of steps that all or most of the nodes in the network must follow to agree on the proposed state or value.

These algorithms acts on the nodes of a blockchain and are currently divided into two main types [32]: proof-of-work (PoW) and proof-of-stake (PoS).
PoW algorithms use a consensus mechanism that relies on competition between network participants to solve complex mathematical problems in order to validate transactions and create new blocks. In a different way, PoS algorithms use a consensus mechanism in which the selection of the validator of a new block is based on the amount of cryptocurrency owned and blocked by the participant.

Furthermore, Proof of stake (Pos) consensus algorithms have been proposed as a solution to reduce the huge energy demand of PoW consensus algorithms [10] to solve a computationally difficult problem.

In order to obtain an overview of real-world approaches to consensus mechanisms, a number of blockchains were considered as case studies. Bitcoin [24] was taken as a reference to examine the functioning of PoW, while Ethereum [33] was considered for the analysis of PoS functioning.

Other PoS consensus blockchains have introduced the concept of **coin age** as mining resource, for instance, Peercoin [30], Cloakcoin [7], and Novacoin [26] use coin age to append new blocks to the blockchain distribuited ledged. The coin age variant chooses nodes based on how long ago the tokens were staked by a node. Once a node has forged a block, the coin age is restored and has to wait a certain period of time to be able to forge another block. This prevents large staking nodes from dominating the blockchain.

2.3 Inequality problem

A block reward is a reward given to miners (validators in PoS) who contribute to the creation of new blocks. The block reward usually consists of an amount of native cryptocurrency given to the validator who has been selected with a pseudo-random sampling from the validator set.
The introduction of the PoS consensus algorithm has raised the concern for the problem of **inequality** within a PoS based blockchain.

This problem was discussed in the blockchain community [14] in which users were concerned about the 'rich getting richer' effect, having more tokens to stake and consequently being more likely to earn block rewards and giving more power to create new blocks to a few validators, the richer ones.

Studies on this topic have been carried out to analyze the problem of wealth compounding and the equitable redistribution of block rewards. The main criticality that differentiates PoW from PoS in the context of the block reward is the compounding effect [11]. It consists of the instantly re-investment of the block reward as stake, which produces a drastic "rich get richer effect". The compounding of wealth [11] in PoS introduces the notion of equitability, to quantify how much a proposer can amplify his stake compared to the initial investment. The authors of [11] prove that the geometric reward function is maximally equitable, by carrying out a study based on a block rewards simulator for a PoS blockchain.
The mathematical description of the simulator presented in this work (Fanti et al., 2019) has been an inspiration for the development of this thesis.

The article *Characterising Wealth Inequality in Cryptocurrencies* [29] explores the economics of cryptocurrencies and focuses on the distribution of wealth within eight major cryptocurrencies: Bitcoin, Ethereum, Bitcoin Cash, Dash, Litecoin, ZCash, Dogecoin and Ethereum Classic. By means of a generic econometric analysis scheme, two main measures are presented: the Gini value and the Nakamoto index, which respectively report the inequality in the distribution of wealth and the concentration of wealth. A parallel is also drawn between the economies of cryptocurrencies and those of the real world. The analysis reveals that despite the emphasis on decentralisation of cryptocurrencies, the distribution of wealth is similar to that of real-world economies, with the exception of Dash. Furthermore, it emerges that three of the observed cryptocurrencies (Dogecoin, ZCash and Ethereum Classic) violate the assumption of the **honest majority**, with fewer than 100 participants controlling over 51% of the wealth in the ecosystem, potentially indicating a security threat. This article was useful in order to understand the methodologies of economic analysis in the blockchain world.

2.4 Security implications of inequality

51% attacks can occur in cases where a single entity or group of participants controls more than 51% of the computing power (in the case of PoW) or stake (in the case of PoS) of a blockchain. In PoS networks, a 51% attack is considered very difficult to carry out and has limited consequences compared to PoW, firstly because owning 51% of the total stake would typically require a large economic investment, secondly most PoS networks have security mechanisms designed to prevent 51% attacks, for example, by blocking stakes for an initial period of time, or by applying a penalty for those who engage in behaviour detrimental to the security of the network. This penalty is called **slashing**, and consists of confiscating all or part of the stake held by the malicious node.

Furthermore, even in the event that this attack is successful, the value of the cryptocurrency would fall due to the loss of user trust, so that the attacker who owns the majority of the stakes would incur the greatest economic losses.

Although implementing the 51% attack in PoS networks is considered very difficult, a way has been found in the literature in which the attacker could make money from this operation by applying the **Short Selling Attack** [19]. This is a strategy used in the traditional stock market, in which, the attacker who owns at least 51% of the stake sells a large amount of cryptocurrency short, and then sabotages the PoS system. Subsequently, the attacker repurchases the cryptocurrency at a reduced price to cover his short selling position and make a profit.

2.5 Measuring Inequality

Economic inequality is a concept that refers to the inequality in the distribution of economic resources between individuals, social groups or nations. In order to measure economic inequality [16], two aspects are generally considered: the **income distribution** refers to the division of income among individuals or groups in a population, while the **wealth distribution** refers to the division of wealth among individuals or groups in a population.
The first one focuses on the division of income as a flow of payments received, while the second one concerns the overall division of financial and wealth resources.

These concepts find a correspondent in the world of cryptocurrencies. They can be used to measure inequality within a blockchain, where the income distribution is associated with the distribution of **block rewards received** by consensus algorithm, while the wealth distribution is associated with the distribution of tokens owned in **stake** by the network nodes.

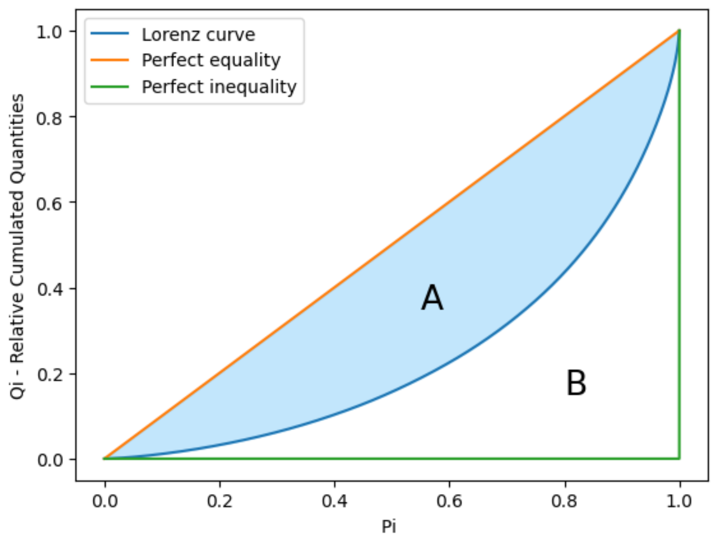
The measurement of wealth inequality can be achieved through **econometrics**, a field of study concerned with applying statistical techniques to economic data in order to produce empirical evidence on financial issues.

In 1905, Max O. Lorenz developed a graphic way to represent economic inequality through the use of the **Lorenz curve** [12]. The Lorenz curve can be used to visualise either the income distribution or the wealth distribution.

The curve graphically shows the proportion of income (or wealth) that the lower x% of the population has. On the y-axis, the percentage of the population is represented, while on the x-axis is the percentage of income (or wealth) they possess. Points on the Lorenz curve (as in Figure 2.1) can be interpreted with the following statement: *"the P_i% of all nodes have Q_i% of the total wealth (income)"*.

The shape and slope of the Lorenz curve highlight the degree of inequality in the distribution of wealth. The **perfect equality** line (45° line) represents a perfect distribution of wealth, where each percentage of the population holds a corresponding percentage of the total wealth.
The **perfect inequality** line, on the other hand, represents a distribution of wealth in which a small number of individuals owns all the total wealth.

Figure 2.1: Lorenz curve



A curve that tends to approach the line of perfect equality represents a more equal population, while a curve that tends to squash near the line of perfect inequality indicates a less equal distribution of wealth.

The Gini coefficient was defined by the statistician Corrado Gini in 1921 in the paper *Variabilità e mutabilità* [13] to measure the inequality of a distribution. It is a real number between 0 and 1. It is calculated from the Lorenz curve, and can be defined as the ratio of the area between the line of perfect equality and the Lorenz curve (A) to the area between the line of perfect equality and the line of perfect inequality ($A + B$). Since the values of x and y are always between 0 and 1, then the area $A + B$ is always equal to 0.5 as in equation 2.1.

$$G = A/(A + B) = 2A = 1 - 2B \tag{2.1}$$

Low values of the coefficient indicate a more homogeneous distribution, with the value 0 corresponding to pure **equal distribution**, the situation in which everyone receives exactly the same income in the case of the income distribution measure, or the situation in which everyone possesses the same wealth in the case of the wealth distribution measure.

High values of the coefficient indicate a more **unequal distribution**, with a value of 1 corresponding to the highest concentration, i.e. the situation where an individual receives all the income, in the case of an income distribution measure, or the situation where an individual is in possession of all the wealth in the case of a wealth distribution measure.

In the course of the thesis we will use the Gini coefficient calculation [22] for discrete random variables, in which each observation will indicate a stake value relative to either the wealth distribution or the income distribution. Given n observations, we denote by $a_1, a_2, ..., a_n$ the non-decreasingly ordered observations. The mean and total sum of these observations are then calculated as shown in the equations 2.2 and 2.3, respectively.

$$\bar{a} = \frac{1}{n}\sum_{i=1}^{n} a_i \tag{2.2}$$

$$T = \sum_{i=1}^{n} a_i = n\bar{a} \tag{2.3}$$

There are two extreme cases:

- **Minimum concentration** if all elements assume the same value: $a_1 = a_2 = ... = a_n = \bar{a}$.
- **Maximum Concentration** if all elements take the value 0 except one: $a_1 = a_2 = ... = a_{n-1} = 0$ and $a_n = T$.

The cumulative relative frequency up to the i-th observation is then considered for $i = 1, 2, ..., n$, as in equation 2.4, and the cumulative relative quantity up to the i-th observation as in equation 2.5.

$$F_i = \frac{i}{n} \tag{2.4}$$

$$Q_i = \frac{1}{T}\sum_{k=1}^{i} a_k \tag{2.5}$$

The pairs (F_i, Q_i) indicate that $100F_i\%$ of the population holds $100Q_i\%$ of the total wealth. In the minimal concentration case all points (F_i, Q_i) lie on the perfect equality line $F = Q$ with $F_i = Q_i \; \forall i = 1, 2, ..., n$.
In the case of maximum concentration the points (F_i, Q_i) lie on the perfect inequality line $Q = 0$ except for the last one for which $F_n = Q_n = 1$ with $F_i - Q_i = F_i$, $\forall i = 1, 2, ..., n-1$ and $F_n - Q_n = 0$.

The ratio G shown in the equation 2.6 is defined as **Gini concentration index** (or Gini coefficient).

$$G = \frac{\sum_{i=1}^{n-1}(F_i - Q_i)}{\sum_{i=1}^{n-1} F_i} = \frac{2}{n-1}\sum_{i=1}^{n-1}(F_i - Q_i) \tag{2.6}$$

It is also noted that the denominator of the equation 2.6 represents the maximum value that the numerator can take, and is therefore used as a normalisation factor to transform values in the range $0 \leq G \leq 1$, as shown in the equation 2.7.

$$\sum_{i=1}^{n-1} F_i = \frac{1}{n}\sum_{i=1}^{n-1} i = \frac{1}{n}\frac{n(n-1)}{2} = \frac{n-1}{2} \tag{2.7}$$

Chapter 3

The Project

This chapter outlines the underlying **assumptions** made for modeling, which introduce certain limitations that allow for a purely theoretical description, but may not fully reflect real blockchain implementations and scenarios. Subsequently, the basic **mathematical theory** will be explored, that takes inspiration from *Compounding of Wealth in Proof-of-Stake Cryptocurrencies* [11] with some modifications.

The mathematical model will serve as a foundation for implementing the simulator. This includes the modeling of validators and stakes, the initialization of the initial state based on the chosen initialization function, the mathematical modeling of block reward functions to distribute stakes to validators over time, and the validator selection process, which involves various functions assigning probabilities to each validator for block validation.

Additionally, the chapter explores the utilization of **metrics**, specifically based on the Gini coefficient, to evaluate economic equality within two aspects: income distribution, represented by how stakes are distributed among validators, and wealth distribution, represented by the amount of stake owned by validators.

3.1 Assumptions

The simulator described in the next chapter makes some necessary assumptions in order to describe the distribution of stakes in a simulated context.

The set of nodes/agents that act as validators, i.e. the nodes that propose themselves to validate the transactions, remains **fixed** during the execution of a simulation. Unlike real implementations where validator nodes can be added and removed, it was decided to model a fixed amount of nodes so as not to add complexity to the simulation comparison.

Typically, the blockchain elects the validator through a proposal mechanism, for example by selecting a node weighted on the amount of stakes. The node that is elected as validator generates the next block in the ledger and earns transaction fees and/or the block reward. These gains are incentives to ensure network security, with the difference that the transaction fees are set by users who want to have priority in consolidating a transaction in the blockchain while the block reward is set by the system designer. In order to analyze the distribution of node stakes over time, the **transaction fees are**

trascured for each block validated and just the block rewards will be considered.

In popular blockchains [24, 15, 34] it may happen that the ledger splits into two branches; this process is called **a fork**. A fork can occur for several reasons, including network latency, disagreements within the blockchain community, multiple proposers elected in the same time slot, or because a block proposer has equivocated, that is, sending blocks that do not comply with the consensus protocol. In these cases the nodes require an algorithm to choose for a branch of the fork. Generally, the blockchain can adopt a rule to choose the branch with the most blocks, with the greatest computational weight, or the one that has received the most support from validators over time. The blockchain then merges into a single branch. Since only one fork branch will eventually be considered, only the proposers belonging to the chosen fork will receive block rewards. In order to model the distribution of block rewards, it is assumed that there are **no fork processes** and that only one proposer is elected at each time slot.

It is also assumed that all rewards received by validators are **instantly staked**, so validators can increase their chances of being elected. This process can be considered a form of instant reinvestment and is enabled in popular blockchain PoS, for example in Casper-the Friendly Finality Gadget protocol [15]. The simulator describes a system made up exclusively of nodes participating in the validation process, also called "validators". The remaining nodes of the network are ignored in order to focus the analysis on the wealth distribution of the validators.

3.2 Mathematical model

In a PoS based blockchain, nodes that want to take the role of validators must put a certain number of tokens at stake. This is known as **staking**.
The set of validating nodes, also called agents in this model, is then defined as $A = \{A_1, A_2, ..., A_m\}$. These agents are responsible for validating transactions and securing the network. They are incentivized to act honestly by the rewards they receive for participating in the validation process and the penalties they incur for any malicious behavior.

The model evolves in discrete time $t \in \{1, 2, .., T\} \subset \mathbb{N}$ where each t denotes a epoch in the model life-time. Each epoch starts with a block proposal and terminates with the update of the model state, that is, the addition of the reward to the selected validator stake or a punishment for a malicious intent.

At each time slot, the state of the model is updated accordingly to the proof of stake algorithm selected at the beginning. The state of the model is described through the agents' stakes. For each agent A_i, where $i \in \{1, 2, .., m\} \subset \mathbb{N}$, we denote by $S_{A_i}(t) \in \mathbb{R}_+$ the stake of the agent A_i at time t.

We denote by $S(t) = \sum_{i=1}^{m} S_{A_i}(t)$ the sum of the stake volume at fixed time t, and by $v_{A_i}(t) = \dfrac{S_{A_i}(t)}{S(t)}$ the fractional stake of agent A_i at time t.

At each time t the proof-of-stake algorithm of the blockchain elects a validator node

$W(t) \in \{A_1, A_2, ..., A_m\}$, and a new block composed of a set of transaction is added to the blockchain. The validator $W(t)$ is rewarded with the block reward $r(t)$, that is an amount of tokens added to the $W(t)$ agent.

The stake of the validator $W(t)$ is then updated consequentially as:

$$S_{W(t)}(t+1) = S_{W(t)}(t) + r(t) \tag{3.1}$$

Initial state

In order to generate the initial state at time $t = 0$, each node must set an amount of stake. The simulator provides the following functions for generating the initial distribution: constant, linear, polynomial, custom, gini. The function that generates the initial function is therefore an input parameter to the simulator.

The generation of the initial state using the **Constant** function assigns each node the same amount of stake. Let $S(0)$ be the initial volume, and m the number of nodes, then $S_{A_i}(0) = S(0)/m \ \forall i \in \{1, 2, ..., m\}$.

The **Linear** function, given the initial volume $S(0)$, predicts that the stakes assigned to the nodes grow linearly as a function of the i index of the node:

$$S_{A_i}(0) = 2i * S(0)/(m + m^2), \forall i \in \{1, 2, ..., m\} \tag{3.2}$$

To obtain the linear function in equation 3.2 in the form of $y = a * i$ such that the sum of all values is equal to the volume $S(0)$, as $\sum_{i=1}^{m} a * i = S(0)$ is necessary to obtain the slope a as in equation 3.3:

$$\sum_{i=1}^{m} a * i = S(0) \rightarrow \sum_{i=1}^{m} i = \frac{S(0)}{a} \rightarrow a = \frac{S(0)}{\sum_{i=1}^{m} i} \xrightarrow{gauss} a = \frac{2S(0)}{m + m^2} \tag{3.3}$$

The **Polynomial** is a generation function in which, given the initial volume $S(0)$ and the degree $d \in \mathbb{N}$ of the polynomial, the assigned stakes grow polynomially according to the index i of the node.

$$S_{A_i}(0) = i^d * S(0)/\sum_{j=1}^{m} j^d, \forall i \in \{1, 2, ..., m\} \tag{3.4}$$

To obtain the polynomial function in equation 3.4 in the form of $y = a * i^d$ with the constaint $\sum_{i=1}^{m} a * i^d = S(0)$ is necessary to obtain the parameter a as in equation 3.5.

$$\sum_{i=1}^{m} a * i^d = S(0) \rightarrow a \sum_{i=1}^{m} i^d = S(0) \rightarrow a = S(0)/\sum_{i=1}^{m} i^d \tag{3.5}$$

Given a volume $S(0)$ and a Gini coefficient g, the generation function **Gini** outputs an initial distribution whose Gini coefficient is g. Let us note that, given a Gini coefficient

g and a initial state volume $S(0)$, there are multiple generative functions that have the same Gini coefficient g. A naive method can be generating randomically the agent stakes until an initial state with a Gini coefficient g is obtained. I explored instead another approach that is more efficient than generating randomically the stakes.

In order to obtain the Gini generation function for the initial state, given the initial volume $S(0)$ and the initial Gini coefficient $g \in [0, 1] \subset \mathbb{R}_+$, a linear Lorenz curve is generated. The F_{m-1} value of the Lorenz curve is associated with the F_{m-1} value. For every Lorenz curve we have that $\sum_{i=1}^{m-1}(F_i - Q_i) = R$, where R is the Gini coefficient before the normalization $G = R * \frac{2}{n-1}$.
In order to obtain the Lorenz curve that satisfies the Gini coefficient g given as input, we can calculate the Q_{m-1} value of the lorenz curve as:

$$Q_{n-1} = g * P_{n-1} = g * \frac{2}{n-1} \tag{3.6}$$

We can then calculate the slope of the lorenz curve $l(i) = m * i$ passing through the points $(0, 0)$ and (P_{n-1}, Q_{n-1}) as:

$$m = \frac{Q_{n-1}}{P_{n-1}} \tag{3.7}$$

The Lorenz curve l, that has the same Gini coefficient as the given g, can be used to extract the initial stakes for all nodes as:

$$S_{A_i}(0) = \begin{cases} S(0) * Q_1 & \text{if } i = 1 \\ \sum_{j=1}^{i}(S(0) * Q_j) - \sum_{j=1}^{i-1}(S(0) * Q_j) & \text{if } i \in \{2, .., m\} \end{cases} \tag{3.8}$$

Block Reward

The model can distribute a total of R tokens in T epochs. At each epoch $t \in [1, T]$ a validator $W(t) \in A$ is selected by the selection function and the reward $r(t)$ is added to the $W(t)$ stake as the equation 3.1.

The reward function r is defined as $r : \mathbb{N} \to \mathbb{R}$, takes a time epoch as input and returns the block reward. Since a total of R tokens are dispensed in T epochs, it follows that the sum of individual rewards at each epoch t must be equal to the total R tokens

$$R = \sum_{t=1}^{T} r(t) \tag{3.9}$$

The block reward of the **Constant** function r_c is fixed for each epoch as:

$$r_c(t) = R/T, \ \forall t \in [1, T] \tag{3.10}$$

The **Geometric** function makes the block reward increase geometrically in function of time as:

$$r(t) = (1 + R)^{t/T} - (1 + R)^{(t-1)/T}, \ \forall t \in [1, T] \tag{3.11}$$

PoS selection

The function $selection_f$ selects the validator $W(t) \in A$ for each epoch $t \in [1, T]$.

The simulator receives as input one of the following selection functions $selection_f$: random, weighted, coin age, dynamic.

In the **Random** case the validator $W(t)$ is sampled randomly in A. All nodes have the same probability of being sampled: $P(t, A_i) = 1/m \ \forall i \in \{1, 2, .., m\}$. The stakes have no role in determining the next validator, as they are not considered during the decision.

In the **Weighted** selection, the probabilty distribution for the sampling is the fractional stakes of the agents: $P(t, A_i) = v_{A_i}(t) \ \forall i \in \{1, 2, .., m\}$. In this sense, having a higher stake increases the probability of being selected.

The coin age C for an agent A_i at time t is defined as the number of epochs for which the tokens have been set in stake, as

$$C(t, A_i) = \begin{cases} 1 + C(t-1, A_i) & \text{if } A_i \neq W(t) \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

In the **Coin age** selection, the probability distribution for the sampling is given by the product of the coin age and the agent stake: $P(t, A_i) = S_{A_i}(t)C(t, A_i)/S(i)$.

In the **Dynamic** selection, a Gini threshold $d \in [0, 1]$ is given. When the Gini coefficient $g = G(S(t))$ of the model state, at time t, satisfies the condition $g \geq d$, then the weights used for sampling the validator are reduced with a linear transformation.

$$l(x) = (1 - 2\theta)x + \theta \tag{3.13}$$

$$\theta = \begin{cases} \frac{1}{2}g - \frac{1}{5} & \text{if } g \geq d \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

$P(t, A_i) = f(v_{A_i}(t)) \ \forall i \in \{1, 2, .., m\}$

Metrics

A metric g is used to calculate the level of **inequality** of the system. It's defined as a measure which associates a state of the system with an inequality value between 0 and 1 as $g : \mathbb{R}^m \to [0, 1] \subset \mathbb{R}$ where m is the number of nodes belonging to the stake pool.

To measure how a transferable asset is divided into a population, **concentration indices** can be used, which mathematically implements the concepts of homogeneity and diversity of wealth. The Gini coefficient [20] is a statistical concentration index which can therefore be used as a metric function.

Given a system state $\vec{s} \in \mathbb{R}^m$ which represents the stakes of m nodes, the **Gini coefficient** (or Gini concentration index) of $\vec{s}$ can be defined in terms of P_i and Q_i as:

$$g(\vec{s}) = \frac{2}{m-1} \sum_{i=1}^{m} -1(P_i - Q_i) \tag{3.15}$$

It should be noted that for convenience the normalized Gini coefficient (that assumes values in range $[0,1] \subset \mathbb{R}$) is calculated, i.e. divided by its maximum value $(m-1)/2$.

If the metric is equal to 0 ($g(\vec{s}) = 0$) the system is in **equidistribution**, i.e. all nodes have the same amount of stake $S_{A_i}(t) = S_{A_j}(t)\ \forall (i,j) \in \{1,2,..,m\}^2$. If the metric is equal to 1 ($g(\vec{s}) = 1$) then the system is in a state of **maximum concentration**, i.e. there is a node A_i that owns the entire quantity staked $S_{A_i}(t) = S(t)$. In case the system is in intermediate states ($g(\vec{s}) \in (0,1)$) then the system is in a state of **concentration**.

The Gini coefficient is useful for calculating the inequality of the system in a given state. Taking inspiration from the definitions of equality in *Inequality In Proof-of-Stake Schemes* [3], the Gini coefficient can be used as indicator to quantify the effect of the inequality over time, where the optimum equitable model is the one that mantains the same Gini coefficient for the whole simulation. Since we want to measure the economic inequality for both the wealth distribution and the income distribution, we have extended the definitions also to include the income inequality.

Given two states of the system $\vec{s_0}$, $\vec{s_T} \in \mathbb{R}^m$ the metric of **Gini stake** g_s is defined as $g_s : \mathbb{R}^m \times \mathbb{R}^m \to [0,1] \subset \mathbb{R}$. In particular, g_s is a function that receives as input the final state $\vec{s_T}$ and the initial state $\vec{s_0}$ of the system and returns the level of inequality of the simulation as:

$$g_s(\vec{s_0}, \vec{s_T}) = g(\vec{s_T}) - g(\vec{s_0}) \tag{3.16}$$

Given two states of the system $\vec{s_0}$, $\vec{s_T} \in \mathbb{R}^m$ the metric of **Gini reward** g_r is defined as $g_r : \mathbb{R}^m \times \mathbb{R}^m \to [0,1] \subset \mathbb{R}$. In particular, g_r is a function that receives as input the final state $\vec{s_T}$ and the initial state $\vec{s_0}$ of the system and returns the level of inequality of the simulation:

$$g_r(\vec{s_0}, \vec{s_T}) = g(\vec{s_T} - \vec{s_0}) - g(\vec{s_0}) \tag{3.17}$$

The newly introduced metrics g_s and g_r measure the **gain or loss of equality** of the system during the simulation from time $t = 0$ to time $t = T$, i.e. by how much the homogeneity of wealth has changed from the state $\vec{s_0}$ to the final state $\vec{s_T}$. In this sense, a positive value indicates that the system has introduced inequality, a positive value indicates that it has become more homogeneous, while a 0 value indicates that the system has maintained the same initial homogeneity.

A system with positive metrics is a factor that results in the effect of the rich getting richer, while a system with a negative metric results in an effect of the poor getting richer, but this would not be realistic, otherwise we would have no incentive to stake many tokens.

The **optimal** system is therefore the one which maintains the initial inequality, the one with metrics equal to zero. In a system with positive metrics, rich individuals would tend to become even richer, while a system with negative metrics would lead to poor individuals becoming richer. However, it would be unrealistic to have a system in which the wealth of individuals only changes based on being block validators, as this would discourage people from blocking tokens in stakes. The optimal system should strike a balance, maintaining the initial inequality while ensuring that being a validator does not make someone significantly richer or poorer. It should provide incentives to validators without changing wealth inequality.

Chapter 4

The Simulator

In this chapter, the implementation of a simulator in Python based on the theory described in Chapter 3 is explored.

During the creation of the simulator, inspiration was drawn from the theory of **agent-based modeling**, a form of simulation that involves multiple entities, called agents, interacting with each other following a programmed behavior. These agents can represent a wide range of elements, such as living cells, animals, human individuals, organizations, or abstract entities. Through the programming of agents, who possess specific attributes, and their interactions with other agents and the environment, effects emerge that go beyond individual behavior. This modeling approach allows us to explore complex systems and understand how individual components of a system influence the overall behavior and the effects that emerge from their interactions.

Initially, the agent based model libraries such as Mesa [23] and NetLogo [25] were considered because the simulator is based on agent modeling. However, since in this case the agents do not interact directly with each other, but are controlled by a main loop that manages the state of the agents and their resources, it was decided to develop a **custom** simulator to ensure **optimized performance**, without adapting any existing library. In practice, in the main loop of the simulator, the complexity is linear with respect to the number of simulations, $O(n)$, whereas in a generic library for agent-based modeling, the interactions between agents are also modeled. In this case, each agent has its own *step()* method for interacting with other agents, leading to a complexity of $O(n * m)$, where n is the number of simulations and m is the number of agents involved.

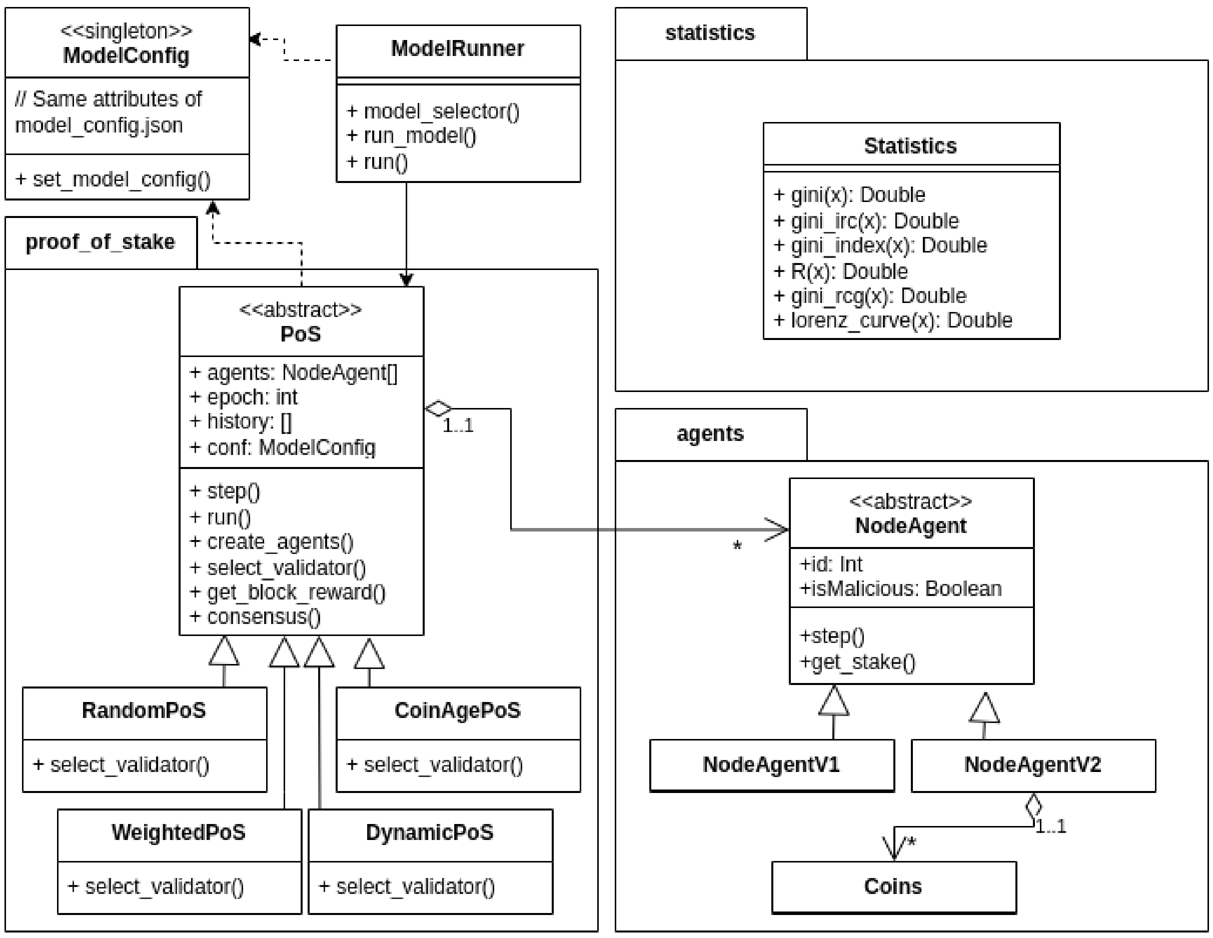
The operation of the simulator can be described as follows at a more general level: it receives as **input a configuration file** (model_config.json), located in the project's root directory. This file contains the parameters needed to start a custom simulation, including the number of agents, the number of epochs, and the PoS variant to be used. Next, the simulator sets the initial state using the algorithm selected from the model_config.json file and starts the simulation. During execution, there are elements of **randomness** in the selection of validators, which leads to **different results each run**, even if starting from the same initial configuration. In order to improve the robustness of the results from a statistical prospective, **Monte Carlo techniques** can be applied by running multiple simulations starting from the same initial state and then merging the results obtained by calculating the mean and variance of the different runs.

The simulator was developed in Python, taking advantage of some of the most popular libraries for data analysis. In particular, **NumPy** was used to efficiently compute the Gini coefficient, **Pandas** was used to obtain the results of the simulations in dataframe form, and **Matplotlib** was used to graphically analyze the results obtained, allowing the results obtained from the simulations to be clearly visualized.

4.1 Design

The simulator provides a simulation configuration class named **ModelConfig**, modeled as a singleton, globally accessible from any other class. It is initialized during the first construction of the object by reading the parameters written in the *model_ config.json* file. It was constructed in such a way that it can be initialized again at run-time; this feature was useful when analyzing the results of running several simulations in a loop.

Figure 4.1: Simulator class diagram



The **ModelRunner** class is responsible for launching simulations via the *run()* method, taking advantage of the parallel programming technique and the *multiprocessing* library to launch several executions of a simulation and to merge the results into a Pandas dataframe at the end of all parallel executions. Each run is launched with the *run_ model()* method that selects the model (*model_selector()*) by accessing the ModelConfig singleton object.

A node in a blockchain is represented by the abstract class NodeAgent, which stores the node identifier and the stake associated with the node. The simulator is designed in such a way as to make it easy to add nodes with different characteristics and capabilities to the simulator: this is the case with **NodeAgentV1** and **NodeAgentV2**, which contain different data structures for managing the coins/stakes of a node. NodeAgentV1 simply stores the stake and coin age for a node, while NodeAgentV2 uses a list of pairs (coins, coinAge) which allow tracking the coinAge of each coin transaction entered into the stake.

The PoS abstract class is inherited from the PoS variants (RandomPoS, Weighted-PoS, CoinAgePoS and DynamicPoS), and contains the state of an execution, including the current epoch, validator agents, and epoch history. At construction time it initializes the agents and the initial stake distribution. The main method is *run()* which contains the main-loop to cycle through the epochs of the execution. Each epoch is executed through the method *step()* which executes the consensus algorithm *consensus()*.

The class **Statistics** is used in the simulator to obtain concentration indices, in particular the method for calculating the Gini coefficient is used in DynamicPoS. More generally, this class is useful at the end of the simulation when analyzing the results.

4.1.1 Input parameters

Since the amount of information in input is high, it was decided to provide a file in JSON format that contains key parameters/configuration value. The configuration file **model_config.json** is located in the root of the project. The parameters set in this file are read from the singleton object of type ModelConfig, which is accessible from all classes in the project.

The configuration parameters allow the user to select the algorithms that were introduced in Chapter 3 on the Project. The user has the option of entering the following parameters:

- The number of nodes that compose the validator pool is described via the parameter **n_agents** which must be set as a positive integer.
- The number of epochs that compose a simulation is described via the parameter **n_epochs** and must be a positive integer. The greater the number of epochs, the greater the time required to run a simulation.
- The parameter **pos_type** allows selection of the PoS consensus variant, and can take the following values: *random*, *weighted*, *coin_age*, and *dynamic_weighted.*
- The parameter **initial_stake_volume** allows specifying the total amount of initial stake, which is distributed to the nodes via the specified *initial_distribution* mechanism.
- The mechanism for distributing the amount of *initial_stake_volume* is selected via the parameter **initial_distribution**, which can take the following values: *gini* in

the case of wanting to initialize the initial state with a specified Gini index, *constant* in the case of constant initialization, *linear* in the case of linear initialization, or *polynomial* in the case of polynomial initialization.

- In case the *initial_distribution* parameter is set as *gini*, then it is necessary to specify the parameter **gini_initial_distribution**, which allows to enter the Gini coefficient for the initial distribution.
- In case it is preferred to use a custom initial distribution, it is possible to specify the distribution list directly via the parameter **custom_distribution**. In this case, the initial volume will be automatically calculated as the sum of the custom distribution.
- The reward type can be selected via the parameter **reward_type**, which can take one of the following values: *constant* or *geometric*.
- The amount of reward to be distributed throughout the simulation to agents is set via the decimal parameter **total_reward**.
- The amount of epochs for which agents cannot be selected for block generation can be set. The parameter **stop_epoch_after_validator** sets this amount when a validator has been selected for block generation.
  The parameter **stop_epoch_after_malicious**, on the other hand, will be triggered when an agent is discovered to be a malicious node.
- The probability of a node behaving maliciously can be set by means of the parameter **malicious_node_probability** which has to be a decimal number between 0 and 1. In case a node is found to be malicious then it will be punished by a stake reduction equal to the percentage of **percent_stake_penalty**.
- It is possible to set a limit to the stake possessed by validators by means of the decimal parameter **stake_limit**.
- In case the selected *pos_type* is *coin_age*, it is possible to specify the coin age reduction factor, which is triggered when a validator is selected, via the decimal parameter between 0 and 1 *coin_age_reduction_factor*. The minimum and maximum limit that the coin age can take can also be set via the decimal parameters *min_coin_age* and *max_coin_age*, respectively. In case a validator does not possess the minimum coin age, it cannot take part in the validation process.
- In the case where the *pos_type* is set to *dynamic_weighted*, it is possible to specify the threshold to be applied via the decimal parameter **gini_threshold** between 0 and 1. In this case, it is also necessary to specify the factor θ via the decimal parameter **dynamic_weighted_theta**.

An example of how the *model_config.json* file is constructed is shown below.

```json
{
    "n_agents": 10,
    "n_epochs": 50000,
    "pos_type": "dynamic_weighted",
    "initial_stake_volume": 1000.0,
    "initial_distribution": "gini",

    "gini_initial_distribution": 0.7,
    "custom_distribution": [1,2,3,4,5,6,7,8,9,10],

    "reward_type": "constant",
    "total_rewards": 1000.0,

    "stop_epoch_after_validator": 0,

    "stop_epoch_after_malicious": 5,
    "malicious_node_probability": 0.0,
    "percent_stake_penalty": 0.2,

    "stake_limit": 999999999999999.0,

    "coin_age_reduction_factor": 0.5,
    "min_coin_age": 0,
    "max_coin_age": 999999999999999,

    "gini_threshold": 0.7,
    "dynamic_weighted_theta": 0.4
}
```

Listing 4.1: Example of model_config.json

4.2 PoS variants

The following will describe some parts that represent the implementation of the **variants of PoS** presented in the previous chapters. The most important parts of code for the performance of the PoS consensus will be shown, while the omitted parts of code can still be further explored.

The selection of validators, performed in the PoS variants, uses the library *random*, included in Python. Other pseudo-random number generation libraries, such as *cryptography*, were initially considered. The *cryptography* library generates cryptographically secure pseudo-random numbers, for which it is extremely difficult to predict or determine successive numbers based on previous numbers. It employs more complex and robust cryptographic algorithms to generate cryptographically secure random numbers. However, these algorithms are slower than the algorithms used in random. It was decided to choose the *random* library to generate random numbers because the **requirements** of the project do not require the generation of cryptographically robust numbers, moreover, generation via *random* is more **efficient**, therefore it results in a faster execution of the simulation.

Random number generation is used for validator selection in PoS variants. It can be seen that the main difference in PoS variant implementations is actually the validator selection algorithm.

4.2.1 The PoS class

All PoS variants have been implemented as subclasses of the abstract PoS class. The PoS class contains the implementation of the **shared methods** for all PoS variants. In particular, the constructor initializes the model configuration parameters, the agents that compose the validator pool, the current epoch, the history of all epochs up to the current one, and the agents that are excluded from the validation process.

Initialization of the initial distribution is done within the *create_ agents* method, which initializes the agents and associated stakes by reading the configuration parameters.

The *step* method is responsible for the execution of a single epoch: it updates the current epoch number, applies the consensus mechanism, and updates the epoch history with information regarding the stakes of all agents in the current epoch.

The consensus algorithm is implemented in the *consensus* method. This method is responsible for selecting a validator and assigning it the block reward.

The *run* method is responsible for performing all the epochs set in the configuration file, one at a given time. The function *tqdm* is also responsible for showing the progress of the model in the console.

```python
class PoS:

    def __init__(self):
        self.conf = ModelConfig()
        self.agents: Set[NodeAgentV1] = set()
        self.stake_volume = 0
        self.epoch = 0
        self.history = []
        self.agents_stop_epochs = set()

    def step(self):
        self.epoch += 1
        self.consensus()
        self.update_stop_epochs()
        self.update_history()

    def run(self):
        for _ in tqdm(range(self.conf.n_epochs)):
            self.step()
        self.history = pd.DataFrame.from_dict(self.history)

    def update_history(self):
        self.history.extend([
            {
                'epoch': self.epoch,
                'id': a.unique_id,
                'stake': a.get_stake()
            } for a in self.agents
        ])

    def create_agents(self):
        stakes = Distributions.generate_distribution(
            self.conf.initial_distribution,
            self.conf.n_agents,
            self.conf.initial_stake_volume
        )
        for i in range(self.conf.n_agents):
            self.agents.add(NodeAgentV1(i, stakes[i]))
            self.stake_volume += stakes[i]
        self.agents = frozenset(self.agents)
        self.update_history()

    def consensus(self):
        validator = self.select_validator()

        ... # Omitted the malicious node code

        validator.stop_epochs += self.conf.stop_epoch_after_validator
        old_stake = validator.stake
        validator.stake += self.get_block_reward()
        if validator.stake > self.conf.stake_limit:
            validator.stake = self.conf.stake_limit
        self.stake_volume += (validator.stake - old_stake)

        if validator.stop_epochs > 0:
            self.agents_stop_epochs.add(validator)
```

Listing 4.2: Validator selection in PoS

4.2.2 RandomPoS

The first variant of PoS that is described is the **RandomPoS**, that is, the variant that is simulated when the configuration parameter *pos_ type* is set to *random*.

The main part of this variant is to randomly select an agent belonging to the validator pool.

It is noted that, in this case, agents that have stop epochs, in which they cannot be selected, are excluded from the validation process. This is the case when, for example, in the model configuration, the parameter *stop_ epoch_ after_ validator* is an integer greater than 0. In this case, the validator cannot be selected for the specified amount of epochs.

```
import random

class RandomPoS(PoS):
    ...

    def select_validator(self):
        agents = list(self.agents.difference(self.agents_stop_epochs))
        return random.choice(agents)
```

Listing 4.3: Validator selection in RandomPoS

4.2.3 WeightedPoS

The WeightedPoS variant can be executed when the parameter *pos_ type* in the configuration is set to *weighted*. This variant is able to select the validator in a weighted manner on the stake owned by the agents. The probability of selecting the validator, in this case, corresponds to the amount of stake possessed by the validator divided by the total volume of stakes. In this case, it is possible to specify the weights within Python's *random.choices* method, which extracts the validator from the specified list of agents.

```python
import random

class WeightedPoS(PoS):
    ...

    def select_validator(self):
        agents = list(self.agents.difference(self.agents_stop_epochs))
        return random.choices(
            agents,
            weights=[a.stake for a in agents]
        )[0]
```

Listing 4.4: Validator selection in WeightedPoS

4.2.4 CoinAgePoS

The coin age variant can be selected from the configuration via the *pos_type: 'coin_age'* parameter in the configuration file. The implementation in this case requires an override of the methods *consensus* and *step*, to integrate the coin age update of the validators into the consensus.

In this case, the selection of the validator is done by a weighted random extraction, where the weight of each agent corresponds to the coin age, obtained by the method *get_coin_age_weight*.

```python
import random

class CoinAgePoS(PoS):
    ...

    def update_coin_ages(self):
        for a in self.agents:
            a.coin_age += 1

    def step(self):
        self.epoch += 1
        self.update_stop_epochs()
        self.update_coin_ages()
        self.consensus()
        self.update_history()

    def select_validator(self):
        return random.choices(
            list(self.agents),
            weights=[a.get_coin_age_weight() for a in self.agents
        ])[0]

    def consensus(self):
        validator = self.select_validator()

        ... # Omitted the malicious node code

        validator.stop_epochs += self.conf.stop_epoch_after_validator
        validator.coin_age -= math.ceil(self.conf.
    coin_age_reduction_factor * validator.coin_age)
        old_stake = validator.stake
        validator.stake += self.get_block_reward()
        if validator.stake > self.conf.stake_limit:
            validator.stake = self.conf.stake_limit
        self.stake_volume += (validator.stake - old_stake)

        if validator.stop_epochs > 0:
            self.agents_stop_epochs.add(validator)
```

Listing 4.5: Validator selection in CoinAgePoS

4.2.5 DynamicPoS

The dynamic variant can be performed through the parameter *pos_ type* valued with *'dynamic_ weighted'* in the configuration file. This variant involves dynamically decreasing the Gini coefficient when it exceeds a certain threshold set as input with the parameter *gini_ threshold*.

The function *reduce_ gini_ transform* allows the Gini coefficient to be lowered through the use of a linear function that "approximates" the weights obtained from the agents' stakes, as described in Chapter 3.

```
import random
class DynamicPoS(PoS):

    def linear_function(self, x1, y1, x2, y2):
        m = float(y2 - y1) / (x2 - x1)
        q = y1 - (m * x1)
        return m, q

    def theta_function(self, original_gini):
        m, q = self.linear_function(
            0.5,
            0.005,
            0.7,
            0.1
        )
        return m * original_gini + q

    def reduce_gini_transform(self, data, theta):
        m, q = self.linear_function(
            0.0,
            theta,
            1,
            1 - theta
        )
        return m * data + q

    def weights_transformation(self, weights, original_gini):
        theta = 0
        if original_gini >= self.conf.gini_threshold:
            theta = self.theta_function(original_gini)
        transformed_data = self.reduce_gini_transform(
            weights,
            theta
        )

    def dynamic_gini_weighted(self):
        agents = list(self.agents.difference(self.agents_stop_epochs))
        data = np.array([a.stake for a in self.agents])
        weights = data / data.sum()
        gini = gini_concentration_index(weights)
        transformed_data = self.weights_transformation(weights, gini)
        return random.choices( \
            agents, \
            weights=transformed_data \
        )[0]

    def select_validator(self):
        return self.dynamic_gini_weighted()
```

Listing 4.6: Validator selection in DynamicPoS

4.3 Initial distribution

The initial distribution is selected using the parameter *initial_ distribution* in the configuration file. The methods for generating the initial distributions are implemented in the class **Distributions**, which contains the implementation of the methods described in Chapter 3.

```
class Distributions:

    @staticmethod
    def generate_distribution(dist_type: str, n: int, volume: float):
        match dist_type:
            case "constant":
                return Distributions.constant(n, volume)
            case "linear":
                return Distributions.linear(n, volume)
            case "polynomial":
                return Distributions.polynomial(n, volume, 100)
            case "custom":
                return ModelConfig().custom_distribution
            case "gini":
                return Distributions.gini(
                    n,
                    volume,
                    ModelConfig().gini_initial_distribution
                )
            case _:
                raise ModelConfigException(f"{dist_type} not found")

    @staticmethod
    def constant(n_agents, stake_volume):
        stake = stake_volume / n_agents
        return [stake for _ in range(n_agents)]

    @staticmethod
    def gini(n: int, volume: float, gini: float):
        def lorenz_curve(x1, y1, x2, y2):
            m = float(y2 - y1) / (x2 - x1)
            return lambda x: m * x

        max_r = (n - 1) / 2
        r = gini * max_r
        prop = ((n - 1) / n) * ((max_r - r) / max_r)
        lc = lorenz_curve(0, 0, (n - 1) / n, prop)
        q = [lc(i / n) for i in range(1, n)] + [1]
        cumulate_sum = [i * volume for i in q]
        stakes = [cumulate_sum[0]] \
                    + [cumulate_sum[i] \
                        - cumulate_sum[i - 1] for i in range(1, n)]
        return stakes

    @staticmethod
    def constant_np(n_agents, stake_volume):
        stake = stake_volume / n_agents
        return np.full(n_agents, stake)

    @staticmethod
    def linear(n_agents, stake_volume):
        m = stake_volume / (.5 * (math.pow(n_agents, 2) + n_agents))
        stakes = []
        for i in range(1, n_agents + 1):
            stakes.append(m * i)
        return stakes

    @staticmethod
    def polynomial(n_agents, stake_volume, degree=2):
        sum = 0
        for i in range(1, n_agents + 1):
            sum += math.pow(i, degree)
        m = stake_volume / sum
        stakes = []
        for i in range(1, n_agents + 1):
            stakes.append(m * math.pow(i, degree))
        return stakes
```

Listing 4.7: Definition of initial distributions

4.4 Parallel executions

This section describes the method used to obtain multiple parallel runs of the same model, and how they are then merged within a single dataset.

Parallel executions are essential for obtaining more accurate and complete results. First of all, parallel execution allows for maximum use of available computing resources, particularly multi-core processors. Indeed, by launching a single execution, only one core is exploited, which executes the entire simulation sequentially, leaving the other cores in an idle state.

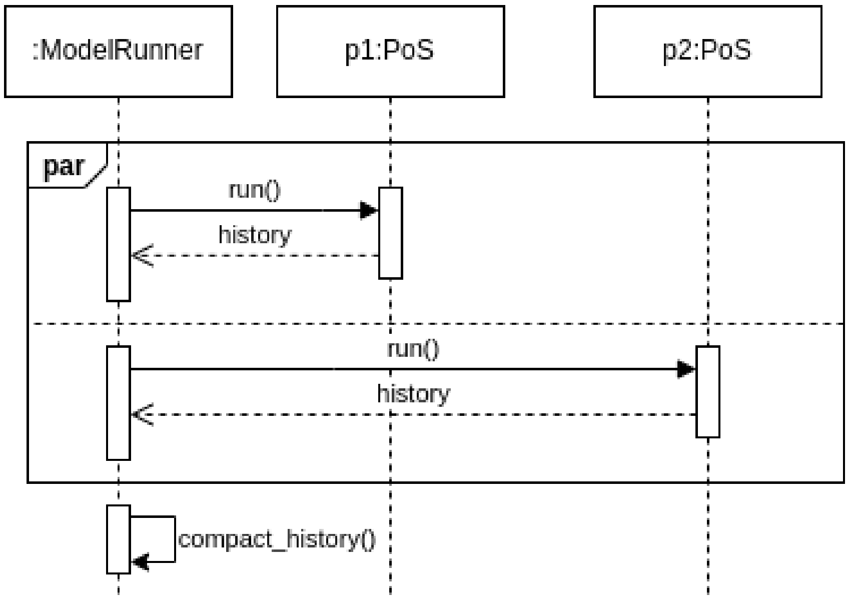
In addition, parallel execution offers a level of scalability, allowing the number of executions performed to be increased or decreased depending on the resources available on the computer. For example, a processor with 8 cores is capable of running 8 executions simultaneously, while a quad-core processor is only capable of running 4 executions simultaneously.

Parallel executions also allow the stability of the results to be evaluated. By running multiple simulations in parallel, the mean and variance of the results obtained can be calculated. This provides a more accurate estimate of the statistical measures, allowing the robustness of the obtained results to be evaluated in terms of mean and variance.

The methods used in this section were inspired by Monte Carlo methods. The basic idea of Monte Carlo methods is to use random numbers to estimate or approximate complex quantities or outcomes that would be difficult or expensive to compute deterministically.

Figure 4.2 shows a diagram that takes 2 parallel executions as an example. Once all parallel executions are completed, the datasets for the histories of each execution are merged into a single dataset.

Figure 4.2: Parallel executions flow



The code that was written to run the simulations in parallel is shown in the following. The main method is *run*, which creates a pool in which the number of parallel processes is set: by default it is given as the number of cpu cores. The number of simulations that want to be run is also set: by default 1 simulation is given. Python's *multiprocessing* library takes care of running the simulations in parallel, where each individual simulation is launched through the *run_ model* method, which selects the PoS variant indicated in the configuration file and executes it.

All runs are then merged into a single dataset, in which the *simulation* column is added to identify which parallel simulation is being referred to.

```python
import pandas as pd
import multiprocessing as mp
...

class ModelRunner:

    @staticmethod
    def model_selector():
        pos_type = ModelConfig().pos_type
        match pos_type:
            case "random":
                return RandomPoS()
            case "weighted":
                return WeightedPoS()
            case "dynamic_weighted":
                return DynamicWeighted()
            case "coin_age":
                return CoinAgePoS()
            case _:
                raise Exception(f"No model matches the type {pos_type}"
   )

    @staticmethod
    def run_model(return_list):
        model = ModelRunner.model_selector()
        model.run()
        return_list.append(model.history)

        return return_list

    @staticmethod
    def run(n_simulations=1, n_processors=mp.cpu_count()):
        pool = mp.Pool(n_processors)
        manager = mp.Manager()
        return_list = manager.list()

        for _ in range(n_simulations):
            pool.apply_async(  \
                ModelRunner.run_model,  \
                args=(return_list,)  \
            )
        pool.close()
        pool.join()

        # Concatenation of the simulations in one dataset
        compact_history = [return_list[i].assign(simulation=i) \
            for i in range(len(return_list))]
        compact_history = pd.concat( \
            compact_history, \
            ignore_index=True \
        )

        return compact_history
```

Listing 4.8: Run multiple simulations in parallel

Chapter 5

Simulations Results

The current chapter shows the results of the simulations of the Python library coded for this thesis, performed with different initial inputs and PoS variants
The main focus is to evaluate the behaviour of different PoS variants in terms of wealth and centralization of the network through the analysis of the Gini coefficient and the g_s, g_r metrics applied to it.

All the built variants presented in the mathematical model will be taken into consideration, as random PoS, weighted PoS, coin-age PoS and dynamic PoS. Since each of these models has unique characteristics that influence the selection of validators and the distribution of wealth in the blockchain network, then for each of them the output data of the experiment will be shown and analyzed.

Lastly, will be presented an experiment in which the metrics g_s and g_r will be applied to the PoS variants to identify the optimal model among all the variants considered. This approach will help us better understand which PoS model may be best suited to meeting the needs of a blockchain in terms of wealth distribution.

5.1 Random PoS

The **random PoS** is a type of proof of stake that selects a validator randomly among the set of agents $A = \{A_1, A_2, ..., A_m\}$. The validators will be selected randomly without considering their stake, in this sense, the amount of stake that a node holds does not influence their likelihood of being selected as a validator. This behaviour eliminates any preference based on stake quantity and makes the selection of the next validator a random event in the blockchain network.

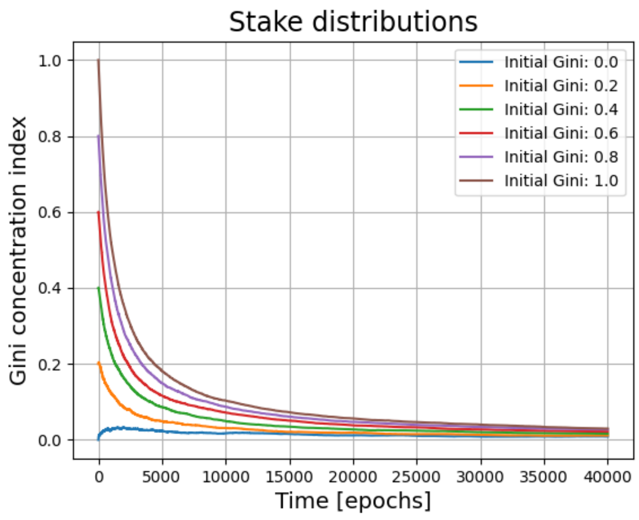
For the simulation, the following parameters are kept fixed:

Parameter	**Value**
Parallel executions	4
Number of agents	10
Number of epochs	40k
Initial Stake Volume	1k
Total Rewards	40k
Reward type	Constant

To model the behavior of the random PoS it was decided to execute the model with different initial wealth distributions, and in particular to generate initial distributions with different **Gini concentration coefficients**: 0, 0.2, 0.4, 0.6, 0.8 and 1. Each simulation executes four indipendent models with the same initial input parameters, in this way it is possible to see the mean and the standard deviation of each simulation.

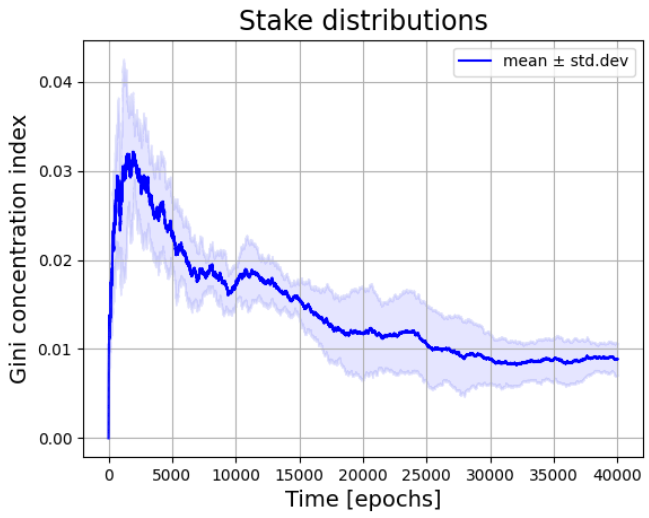
The plot in Figure 5.1 shows that starting from any initial stakes distribution, the value of the Gini coefficients tends to 0 as the time tends to ∞.

Figure 5.1: Evolution of Gini concentration index with different inititial stakes



The only exception is the simulation starting from the initial Gini value 0 (the blue graph), where after a few periods the simulation stabilizes. Also this graph has a coefficient which tends to 0 as the time tends to ∞, as illustrated in Figure 5.2.

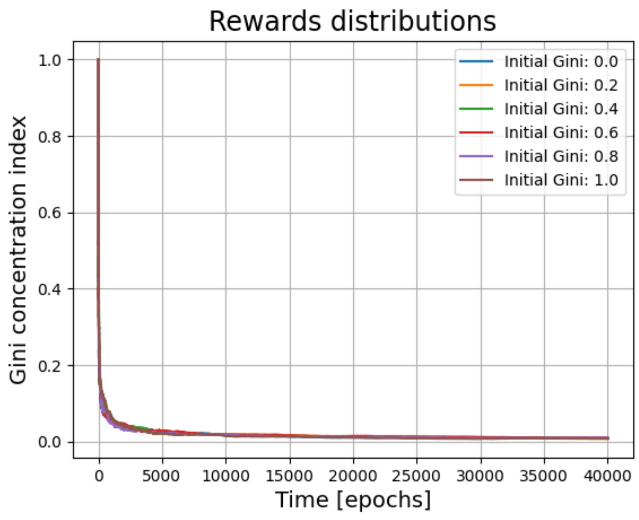
Figure 5.2: Statistical behaviour of the simulation with initial Gini concentration index set to 0



The behavior of this type of PoS can also be viewed as a **reward distributions trend**, i.e. without taking into consideration the initial stakes.

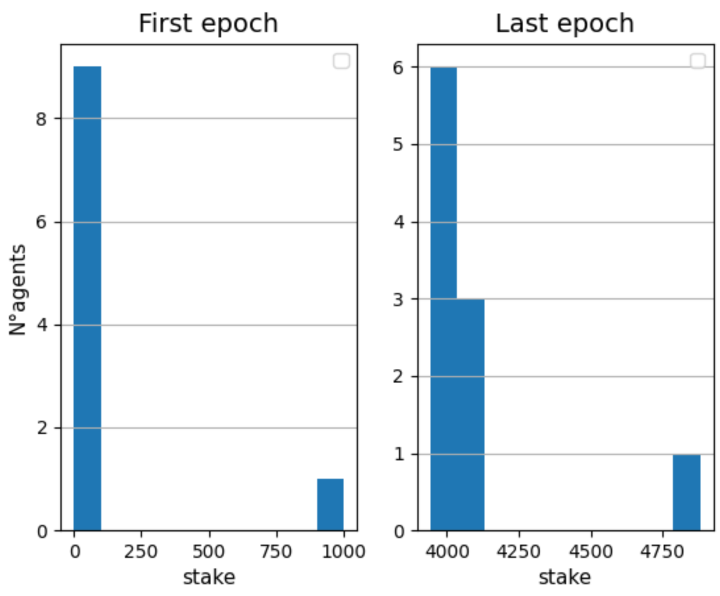
In this case, since the rewards do not depend on the stakes, the Gini concentration index of the reward functions is very similar for all the initial distributions, as shown in Figure 5.3, and tends to 0 as time tends to ∞. This indicates a trend towards greater equity in the distribution of wealth in the Random PoS system as time passes.

Figure 5.3: Evolution of Gini concentration index of the rewards for different initial ditributions



Consider the simulation with the initial stake distribution with the Gini coefficient set to 1, i.e. in case of maximum concentration. Since the total stake is set to 40000 and these are distributed uniformly on the nodes, the expected reward value for each node is given by the total rewards divided by the number of agents, i.e. $40000/10 = 4000$. Observing the variation of stakes between the first epoch (where no validator has yet been selected) and the last epoch, we can see in the histogram in Figure 5.4 that the stakes have **increased by** approximately **4000** for each node.

Figure 5.4: Histograms of first and last epochs for initial distribution with Gini concentration index set to 1



5.1.1 Metrics Analysis

To verify the behavior of the model with random PoS, simulations were analyzed as some input parameters varied. In particular, **60** simulations were carried out with 4 parallel executions for each simulation.

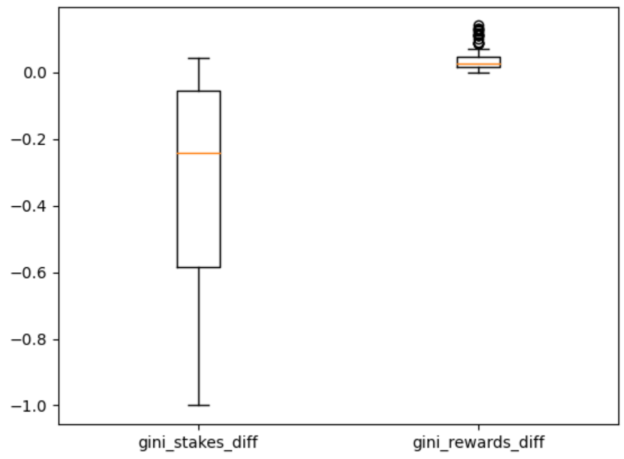
Parameter	**Values**
Parallel executions	4
Number of agents	10
Number of epochs	10k
Initial Stake Volume	1k
Total Rewards	100, 1k, 10k, 100k, 1M
Reward type	Constant, Geometric
Gini initial Distribution	.0, .2, .4, .6, .8, 1.0

For this test, two metrics seen in the previous chapter were taken into consideration, namely g_s and g_r, which calculate the difference between the Gini coefficient of the stakes and rewards at the last epoch and those in the first epoch.

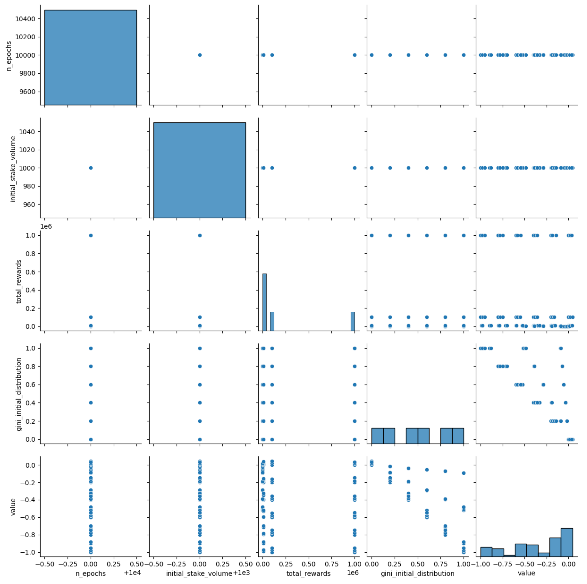
If a metric is positive, it means that an increase in inequality has occurred, while if the metric is negative, a decrease in inequality has occurred. The optimum is calculated as the zero difference, i.e. the one where the Gini coefficient has remained unchanged.

The boxplot in Figure 5.5 shows how the two metrics are distributed. In the case of the metric based on the difference between the stakes, it is possible to notice a value of -0.3432 $\pm$ 0.3249, indicating a reduction of the Gini coefficient. The metric g_r shows a value of 0.039326 $\pm$ 0.032431, indicating a metric that is always positive and close to the optimal value.

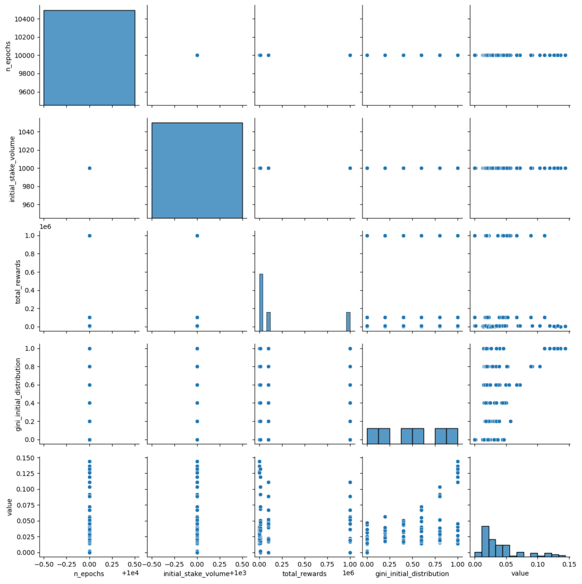
Figure 5.5: Boxplots of the g_s metric and g_r metric in Random PoS



The pairplot is useful for exploring data and showing whether relationships exist between variables. In particular, it is possible to note in Figure 5.6a, calculated on the g_s metric, how the most evident relationship is the one between the Gini initial distribution and the value. In this case the graph seems to follow 3 separate and different trends, some tend to go towards the maximum equality, while others tend to go towards the minimum equality. This behavior is explained by the different input parameters, when the total reward is low, the Gini coefficient tend to remain similar to the initial one. On the contrary, when the total reward is higher then it is possible to appreciate the increase in equality. In case of the g_r the Figure 5.6b, shows a small positive correlation between the initial Gini and the value.



(a) Gini stakes difference metric



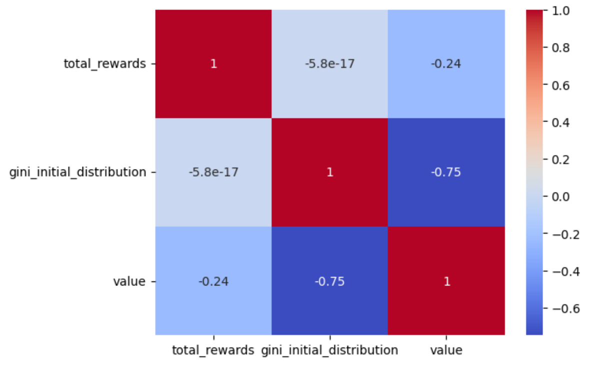
(b) Gini reward difference metric

Figure 5.6: Random PoS pairplot for g_s and g_r metrics

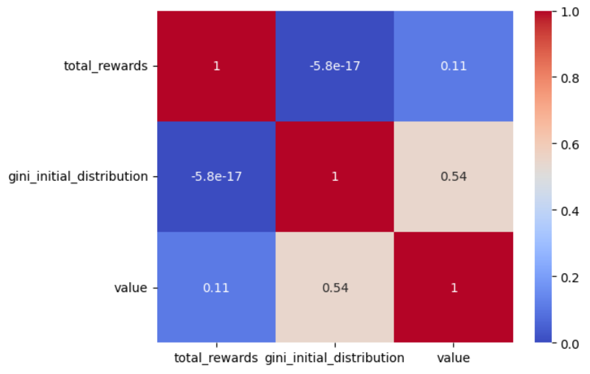
The graphs 5.7a and 5.7b show the heatmap of the correlation matrix.
The Figure 5.7a shows that there is a negative correlation of -0.75 between the the initial

Gini coefficient and the g_s metric, to indicate that the Gini initial distribution has a good effect on the outcome of this metric. It also indicates that there is a small negative correlation of -0.24 between the g_s metric and total rewards.

The Figure 5.7b instead shows how the g_r metric has a lower but always positive correlations, with a value of 0.11 for the total reward and 0.54 for the initial Gini coefficient.



(a) Gini stakes difference metric



(b) Gini reward difference metric

Figure 5.7: Random PoS correlation heatmap for g_s and g_r metrics

5.2 Weighted PoS

The **weighted PoS** is a type of PoS variant that selects a validator, based on the amount of stake, among the set of agents $A = \{A_1, A_2, ..., A_m\}$. Unlike the other PoS variants, the weighted PoS selects validators with a likelihood poportional to their stake. In this sense, the nodes with an high stake have an higher chance of being selected as validators. In this section, the behaviour of the weighted PoS is presented, exploring how it can influence the network wealth distribution in a blockchain.

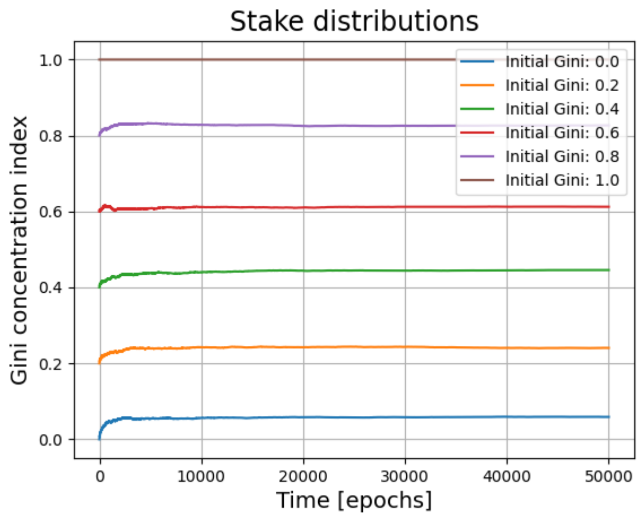
For the simulation, the following parameters are kept fixed:

Parameter	**Value**
Parallel executions	4
Number of agents	10
Number of epochs	50k
Initial Stake Volume	1k
Total Rewards	50k
Reward type	Constant

As in the experiments in the previous section on randomPoS, six simulations are taken into consideration, with **different initial** inputs, and in particular, that differ by the **Gini concentration coefficient** of the initial stake distribution: 0, 0.2, 0.3, 0.4, 0.6, 0.8 and 1.

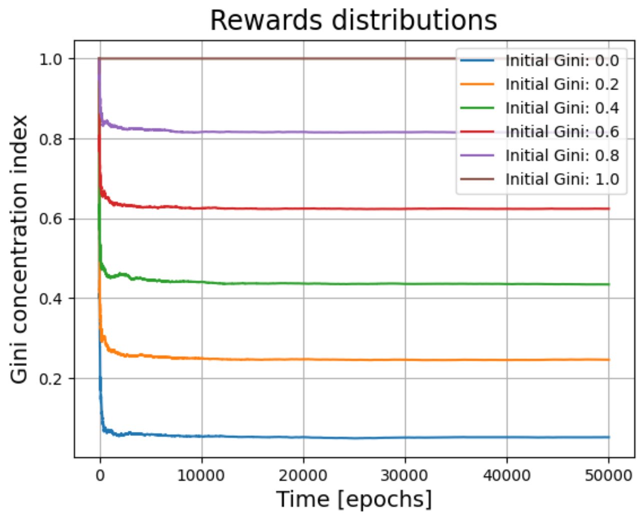
The experiments in Figure 5.8 show that the Gini coefficient tends to increase for the first epochs, before **stabilizing** around a certain threshold. In the case of the initial distribution with the Gini coefficient set to 1, the coefficient shows **no changes**: all wealth is held by a single individual, namely the holder of the total stakes.

Figure 5.8: Evolution of Gini concentration index with different initital stakes for weighted PoS



It is also possible to observe the rewards distribution in Figure 5.9 over time, i.e. the amount of tokens accumulated by each node over the epochs. These simulations show a **initial decreasing** trend that tends to stabilize at a certain value. Unlike the stakes distributions, the Gini coefficient applied to the rewards initially is 1 for all simulations. In this senes, whatever validator is chosen in the first epoch, his quantity of stake will be equal to the total amount of stakes. This configuration leads to a Gini coefficient equal to 1 for that selected node.

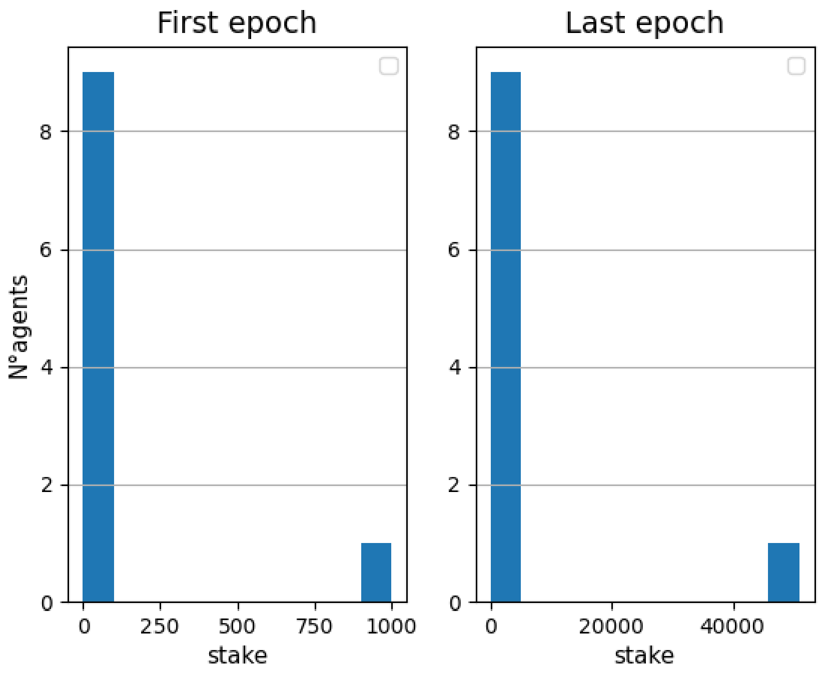
Figure 5.9: Evolution of Gini concentration index on rewards with different initital stakes for weighted PoS



The simulation with the initial Gini coefficient set to 1, in Figure 5.10, shows an expected behavior: **all rewards are attributed to a single individual**, the one who owns the total stakes. Although this is an extreme case, it can be observed that the proof of stake that bases its reward mechanism solely on the weight of the stakes favors the wealthiest, helping to amplify the effect of the rich getting richer.

However, the weighted PoS has a behavior that deserves further exploration. The tendency of the weighted PoS is to maintain the initial Gini. This characteristic has a significant implications on the fairness of a blockchain system since the probability to select a validator is proportional to the stake, the nodes with larger stakes have more probability to being selected as validators. The weighted PoS system can limit opportunities for nodes with smaller stakes to rise influence in the newtork.

Figure 5.10: Histograms of first and last epochs for initial distribution with Gini concentration index set to 1



5.2.1 Metrics analysis

To verify the behavior of the model with weighted PoS, simulations were analyzed as some input parameters varied. These metrics serve as valuable indicators of the impact on wealth distribution throughout the epochs of the simulation. In particular, **60** simulations were carried out with 4 parallel executions for each simulation.

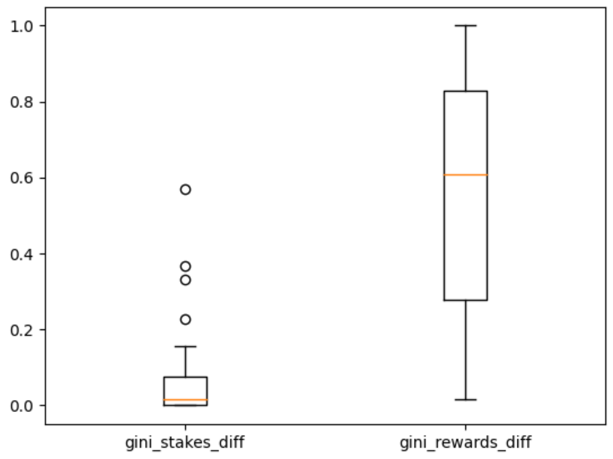
Parameter	**Values**
Parallel executions	4
Number of agents	10
Number of epochs	10k
Initial Stake Volume	1k
Total Rewards	100, 1k, 10k, 100k, 1M
Reward type	Constant, Geometric
Gini initial Distribution	.0, .2, .4, .6, .8, 1

The boxplot in Figure 5.11 shows how the two metrics g_s and g_r are distributed.

Let's start by examining the metric g_s, which measures the difference between the stakes held by various nodes from the beginning to the end of the simulation. This metric shows an interesting trend, with a value of 0.0574 ± 0.1003, indicating an almost optimal maintenance of equality. However, what truly captures our attention is the standard deviation of 0.1003, which indicates that the model behaves in this way with low variability, for all initial Gini coefficients.

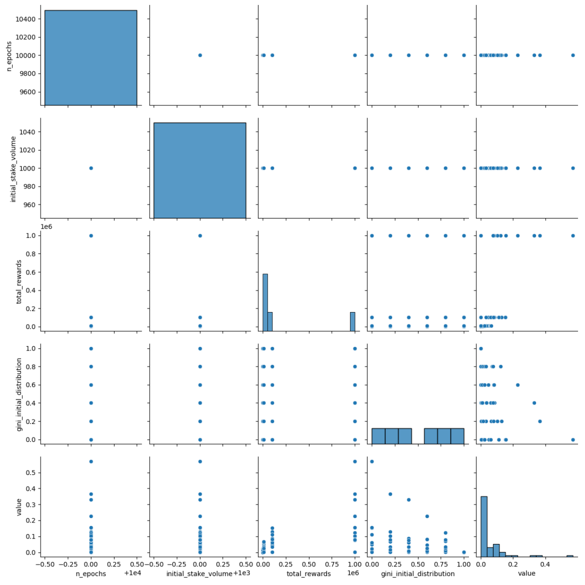
Moving to the metric g_r, we shift our focus to the difference in rewards allocated to nodes throughout the simulation. Here, we observe a value of 0.5624 ± 0.3190. This metric's positive average suggests that, in terms of rewards, the Weighted PoS system tends to favor certain nodes over others as epochs progress. The positive skewness, as shown in the Figure, hints at an accumulation of rewards among a subset of nodes, surely those with larger stakes. This outcome indicates that the weighted PoS, by design, empowers participants with higher stakes, making them more likely to accumulate rewards and ,in result, be healhtier.

Figure 5.11: Boxplots of the g_s metric and g_r metric in Weighted PoS

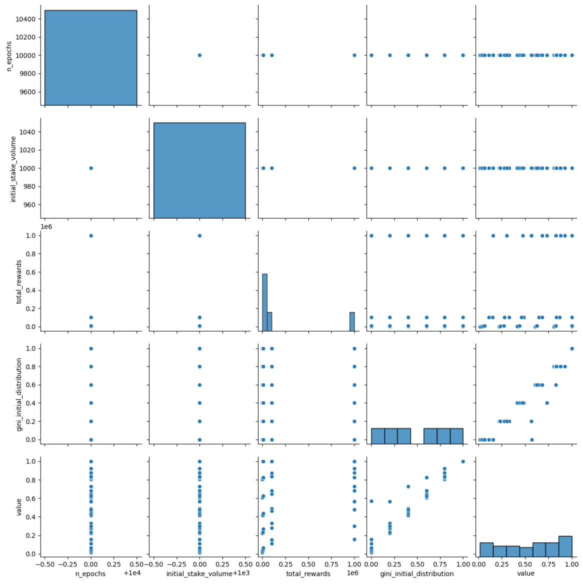


The pairplot in Figure 5.12a illustrates the correlation between total rewards and the g_s value. This correlation emerges looking at the graph between these two variables. As the total rewards increase, g_s demonstrates a tendency to increase. This observation suggests that a lower total reward pool within the blockchain system corresponds to a more equal distribution that remains relatively stable over time.
Also the initial Gini coefficient exhibits a notable correlation with the g_s metric. The initial concentration of wealth plays an important role in shaping the subsequent dynamics of stake redistribution. Specifically, as the initial Gini coefficient increases, the model exhibits an higher equality over time. This phenomenon suggests that a higher level of initial wealth concentration, as indicated by a higher Gini coefficient, corresponds to a less volatile wealth distribution.

The pairplot in Figure 5.12b offers insights for the relationship between the initial Gini coefficient and the value g_r. In this sense, as the initial Gini coefficient increases, we observe a corresponding rise in the g_r value, indicating an increase in wealth inequality within the model. This suggests that a higher initial Gini coefficient contributes to a more unequal wealth distribution over time for the reward distribution.



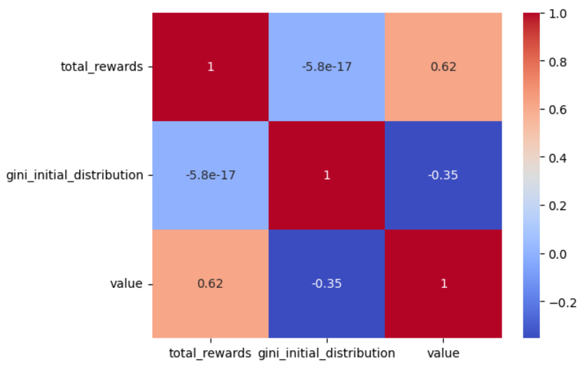
(a) Gini stakes difference metric



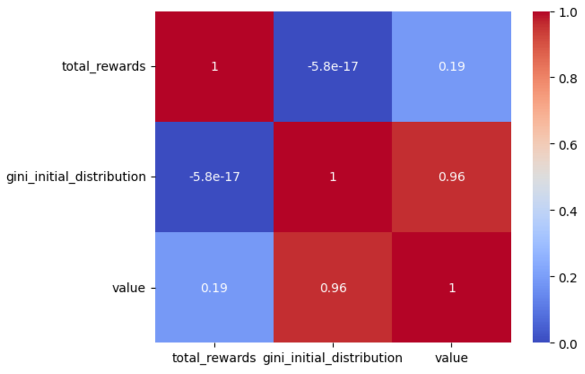
(b) Gini reward difference metric

Figure 5.12: Weighted PoS pairplot for g_s and g_r metrics

The input variable that is most correlated to the final metrics is the initial value of the Gini coefficient. The Figures 5.13a and 5.13b also show how the initial Gini coefficient has a positive correlation of 96% in the case of differences on the rewards; this is due to the growth of inequality for this type of PoS.



(a) Gini stakes difference metric



(b) Gini reward difference metric

Figure 5.13: Weighted PoS correlation heatmap for g_s and g_r metrics

5.3 Coin Age PoS

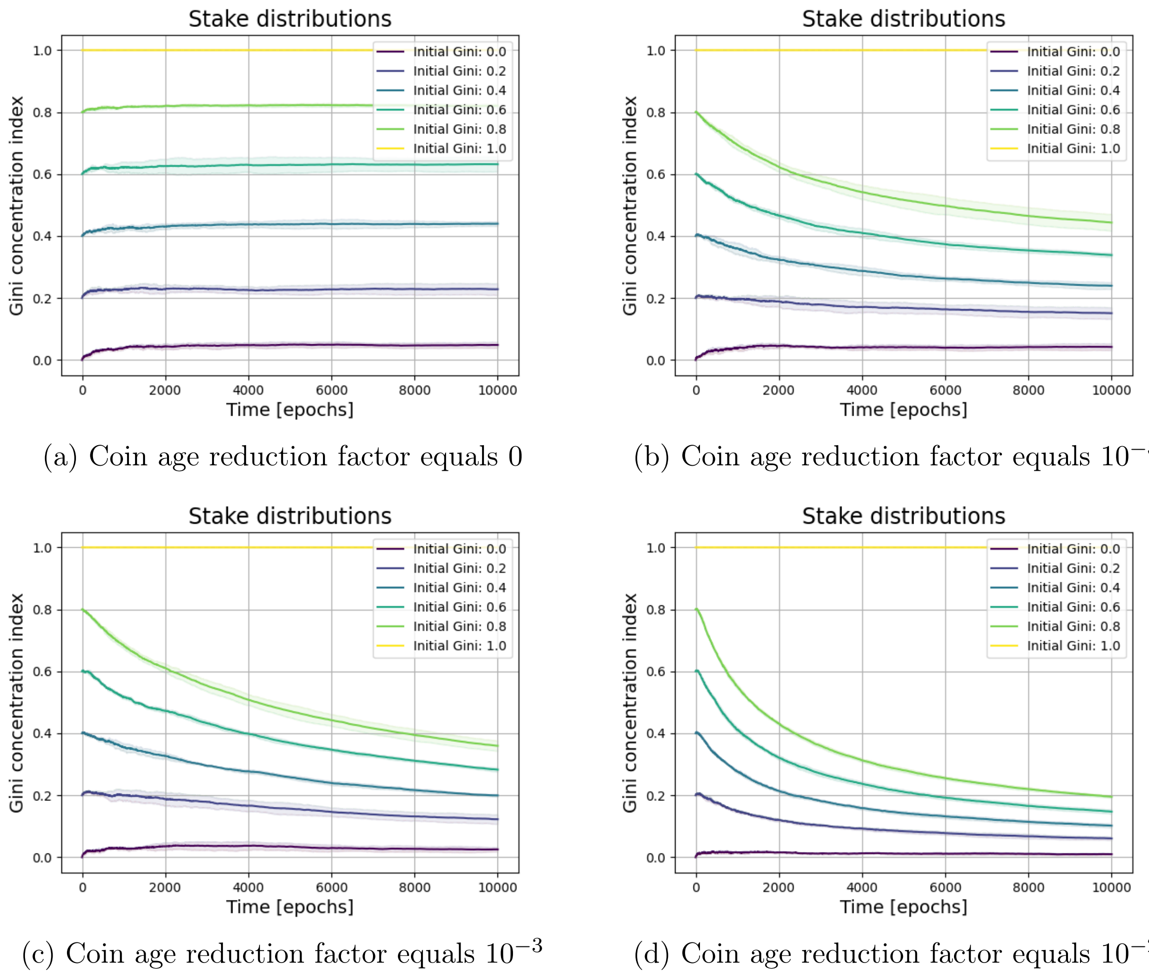
The **Coin Age PoS** is a type of proof of stake that selects a validator among the set of agents $A = \{A_1, A_2, ..., A_m\}$. It can be considered a variant of the weighted PoS where the weights are calulated combining the coin age with the amount of stake.

In these experiments the following parameters are kept fixed:

Parameter	**Value**
Parallel executions	4
Number of agents	10
Number of epochs	10k
Initial Stake Volume	1k
Total Rewards	10k
Reward type	Constant

To show the behavior of the various models, some graphs are displayed as the initial distributions vary, in particular with the following Gini concentration coefficients: 0, 0.2, 0.4, 0.6, 0.8, 1.

Furthermore, the **reduction factor** is also taken into consideration, i.e. the coefficient that regulates by how much the coin age must be decreased if a node is elected as validator. In particular, the graphs showing the trend Gini concentration index will be displayed for the following reduction factor values: 0, 10^{-5}, 10^{-3}, 10^{-2}.



(a) Coin age reduction factor equals 0

(b) Coin age reduction factor equals 10^{-5}

(c) Coin age reduction factor equals 10^{-3}

(d) Coin age reduction factor equals 10^{-2}

Figure 5.14: Coin-Age PoS simulations with different reduction factors

As it can be seen from the graphs of Figures 5.14, the reduction factor influences wealth distributions and the associated Gini coefficient. In particular, it can be seen from the graphs in Figure 5.14a that a reduction factor closer to 0 **increases the Gini coefficient**, i.e., increases wealth inequality. The explanation of this phenomenon can be traced back to the **weighted proof of stake**, in fact, given that the reduction factor does not influence the coin age then all the nodes will have the same coin age at every epoch. Therefore the coin age does not influence the weight of the stakes, which it will therefore be equal to the weight calculated exclusively on the stakes.

Let $w(t)$ be a function which associates each epoch to the sum of the coin age weights, and let $C_{Ai}(t)$ be a function that associates a node A_i at epoch t with the corrisponding coin age.

$$w(t) = \sum_{i=1}^{m} S_{Ai}(t) * C_{Ai}(t)$$

Then the coin age weight of a single node A_i at time t is calculated as the ratio between the coin age weight of the individual A_i and the sum of the coin age weights $w(t)$. This weight will be equal to v_{Ai}, which corresponds to the fractional stake used in the weighted coin age.

$$w_{Ai}(t) = \frac{S_{Ai}(t) * C_{Ai}(t)}{w(t)} = \frac{S_{Ai}(t)}{S(t)} = v_{Ai}(t)$$

The graphs in Figures 5.14b and 5.14c show how an increase in the reduction factor produces stakes distributions with flatten Gini coefficients, which however tend to decrease in contrast to the reduction factor equal to 0 in the plot of Figure 5.14a.

These graphs, together with the graph in Figure 5.14c, indicate that a high reduction factor causes a **gain in stake equality**.

5.3.1 Metrics analysis

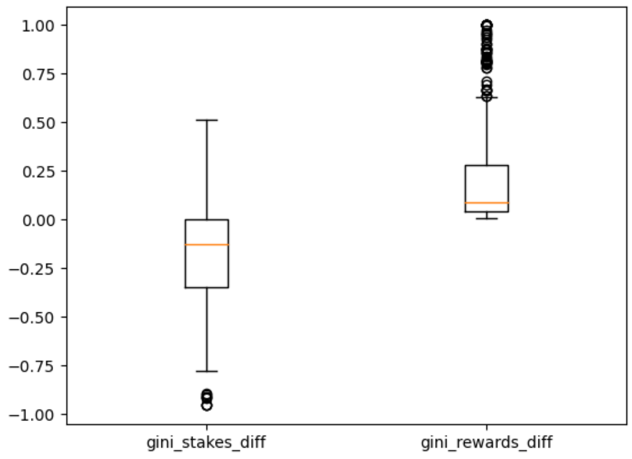
To verify the behavior of the model with coin age PoS, simulations were analyzed as some input parameters varied. In particular, **360** simulations were carried out with 4 parallel executions for each simulation, i.e., a total of 1440 executions.

Parameter	**Values**
Parallel executions	4
Number of agents	10
Number of epochs	10k
Initial Stake Volume	1k
Total Rewards	100, 1k, 10k, 100k, 1M
Reward type	Constant, Geometric
Gini initial Distribution	.0, .2, .4, .6, .8, 1
Reduction factor	.0, .2, .4, .6, .8, 1

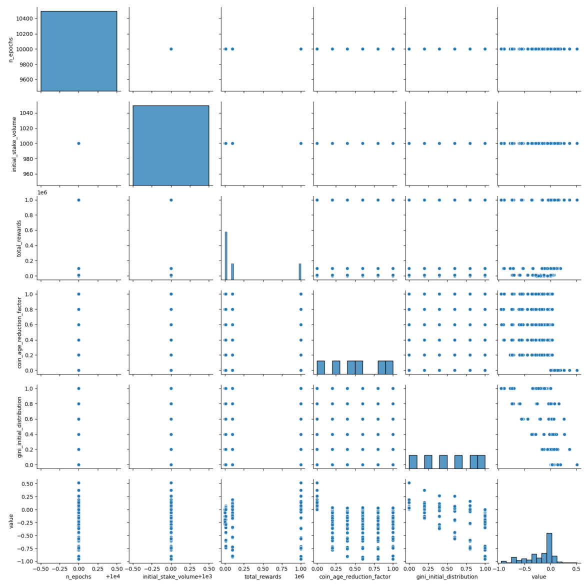
The boxplot in Figure 5.15 shows how the two metrics are distributed. In the case of the metric based on the difference between the stakes, it is possible to notice a value of

-0.1986 ± 0.2705, and for the metric based of the reward difference a value of 0.2370 ± 0.2895

Figure 5.15: Boxplots of the g_s metric and g_r metric in Coin-Age PoS



The pairplot in Figure 5.17a shows how a reduction factor tending towards 0 is correlated to a positive metric, which increases inequality with a Gini coefficient between 0% and 50%. Increasing the reduction factor causes the opposite effect, and is correlated to a tendentially negative metric between 0% and -1%.
However, this behavior is not visible in the rewards-based metric in Figure 5.16b.


(a) Gini stakes difference metric

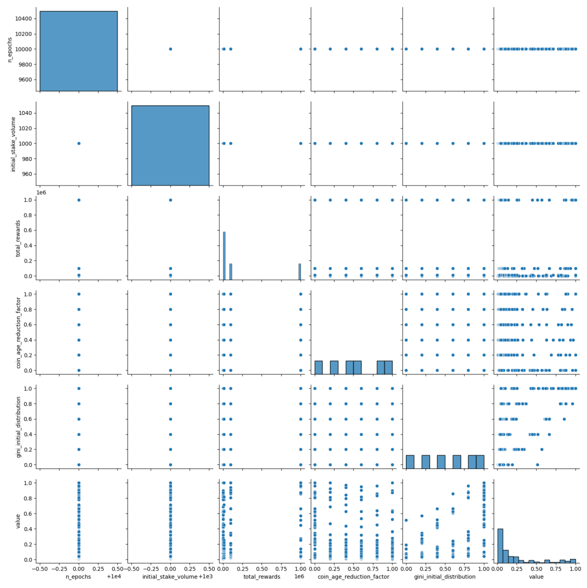
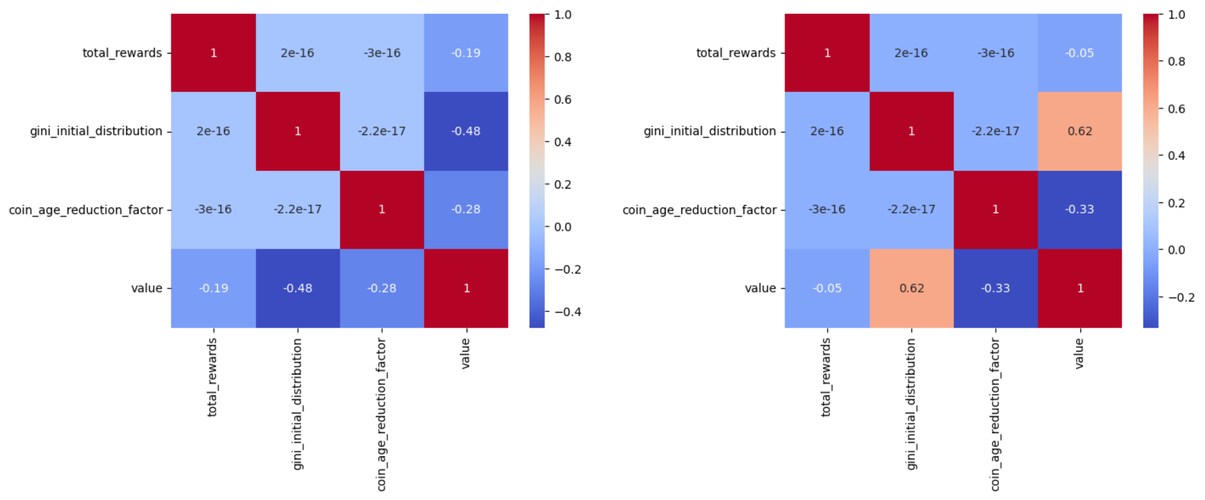

(b) Gini reward difference metric

Figure 5.16: Coin-Age PoS pairplot for g_s and g_r metrics

Plots in Figures 5.17a and 5.17b shows how, although the most significant correlation remains that of the initial Gini coefficient (0.48 and 0.62 respectively), the reduction factor still has a good negative correlation (-0.28 and -0.33) with the metrics considered.



(a) Gini stakes difference metric (b) Gini reward difference metric

Figure 5.17: Coin-Age PoS correlation heatmap for g_s and g_r metrics

5.4 Dynamic PoS

The **Dynamic PoS** is a type of proof of stake that selects a validator among the set of agents $A = \{A_1, A_2, ..., A_m\}$. It can be considered a variant of the weighted PoS where the weights used to extract a validator are calculated differently if the Gini concentration coefficient exceeds a certain threshold.

In these experiments the following parameters are kept fixed:

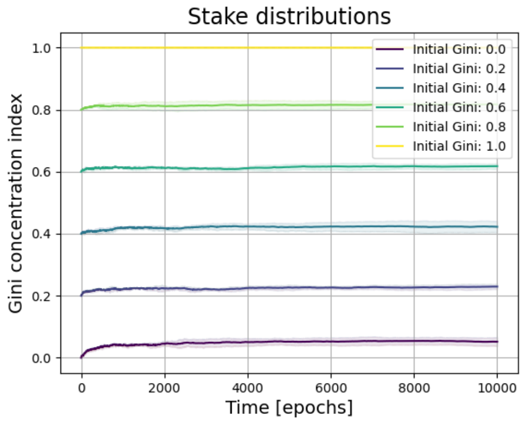
Parameter	**Value**
Parallel executions	4
Number of agents	10
Number of epochs	10k
Initial Stake Volume	1k
Total Rewards	10k
Reward type	Constant
θ function	$\theta(x) = 0.475x - 0.2325$

To show the behavior of the various models, some graphs are displayed as the initial distributions vary, in particular with the following Gini concentration coefficients: 0, 0.2, 0.4, 0.6, 0.8, 1.

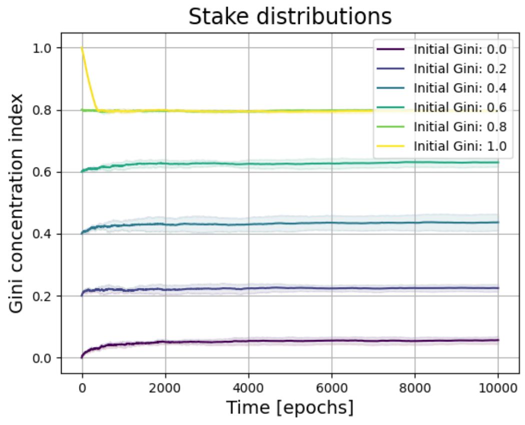
Furthermore, the **Gini threshold** is also used in the simulations, i.e., the threshold that controls when to reduce the Gini concentration index of the weights used to extract a validator. In particular, the graphs showing the trend Gini concentration index will be displayed for the following Gini threshold values: 1, 0.8, 0.6, 0.4.

As it can be seen from the graphs, the value of the Gini threshold set for a given model **influences the distribution of wealth** in the long term.
If the threshold is set to 1 then the dynamic model doesn't have the possibility to activate, since the Gini coefficient cannot by definition exceed this threshold, as shown in

the graph in Figure 5.18a.



(a) Gini threshold set to 1

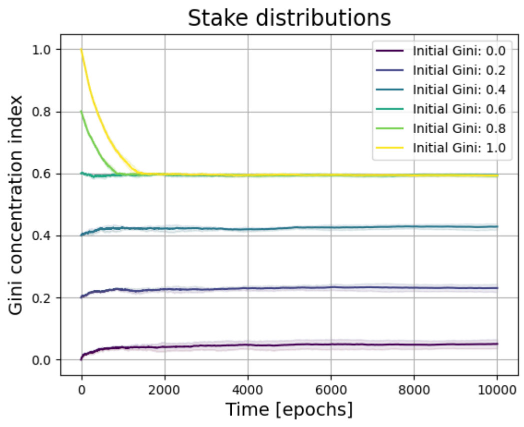


(b) Gini threshold set to 0.8

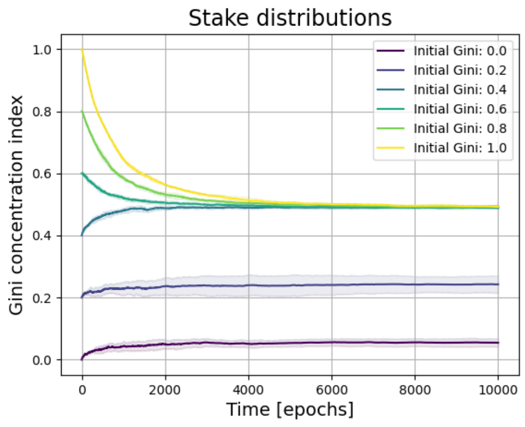
Figure 5.18: Dynamic PoS simulations with 1 and 0.8 Gini thresholds

If this value decreases, a system can be obtained that **progressively reduces the inequality** of wealth up to the Gini threshold. This phenomenon can be observed in the Figures 5.18b and 5.19a in which the Gini concentration coefficient decreases progressively up to the corresponding thresholds of the model.

The Figure 5.19b shows how the Gini coefficient **never reaches the pre-set threshold**. This is due to the parameter θ with which the transformation of the data on the Gini coefficient is carried out and in particular of the function used to calculate this parameter, which in this case is kept fixed between simulations and linear.



(a) Gini threshold set to 0.6



(b) Gini threshold set to 0.4

Figure 5.19: Dynamic PoS simulations with Gini 0.6 and 0.4 Gini thresholds

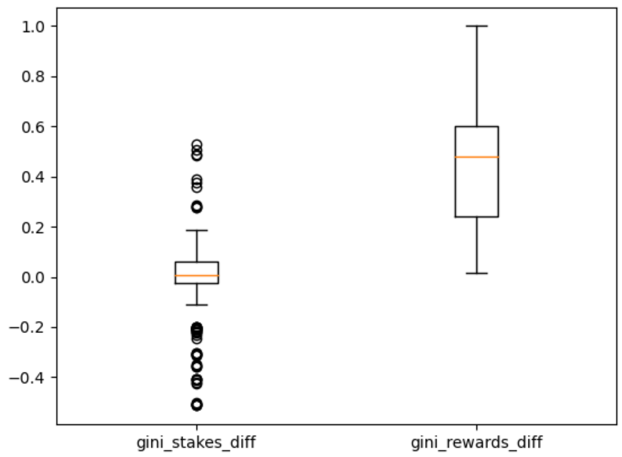
5.4.1 Metrics analysis

To verify the behavior of the model with coin age PoS, simulations were analyzed as some input parameters varied. In particular, **240** simulations were carried out with 4 parallel executions for each simulation, i.e. a total of 960 executions.

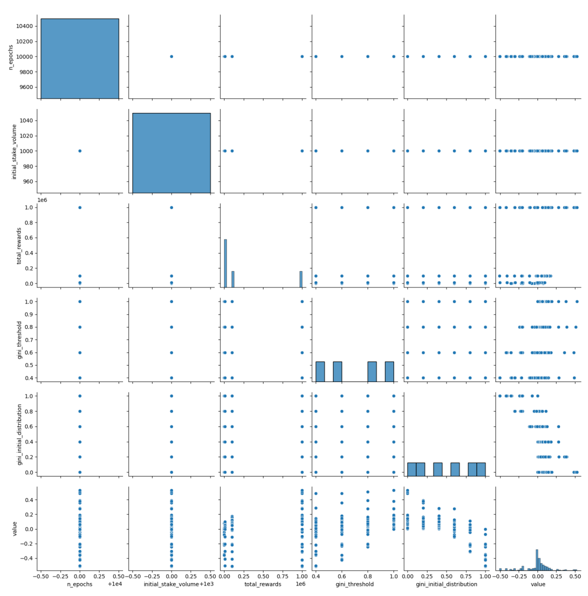
Parameter	**Values**
Parallel executions	4
Number of agents	10
Number of epochs	10k
Initial Stake Volume	1k
Total Rewards	100, 1k, 10k, 100k, 1M
Reward type	Constant, Geometric
Gini initial Distribution	.0, .2, .4, .6, .8, 1
Gini Threshold	.4, .6, .8, 1

The boxplot in Figure 5.20 shows how the two metrics are distributed. In the case of the metric based on the difference between the stakes, it is possible to notice a value of -0.0116 ± 0.1714, and for the metric based of the reward difference a value of 0.4540 ± 0.2521. In this case the g_s metric is very close to the optimal 0 value; however, there are some outliers that have a maximum of 0.5512 and a minimum of -0.4637, meaning that this type of model too could, in some circumstances, increase the level of inequality or decrease it.

Figure 5.20: Boxplots of the stakes g_s and g_r metric in Dynamic PoS



The pairplot in Figure 5.21a shows how increasing the Gini threshold can increase inequality, while the factor that seems to influence the metric the most is the initial Gini value.
In the case of the graph in Figure 5.21b graph, however, the influence of the threshold seems to have less influence on the metric than in the previous graph.


(a) Gini stakes difference metric

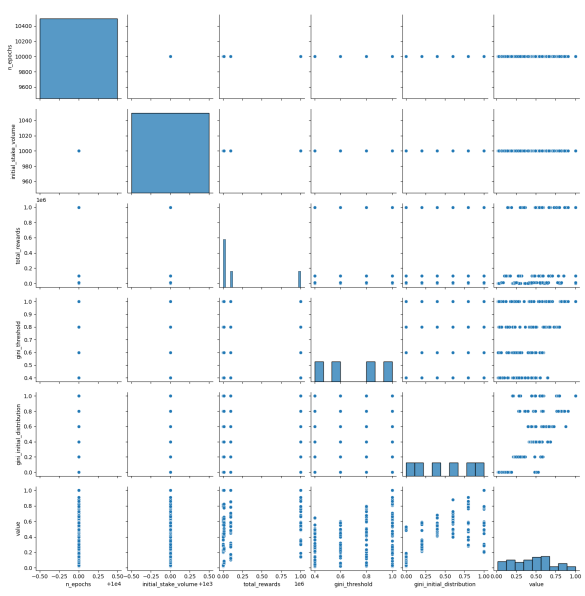
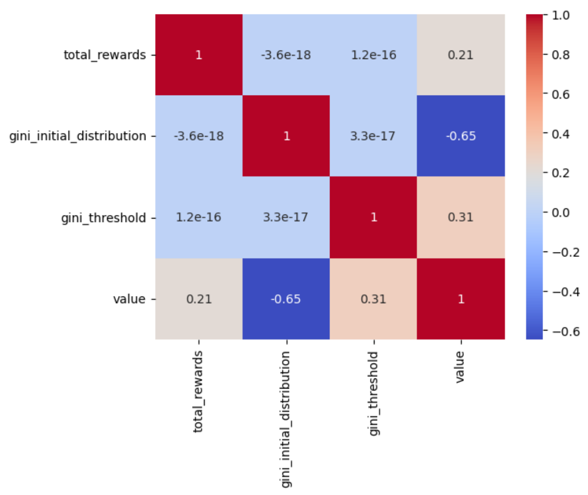
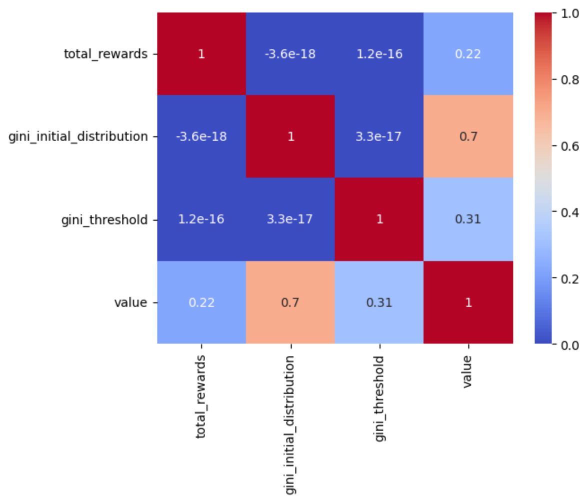

(b) Gini reward difference metric

Figure 5.21: Dynamic PoS pairplot for g_s and g_r metrics

Plots in Figures 5.22a and 5.22b shows how, although the most significant correlation remains that of the initial Gini coefficient (-0.65 and 0.7 respectively), the Gini threshold is correlated equally for both metrics with a value of 0.31.



(a) Gini stakes difference metric

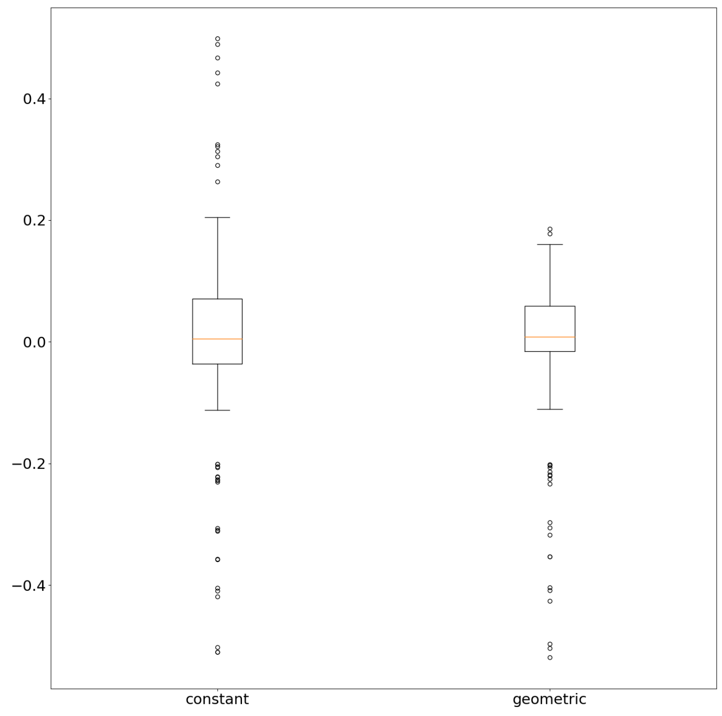


(b) Gini reward difference metric

Figure 5.22: Dynamic PoS correlation heatmap for g_s and g_r metrics

In this case, another factor that influences the metric based on the difference in stakes is the reward type. As it can be seen in the graph of Figure 5.23 the geometric reward type is generally associated with a metric that favors equality; in fact, the values of the g_r metric for the constant reward are between a minimum of -0.5134 and a maximum of 0.4622, while in the case of geometric reward the maximum value is 0.2.

Figure 5.23: Boxplots of the g_S metric and g_r metric, in Dynamic PoS, based on reward type



5.5 Model comparison

An additional experiment was set up to compare the models based on the type of PoS. In this comprehensive experiment, we aimed to conduct a comparison of various PoS variants, including Random PoS, Weighted PoS, Coin Age PoS, and Dynamic PoS.

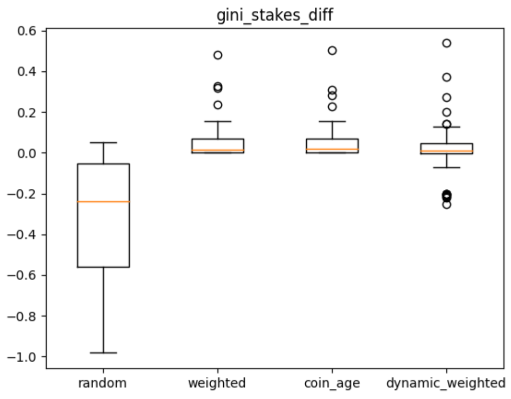
By analyzing these models together, we look to gain insights into their behavior of the variants. To ensure a correct evaluation, we selected and common parameters, with a particular focus on the initial Gini coefficient, that as the provious experiments shown, is a key factor affecting wealth distribution in each model.

The following table summarizes the input parameters used for the experiment, which executes 240 models with 4 parallel executions for a total of 960 executions.

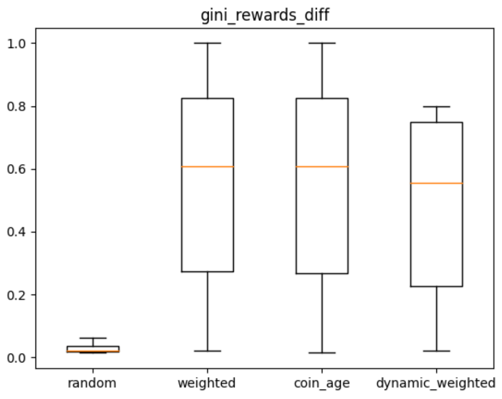
Each simulation performed 4 parallel executions, and for each execution the average of the Gini coefficient used to calculate the metrics g_s and g_r was taken.

Parameter	Value
Parallel executions	4
Number of agents	10
Number of epochs	10k
Initial Stake Volume	1k
Reward type	Constant, Geometric
θ function	$\theta(x) = 0.475x - 0.2325$
Gini threshold	80%
Pos type	random, weighted, coin age, dynamic
Total reward	100, 1k, 10k, 100k, 1M
Initial Gini coefficient	0, 0.2, 0.4, 0.6, 0.8, 1

The comparison of the models, and in particular of the metrics attributed to each model, has been summarized in the boxplots presented in Figures 5.24a and 5.24b.



(a) Gini stakes difference metric by pos type



(b) Gini reward difference metric by pos type

The simulations based on the **Random PoS** model show an average g_s = -0.3304 $\pm$ 0.3188, with a maximum value of 0.046232 and a minimum value of -0.983612. The Random PoS exhibits the lowest average Gini stakes difference g_s value, indicating a tendency towards a more equitable wealth distribution compared to the other models. This suggests that, in terms of stake distribution, Random PoS can be considered the fairest among the models, because it tends to lower the Gini coefficient.

The average metric based on the difference in rewards g_r assumes a value of 0.0279 $\pm$ 0.0114, with a minimum value of 0.0146 and a maximum of 0.055195. The Random PoS model is the one that comes closest to the optimal point for the g_r metric, furthermore it has a very low volatility, compared to the other models.

If the goal is to mitigate the *rich get richer* problem, the Random PoS model seems to be the best choice. With its negative average g_s value, it indicates a tendency towards reducing wealth concentration and provides a more balanced opportunity for participants, potentially addressing the issue of wealth accumulation.

The boxplots describing the **Weighted PoS** models show an average metric g_s = 0.0614 $\pm$ 0.102044, with a maximum value of 0.5653 and a minimum value of 0, and g_r = 0.5664 $\pm$ 0.319786 with a maximum value of 1 and a minimum value of 0.0201.
The Weighted PoS model shows positive average g_s values, implying that nodes with larger stakes are favored, leading to a higher concentration of wealth in the hands of a few. This model appears to have a *rich get richer* problem in terms of stake distribution but it comes very close to the optimal point for the metric g_s, in fact it tends to maintain the same initial Gini coefficient.

The Weighted PoS model, with its positive average g_s values, may contribute to centralization problems. Nodes with larger stakes are more likely to become validators, consolidating control among a smaller group that have less tokens in stake. This centralization could potentially weaken the decentralized nature of the blockchain system.

The **Coin Age PoS** simulations, instead, present an average metric $g_s = 0.054248 \pm 0.095568$, with a maximum value of 0.5453 and a minimum value of $4.0404 * 10^-7$. The average metric takes on the value $g_r = 0.566422 \pm 0.319786$ with a maximum of 1 and a minimum value of 0.0201.

Finally, the **Dynamic PoS** simulations show an average value for the metric $g_s = 0.0176 \pm 0.132995$, with a maximum value of 0.5197 and a minimum value of -0.2223, and an average metric $g_r = 0.4897 \pm 0.2613$ with a maximum value of 0.7978 and minimum value in 0.0178.

Both the Coin Age PoS and Dynamic PoS models exhibit positive average g_s values, suggesting a favoring of nodes with more stake. While not as extreme as the Weighted PoS, they still present a good maintenance in terms of initial wealth distribution, in fact the g_s metric is close to the optimal for both Coin Age and Dynamic Pos models. However, in Dynamic PoS, the g_r metric has a distribution with a wider range of values than in Coin Age PoS, to indicate that a larger number of participants may receive rewards, making the distribution of rewards less concentrated among a few privileged participants.

Chapter 6

Conclusions and Directions for Further Research

The models presented in the results chapter are compared below, showing their differences when the system is in the initial state of maximum concentration ($g = 1$), in the state of equidistribution ($g = 0$), and in the intermediate states of concentration ($g \in (0, 1)$).

In case of **maximum concentration**, it is possible to note how in figures 5.1 and 5.3 the behavior of the **Random PoS** tends to reduce the Gini coefficient towards 0, leading to an opposite state of equidistribution as the number of epochs T increases, both for the metric g_r and for the metric g_s. In this case, the blockchain assigns the validation of the blocks in a pseudo-random way to all the participants. The Weighted PoS and the Coin-Age PoS models obtained, on the contrary, keep the state of maximum concentration unchanged as shown in figure 5.8, producing the opposite effect with respect to the Random PoS, both for the metric g_r and for g_s. In this case the richest individual becomes richer, also introducing security problems in the blockchain, whose validation of the blocks is attributed to a single individual. The Dynamic PoS maintains the state of maximum concentration unchanged if the Gini threshold is set to 1, otherwise it obtains a reduction effect of the Gini coefficient which depends on the θ function and the Gini threshold set, as it can be seen in Figures 5.18a, 5.18b, 5.19a and 5.19b.

In cases where the system is initialized in the state of **equidistribution**, it is possible to notice how the Random PoS model, despite the instability of the first epochs, tends to reduce the metrics g_r and g_s to 0 over the epochs. The weighted and dynamic pos models produce a slight increase in the Gini coefficient which tends to stabilize at a constant value of about 0.5, for both the metrics g_s and g_r. The Dynamic model in this case shows a stabilization at a constant and low value which decreases as the reduction factor increases.

In cases where the system is in intermediate states, i.e., in states of **concentration**, it is possible to notice how the random model can in any case reduce the metrics g_S and g_r towards 0, as the number of epochs T increases. The Weighted models tend to slightly increase the g_r and g_s metrics and keep the Gini coefficient constant as the epochs increase. Conversely, coin age models tend to reduce the Gini coefficient towards 0 only if a reduction factor less than 0 is set. The decrease in the Gini coefficient occurs more slowly than in Random models. The dynamic pos models show similar behaviors to the weighted ones, in case the initial concentration state is set to a value lower than the threshold, otherwise they show a decrease of the Gini coefficient which does not tend to 0, but to a value which depends from the threshold set and from the transformation function θ.

The comparison of the models showed through a further experiment how the metrics g_r and g_s approach the optimum, i.e. $g_r* = 0$ and $g_s* = 0$. In particular, for the metric g_r , the models that are closest to the optimum are the dynamic models, with an average value $g_r = 0.0176 \pm 0.1329$. In this sense, the best models, on average, maintain the values of the Gini coefficient of the initial distribution unchanged.

The metric g_s that came closest to the optimal point was obtained using the random models $g_s = 0.0279 \pm 0.0114$. In this sense, random models are those that achieve the best fairness values for the metric g_r, when the Gini coefficients of the initial distribution of the system are neglected.

In conclusion, it is possible to see how random models are those which, over the course of the ages, make the redistribution of wealth more homogeneous by redistributing wealth equally to all nodes of the network. This produces the effect of reducing the initial Gini coefficient highlighted by the metric g_s. In practice, however, this model is not feasible, as the nodes would have no interest in staking large capital, investing in network security. The dynamic models, on the other hand, proved to be the best ones from the point of view of the g_s metric, guaranteeing a good maintenance of the initial Gini coefficient and at the same time reducing the rich get richer effect.

6.1 Future developments

The current project analyzes the effect of wealth redistribution in a non-distributed context, in which some real effects of a typical blockchain network are neglected. Furthermore, the presented models would have to undergo adaptations to be exploited in a distributed network, so it would be a challenge to implement distributed algorithms that maintain the probabilistic and redistribution effects of the theoretical models just presented.

Further aspects could be considered to add complexity to the project, for example transforming the simulator into a distributed one (BlockSim [2]), in order to simulate ledger ramifications and network latencies. This could open up the possibility of analyzing the effects of any Denial-of-Service (DoS) or Stake Bleeding attacks to a blockchain network (LUNES-Blockchain [28]).

Other improvements could concern improving speeds through a greater amount of parallel simulations, number of agents, and exploiting parallel computation methods with a greater number of computational elements, or by increasing the amount and variety of initial distributions from analyzes with different values of Gini. To make the model more similar to real scenarios, the distributions of wealth and number of popular blockchain nodes (Ethereum [33], [30]) could be taken as input parameters.

Bibliography

[1] *40%+ of Ethereum PoS nodes are controlled by 2 addresses, says Santiment data.* URL: https://cointelegraph.com/news/40-ethereum-pos-nodes-are-controlled-by-two-addresses-says-santiment-data. (accessed: 21.03.2023).

[2] Maher Alharby and Aad van Moorsel. "BlockSim: An Extensible Simulation Tool for Blockchain Systems". In: *Frontiers in Blockchain* 3 (2020). ISSN: 2624-7852. DOI: 10.3389/fbloc.2020.00028. URL: https://www.frontiersin.org/articles/10.3389/fbloc.2020.00028.

[3] Luca Bandelli. *Inequality In Proof-of-Stake Schemes: A Simulation Study.* 2021. URL: https://fse.studenttheses.ub.rug.nl/23978/. (accessed: 25.04.2023).

[4] Imran Bashir. *Mastering blockchain.* Packt Publishing Ltd, 2017.

[5] Eric Brewer. "Towards robust distributed systems". In: July 2000, p. 7. DOI: 10.1145/343477.343502.

[6] Michael Catalano, Tanya Leise, and Thomas Pfaff. "Measuring Resource Inequality: The Gini Coefficient". In: *Numeracy* 2 (July 2009), pp. 1–7. DOI: 10.5038/1936-4660.2.2.4.

[7] *Cloak posa v3.0 - A trustless, anonymous transaction system for cloakcoin.* April 2015. URL: http://cryptochainuni.com/wp-content/uploads/CloakCoin-posa3-whitepaper.pdf. (accessed: 18.10.2023).

[8] Giovanni Cornia and Julius Court. "Inequality, Growth and Poverty in the Era of Liberalization and Globalization". In: (Mar. 2004). DOI: 10.1093/0199271410.001.0001.

[9] Cynthia Dwork and Moni Naor. "Pricing via Processing or Combatting Junk Mail". In: *Advances in Cryptology — CRYPTO' 92.* Ed. by Ernest F. Brickell. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 139–147. ISBN: 978-3-540-48071-6.

[10] *Ethereum's energy usage will soon decrease by 99.95%.* URL: https://blog.ethereum.org/2021/05/18/country-power-no-more. (accessed: 21.03.2023).

[11] Giulia Fanti et al. "Compounding of wealth in proof-of-stake cryptocurrencies". In: *Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23.* Springer. 2019, pp. 42–61.

[12] Joseph L Gastwirth. "The estimation of the Lorenz curve and Gini index". In: *The review of economics and statistics* (1972), pp. 306–316.

[13] Corrado Gini. "Variabilità e mutabilità (1912)". In: *Memorie di metodologica statistica. Rome: Libreria Eredi Virgilio Veschi* 10 (1955).

[14] *How does POS stake concept deal with rich becoming richer issue?* 2017. URL: https://www.reddit.com/r/ethereum/comments/6x0xv8/how_does_pos_stake_concept_deal_with_rich/. (accessed: 30.04.2023).

[15] Akshita Jain et al. "Proof of stake with casper the friendly finality gadget protocol for fair validation consensus in ethereum". In: *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* 3.3 (2018), pp. 291–298.

[16] Virendra Kamalvanshi and Saket Kushwaha. "Economic Inequality: Measures and Causes". In: Jan. 2021, pp. 1–12. ISBN: 978-3-319-71060-0. DOI: 10.1007/978-3-319-71060-0_27-1.

[17] Elie Kapengut and Bruce Mizrach. "An event study of the ethereum transition to proof-of-stake". In: *Commodities* 2.2 (2023), pp. 96–110.

[18] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem". In: *ACM Transactions on Programming Languages and Systems* (July 1982), pp. 382–401. URL: https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/.

[19] Suhyeon Lee and Seungjoo Kim. "Short selling attack: A self-destructive but profitable 51% attack on pos blockchains". In: *Cryptology ePrint Archive* (2020). URL: https://eprint.iacr.org/2020/019.pdf. (accessed: 18.10.2023).

[20] Robert I Lerman and Shlomo Yitzhaki. "A note on the calculation and interpretation of the Gini index". In: *Economics Letters* 15.3-4 (1984), pp. 363–368.

[21] Wenting Li et al. "Securing proof-of-stake blockchain protocols". In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology: ESORICS 2017 International Workshops, DPM 2017 and CBT 2017, Oslo, Norway, September 14-15, 2017, Proceedings*. Springer. 2017, pp. 297–315.

[22] D. Malchiodi. *Indici di concentrazione*. URL: https://dariomalchiodi.gitlab.io/sad-python-book/L05-Indici_di_eterogeneit%C3%A0.html#Indici-di-concentrazione. (accessed: 25.06.2023).

[23] *Mesa*. URL: https://mesa.readthedocs.io/en/stable/overview.html. (accessed: 18.10.2023).

[24] Satoshi Nakamoto. "Bitcoin: A peer-to-peer electronic cash system". In: *Decentralized business review* (2008), p. 21260.

[25] *NetLogo*. URL: https://ccl.northwestern.edu/netlogo/. (accessed: 18.10.2023).

[26] *Novacoin - proof of stake,* URL: https://github.com/novacoin-project/novacoin/wiki/Proof-of-stake. (accessed: 18.10.2023).

[27] Gert Rammeloo. *The economics of the proof of stake consensus algorithm.* Oct 29, 2017. URL: https://medium.com/@gertrammeloo/the-economics-of-the-proof-of-stake-consensus-algorithm-e28adf63e9db. (accessed: 25.04.2023).

[28] Edoardo Rosa, Gabriele D'Angelo, and Stefano Ferretti. "Agent-Based Simulation of Blockchains". In: *Communications in Computer and Information Science*. Springer Singapore, 2019, pp. 115–126. DOI: 10.1007/978-981-15-1078-6_10. URL: https://doi.org/10.1007%5C%2F978-981-15-1078-6_10.

[29] Ashish Rajendra Sai, Jim Buckley, and Andrew Le Gear. "Characterizing Wealth Inequality in Cryptocurrencies". In: *Frontiers in Blockchain* 4 (2021). ISSN: 2624-7852. DOI: 10.3389/fbloc.2021.730122. URL: https://www.frontiersin.org/articles/10.3389/fbloc.2021.730122.

[30] Scott Nadal Sunny King. *PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake*. 2012. URL: https://www.peercoin.net/read/papers/peercoin-paper.pdf. (accessed: 25.04.2023).

[31] *Two Addresses Control Over 45% of Ethereum Validator Nodes Post Merge*. URL: https://bitcoinke.io/2022/09/ethereum-validator-nodes-control-post-merge/. (accessed: 21.03.2023).

[32] Oleksandr Vashchuk and Roman Shuwar. "Pros and cons of consensus algorithm proof of stake. Difference in the network safety in proof of work and proof of stake". In: *Electronics and Information Technologies* 9.9 (2018), pp. 106–112.

[33] Gavin Wood et al. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32.

[34] Neo CK Yiu. "An overview of forks and coordination in blockchain development". In: *arXiv preprint arXiv:2102.10006* (2021).