The Marketplace: A Decentralized Execution Network for Useful Computational work

Abstract: The Marketplace network is a permissionless market where nodes execute useful compute bound tasks in exchange for a digital cryptocurrency Goldcoin(gdc). It functions as a decentralized world computer using a new model called **PESF** (Parallel Execution Synchronous Finalization) as further discussed in this paper to achieve near infinite linear scalability, and competitive execution speeds whilst still guaranteeing trustless security. This model works by separating compute work execution from financial state ordering for which the entire network blocks on to reach consensus. For every new compute work request referred to as a **Job,** an instance of a **Whiteroom** is created to handle the job in a process similar to spawning a new execution thread which does not block the execution of other jobs. Internally, the whiteroom is a committee of nodes secretly chosen to execute a job on behalf of the network, and return the result alongside a cryptographic proof of membership after which the members participation is over. If the network observes consensus on the result of the whiteroom committee, the job result can be used by the **Employer** (user that initiated the job) immediately. But the payment of nodes that participated in the whiteroom is finalized synchronously every **Epoch** (periodic time interval to update ledger) using pure proof of stake (**PPOS**).

1. Introduction

Blockchain technology was initially created to allow for truly non-reversible transactions, and eliminate the need for trusted central authority in financial systems. But since it's inception, blockchain tech has greatly expanded its application into a lot of sectors which includes NFTs, Gaming, Law, AI e.t.c. In addition, blockchain projects like Ethereum included the use of turing complete programming to allow for the arbitrary execution of units of code called smart contracts, this made the blockchain network function like a decentralized world computer. But this world computer could not scale effectively because to reach consensus, every on-chain execution had to be re-executed in sequential blocking order by every node on the network. As computational workloads became more demanding, this model resulted in redundant computation, limited throughput, and increased execution costs reflected in the extreme upward trend of blockchain fee prices during peak usage times.

To solve this problem, the Marketplace introduces a decentralized execution network designed to separate computational work from financial state finalization. Rather than requiring the entire network to fully execute every computational task, the Marketplace assigns each Job to a privately selected committee of nodes called a **Whiteroom**. The selected nodes independently execute the Job and provide cryptographic evidence of their membership, and the resulting computation.

The Marketplace uses **Parallel Execution Synchronous Finalization (PESF)** to allow independent Jobs to execute concurrently while financial state changes are finalized periodically by the network. This separation is intended to provide a scalable environment for compute-bound workloads while preserving decentralized consensus over the financial state of the network.

Note: This paper acts as a strong overview of the Marketplace idea, but technical deep-dives will be explored in the Marketplace Technical Report.

2. Parallel Execution

Parallel Execution in the Marketplace refers to the ability of the network to execute multiple **Job** requests (Compute bound task) in parallel. The network achieves this by decoupling financial event ordering from work execution, and forming small committees called **Whiteroom** for each job to act like a free CPU thread for the network. A job is initiated when a **Work Pointer** Message is sent to the network by an **Employer**.

Work Pointer

A Work Pointer contains information for executing a job on the network. The Work Pointer is a structure that references by address, a compute work stored on a decentralized storage system, where other nodes can access it. But the decentralized storage system must be advanced enough to restrict nodes that do not have authority from accessing the compute work.

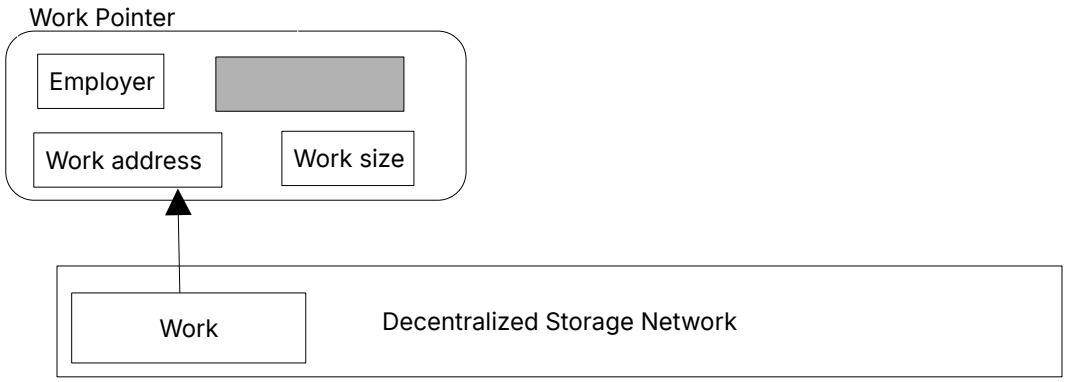

Fig 2.1. Work Pointer

When a node on the network receives a work pointer, it needs to gain the authority to access the job referenced by the Work Pointer, and start executing. To achieve that, the node will attempt a challenge to privately confirm membership in a secret committee called the Whiteroom. Upon membership confirmation, the node may execute the job, and send a **Result Pointer** message to the network containing the computation result, and proof of Whiteroom membership.

Whiteroom

The Whiteroom is a committee of N nodes that are privately selected to execute a job and return its result. This means a node can solve the challenge and confirm its Whiteroom membership without having to interact with the rest of the network.

For a node to be selected into the whiteroom, the challenge it attempts is a **Wesolowski Verifiable Delay Function (VDF)**: A sequential calculation over class groups that requires no trusted setup. To start the VDF, the node uses the hash of the Work Pointer and it's own public key to create a unique VDF seed, and the VDF difficulty is set by the network as described later

in the VRF threshold section. Solving the VDF produces a proof that only takes a tiny fraction of the time it took to compute to validate, and a unique output.

The unique output of the VDF is used as a seed to a Verifiable Random Function (VRF) which generates a random 256 bit number and a proof, if the random number generated is less than the network set VRF threshold, the node is considered part of the whiteroom committee, or else it may try the process again.

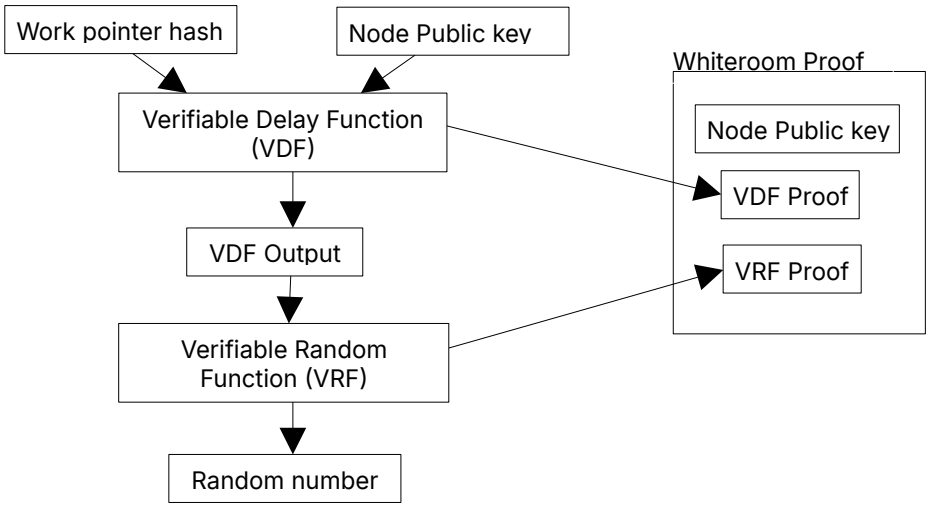

Fig 2.2. Whiteroom Selection Process

Before interacting with the network, the node batches the VDF and VRF proofs as a single structure called the Whiteroom Proof [see Fig 2.2]. If a false identity attempts to hijack a whiteroom proof, the proof would not be valid simply because changing the identity would corrupt the VDF seed, and therefore the output, forcing the false identity to redo everything. Finally, The execution result is sent to the network, alongside its whiteroom proof, which the network can use to efficiently prove the node's whiteroom membership.

For a whiteroom committee to be considered in consensus state by the network, a total of 2f + 1 nodes must reach consensus on the result of the execution according to byzantine quorum agreement, where the total number of nodes is N = 3f + 1, and f is the total amount of disagreeing or unresponsive members. This consensus voting process is as observed by the network because whiteroom members upon sending their result, play no further part in the consensus process.

Upon observed consensus, the execution can be used by the **Employer** instantly. There are also 2f + 1 copies of the result, such that if one result is inaccessible, other results can be used. But the payment of the whiteroom nodes is only confirmed during the Synchronous Finalization phase.

This enables the network to handle many Whiteroom instances in parallel because the network does not need to agree on the order of jobs nor wait on the result of one job changing global state, allowing for job execution to be non-blocking.

An **attacker** may attempt to split the network by sending two jobs spending the same coin. Naturally, each node on the network selects and starts working on the first job it receives rejecting all others as double spend. But assuming that the committee selection produces roughly the same committee size N out of the entire network using VRF threshold calibration, only one of the jobs can have enough whiteroom members to reach consensus.

Even if consensus were reached on both jobs, the synchronous finalization phase would ensure only one job is finalized, which means the whiteroom members that executed the discarded job are not paid. Although the attacker could still use both whiteroom results. Let's investigate the VRF threshold calibration, and how the VRF threshold and VDF difficulty is set by the network.

VRF THRESHOLD

The ideal whiteroom size is given by N, but due to changes in the network size or randomness, the number of selected node may be greater or lesser than N. As discussed earlier, once a whiteroom size has reached a threshold size of:

$$S = 2f + 1$$

where f is the amount of allowable faulty nodes, and N is given by

$$N = 3f + 1$$

The whiteroom can be declared to be in consensus state.

To allow for calibration of the whiteroom based on changes to its size, the maximum number of whiteroom nodes MAX is set to:

$$MAX = 2S - 1$$

where S is the whiteroom threshold size.

When a node receives MAX whiteroom members, it rejects any new members it receives. The MAX formula ensures that threshold size S remains an effective consensus threshold representing at least more than half of whiteroom nodes.

We can take the observed average whiteroom size of the network during a previous epoch t - 1, and then attempt to calibrate the whiteroom to a size of N. To do that, we change the probability of whiteroom selection Pr(t), using the previous Whiteroom Probability Pr(t - 1) as given by:

$$Pr_t = \left(\frac{N}{\text{whiteroom size}} \right) * \left(Pr_{t-1} \right)$$

where t is the current epoch, and t – 1 is the last epoch.

where "whiteroom size" is the average whiteroom size during an epoch, which must fall within the inclusive range of S to MAX.

NOTE: This process is done in the Synchronous Finalization Phase.

Using the probability formula, we can calculate the new VRF threshold as follows:

$$\text{VRF\_T} = \left(Pr_t \right) * \left(2^{256} - 1 \right)$$

where the max VRF_T value is the maximum 256 bit number.

We also define an **Epoch** as a time frame in which multiple active jobs are considered to be running in parallel.

Let's examine the process of randomly selecting a whiteroom committee size N from the entire network size of M nodes.

Consider a network size of M = 1 with a genesis probability Pr(0) = 1, for the network to fill the whiteroom size of N, the single network node has to attempt the VDF challenge N times.

Now consider a network size of M = 2 with a genesis probability Pr(0) = 1, if the two nodes on the network attempt the VDF N times, the two network nodes would both produce 2N result pointers which is more than the MAX whiteroom size for a job. The network gradually corrects itself every epoch using the probability formula until an ideal Pr(t) = (1 /2) or equivalently 0.5 is gotten.

Now consider a network size of M = 2 with Pr(t) = 0.5, if the two nodes attempt the VDF N times with the selection probability of each attempt being 0.5, an average whiteroom size of N instead of 2N nodes is expected.

If an **attacker** attempts to divide this network by sending two double spend jobs to both nodes, each node would work on the job it received first and abandon the other. The probability of any of the two jobs reaching whiteroom consensus is low, because each node only attempts the VDF a maximum of N times with every attempt only having a probability of 0.5. The network is double-spend resistant.

As the network size M tends to infinity for an evenly split network, the whiteroom size of both jobs will be (N / 2) according to the law of large numbers which means only one job may reach consensus. In this case, the whiteroom probability Pr(t) is given by (1 / M) where M is the total amount of network nodes.

How much does an attacker need to invest to control a single Whiteroom? Ideally, the answer to this question should be greater than half the networks computational power. Let's work towards this ideal.

We cannot know the computational power of the entire network, but we can observe how much work the network does every epoch. We call this value the **Observed computational power of the network.** We do this by taking the continual average of the sum of job work size done by the network every epoch. We define this average of total work done W(t) for an epoch t to be:

$$W_t = \frac{W + W_{t-1}}{2}$$

Where W is the total jobs done or more specifically, total jobs finalized by the network during that epoch.

Given M nodes on the network, an attacker attempting to fill a whiteroom should need to invest the following to guarantee a single seat in a whiteroom:

$$\text{Invested power} = \left(\frac{W_t}{N} \right)$$

For this to be true, every participating node on the network has to invest the following:

$$\text{VDF difficulty} = \left(Pr_t \right) * \left(\frac{W_t}{N} \right)$$

This formula ensures that an attacker needs to control majority of the network's resources to control a single whiteroom, while the cost of sybil resistance remains trivial for honest nodes. A financially driven attacker gains more by acting honestly, and spreading their computational power across many jobs.

Result Pointer

When a node confirms it's whiteroom membership, it executes the Job, and stores the execution result on the decentralized storage network. It then batches the address of the result, its proof of whiteroom membership, and other metadata into a structure called a **Result Pointer**.

The Result Pointer is just like the Work Pointer in that it contains the address of the result which points to its location in the decentralized storage, so that the execution owner can access it at any time. The decentralized storage should ideally only allow the employer to access the result, by locking with the employer's identity for privacy reasons.

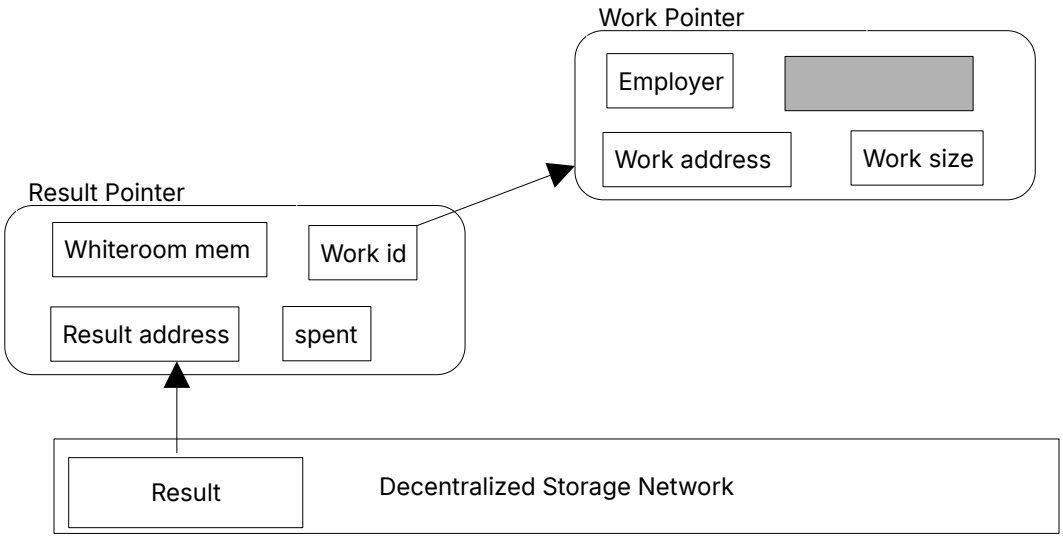

Fig 2.3. Result Pointer

The Result Pointer also acts as a blueprint that any node on the network can use to generate the same transaction outputs sending the job pay to the Whiteroom node that owns the Result Pointer. This is how Whiteroom nodes gets paid.

The job employer only inputs the price of the job as the work pay, but at least 2f + 1 Whiteroom nodes executed the work and are expecting the full job payment. This means there is a difference of (2f * (work price)) total to be paid to the rest of the Whiteroom members. The network solves this problem by multi-spending the job price to pay the total of (2f * (work price)) owed to the remaining whiteroom nodes. This is the only way new native currency **Goldcoin** is created on the network.

Contract

The Contract is the smallest representation of a financial agreement that can be stored on the Marketplace ledger. There are two types of contracts, these are the Job Contract, and the Transaction Contract.

Job Contract

This is a contract that contains all the information concerning a Job which includes what work was done, who did it, who was paid, and how much. In practical terms, it contains a Work Pointer, and the corresponding Whiteroom committee of Result pointers.

Job Contracts are valid only when the Whiteroom of Result Pointers is in a consensus state. Contracts with a Whiteroom that could not reach consensus is automatically abandoned by the network with no financial loss to the Employer that created the job.

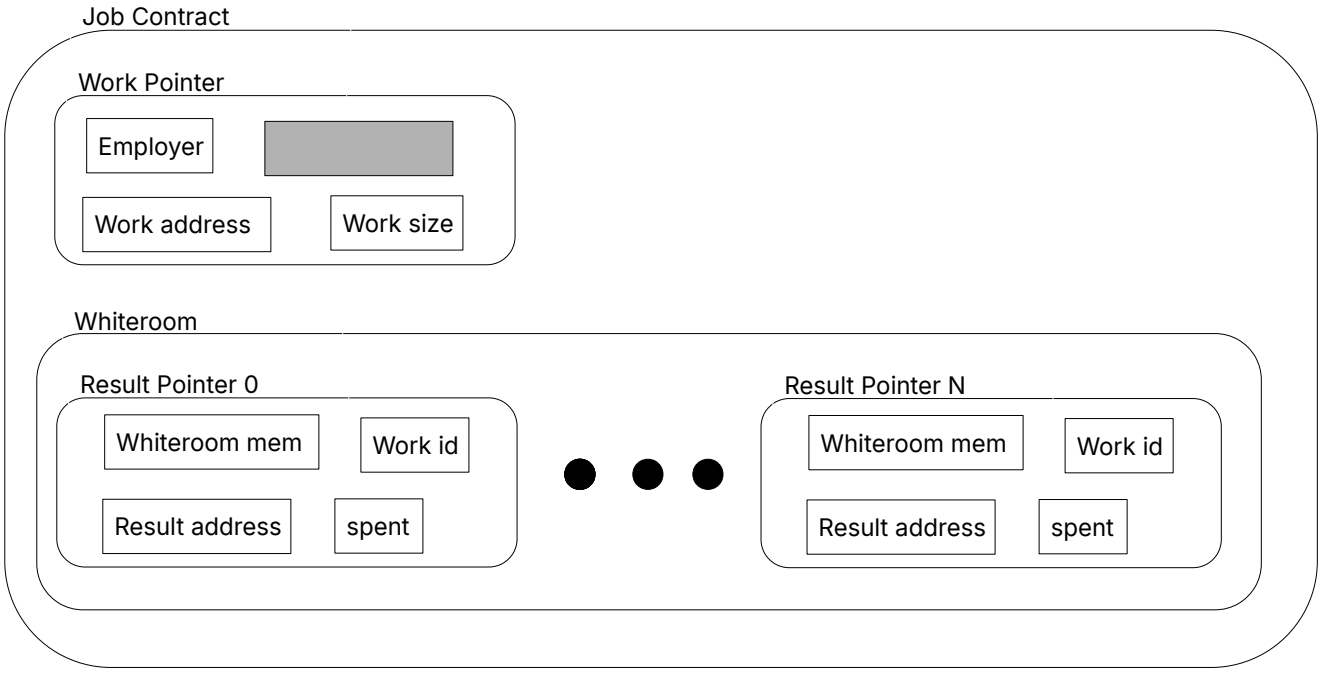

Fig 2.4. Job Contract

Transaction Contract

This is a contract that contains a list of transactions called a Bill. A transaction is simply a structure that records the change of ownership of assets. There are two kinds of assets in the Marketplace which are Goldcoin(gdc) and Account Tokens.

Goldcoin(gdc) is the native cryptocurrency used by the network. It is the only currency that can be used by an employer to initiate a job, and it is only printed to reward Whiteroom members for the amount of useful work done. This currency is backed by useful computational work done on the network even though its supply is inflationary, and every other asset on the network can be backed by it [Account Tokens]. The measurement for gdc is given by:

$$1\,gdc = 3 * 10^{12}\,Operations$$

Where an Operation is a single ADD instruction.

Account Tokens are a type of account that owns other assets and has multiple owners. An account token is instantiated when a transaction with no inputs, and some initial output is hashed and stored on the Marketplace ledger. The transaction hash can be used as an address/account for the Token, where value can be injected into it by sending other assets including gdc directly to its address.

The assets sent to the address of a token can never be withdrawn, so they are effectively destroyed. But the value of the destroyed assets in gdc is added to the worth of the token it was sent to. The supply of a token is fixed at initialization, and cannot grow like gdc. But gdc can be sent to the token to increase it's value.

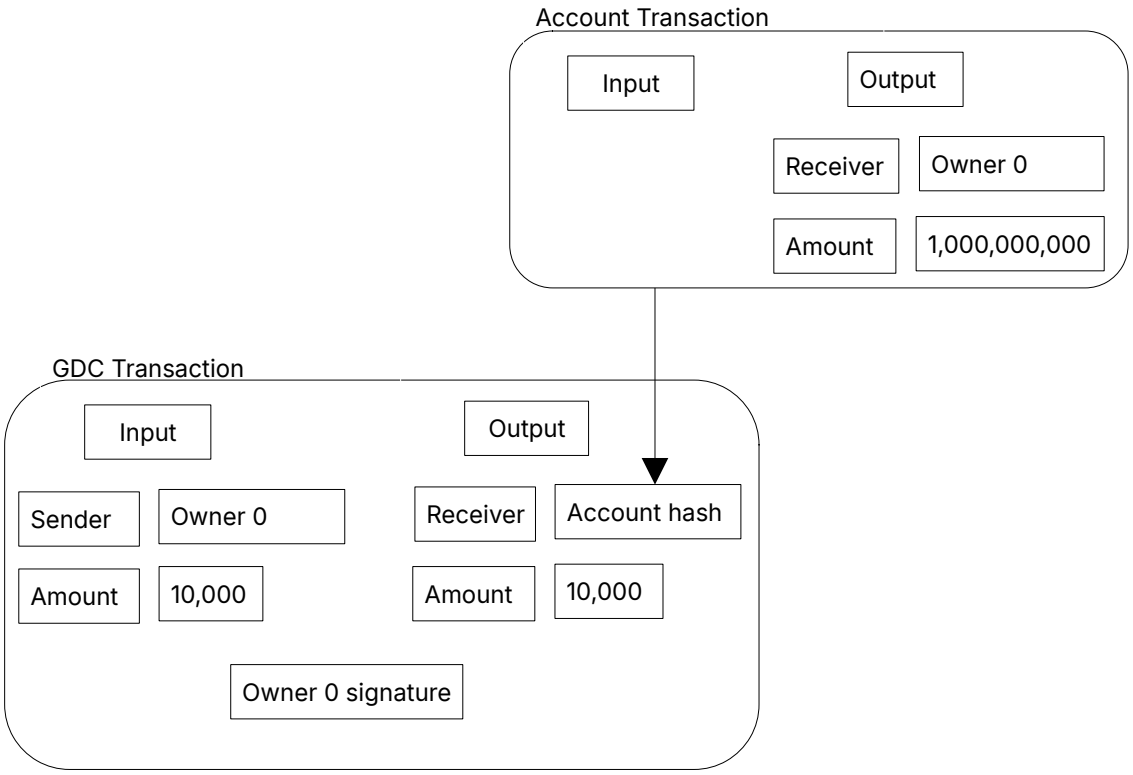

Fig 2.5. A user injecting liquidity into a newly created Account Token

3. Synchronous Finalization

In this phase, the network simply reaches consensus on the contracts they have observed. As discussed earlier, payment of whiteroom members is not finalized immediately after job execution consensus, let's examine why.

Consider this scenario, A whiteroom committee of 25 members needs a threshold of 17 members to be considered valid. When a single node on the network receives 17 result pointers, it concludes the whiteroom and finalize the contract without concerning itself with member identity because only reaching consensus matter for the execution result. In that case, it is trivial for a result pointer to end up in one nodes observed whiteroom, and not be included in another nodes observed whiteroom. If the network does not reach consensus on whether the result pointer is a part of the whiteroom or not, both nodes would have a differing ledger which would split the network.

This is why synchronous finalization exist. This phase is necessarily sequential in nature, and because the entire network blocks on ledger consensus, we split it from the execution phase that needs to be parallelized.

Ledger

The marketplace ledger is a chain of blocks containing a list of contracts stored as a merkle tree, and the previous block hash. Every epoch, a new block is added to the ledger which contains all the contracts finalized during that epoch.

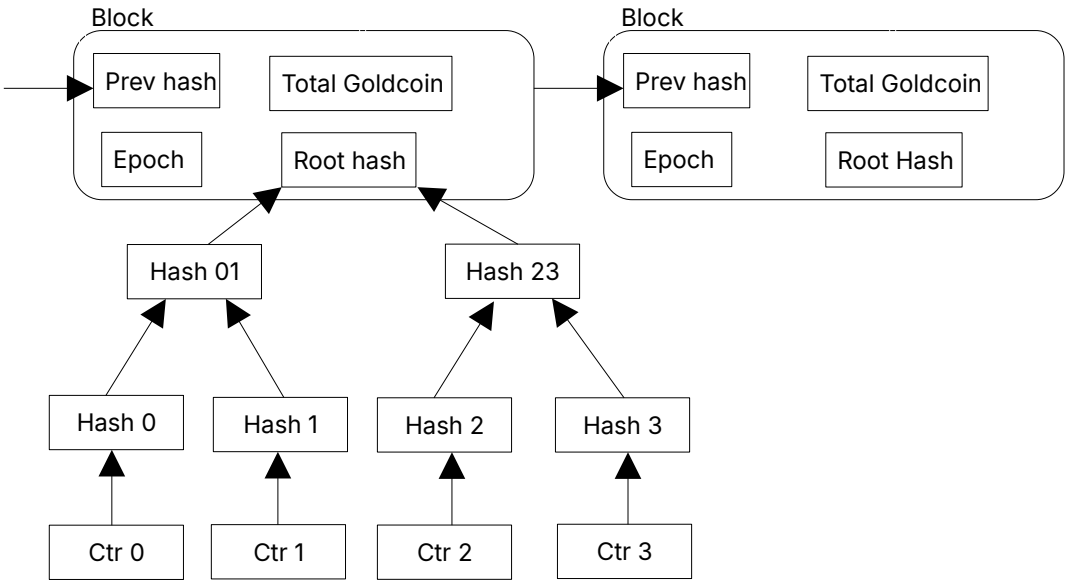

Fig 3.1. Block Chain with merkle tree of Contracts.

Unlike Proof-of-Work blockchains where changing one block changes it's hash and breaks the chain of blocks, forcing the attacker to redo the POW for every other block on the chain, an **attacker** may attempt to go back M blocks in the marketplace blockchain to change a contract in one block. To repair the broken chain, an attacker can trivially add all other blocks into it's false chain, and propose it as an alternative ledger splitting the network.

To solve this problem, the hash of the latest block is referenced by the work pointers created during the current epoch, which then becomes the seed of those job's whiteroom. If a previous blocks content is changed, every job after that block becomes corrupted invalidating every gdc created after that block.

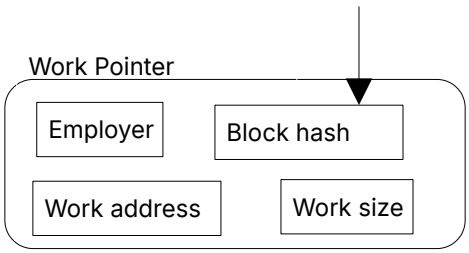

Fig 3.2. Complete Work Pointer

When the network receives multiple conflicting blockchain ledgers, the network can safely select the ledger that has the most amount of gdc, given that gdc is the measure of useful computational work invested into the chain. This is how the network resolves ledger partitions.

Pure proof-of-stake (PPOS)

To reach consensus on the state of the ledger every epoch. The network uses a two-phase pure proof of stake method similar to Algorand. The two phases of this consensus mechanism include **Block Proposal** and **Block Agreement.** By utilizing Verifiable Random functions (VRF) and cryptographic sortition, each node can privately confirm selection into any of these two phases, and the likelihood of selection is directly proportional to their stake in gdc.

Block Proposal

In this phase, every participating node creates a block from all the valid contract it witnessed during an epoch, and attempts to use its stake in gdc to privately confirm if it has been selected to send the newly created block to the network. If selected, the node proposes the new block to the network.

The network receives the blocks proposed by the Block Proposal nodes, and their proof of selection, the network then selects the block with the highest total gdc they received, and reject the others. If more than one block has the same gdc amount, the block with the lowest block hash is chosen. This creates the conditions for the network to enter the Block Agreement phase.

Block Agreement

In this phase, the network having observed the block with the highest gdc they received, vote on it as the final block. They attempt the same cryptographic sortition process as the earlier phase and if chosen, they vote on the block they received. The block with the majority vote wins and is added to the ledger. If no block reaches a majority, the cryptographic sortition is run again with the blocks from the last rounds, until consensus on the block with the most gdc is reached.

For an **attacker** to control this process, the entity would have to posses a majority of gdc on the entire network, but because gdc represents real computational value, the attack cost would be very great relative to the attackers gain.

Use Cases

Let examine some real world use cases of the marketplace protocol.

1) The marketplace network can be used to execute complex financial transactions, but the executed result never changes the ledger state immediately simply because execution is isolated from the ledger as discussed earlier. But the result can be used as a bill that the entity can later send to the network to finalize the transactions.

This enables stress free atomicity, such that if any error occurs executing a part of the transactions, the whole bill can be discarded without ever touching the ledger.

2) The marketplace acts like a proof of computation, where an entity can prove to another that a result it possesses was computed correctly from a specific task at a specific time.

The perfect example of this is a Stall.

A **Stall** is a piece of code that runs pre-set actions depending on it's input which is similar to a smart contract in Ethereum. But unlike smart contracts, a stall is lazy and resides off-chain.

For example, a stall representing a financial service may be executed as follows:

a) The user sends a Work pointer to the network pointing to the stall code off-chain (decentralized storage), but the inputs to the stall is private and can only be discovered by Whiteroom members.

b) The network assembles a Whiteroom by VDF + VRF selection, and the selected members access the inputs, execute the stall, and produce an output.

c) The output is stored off-chain and made private so that only the user can view it, but the result hash is sent to the network by the whiteroom, and the result is finalized

d) Once the user locally observes consensus, it can then view the result.

The result of the execution may just be a single transactions saying:

"Stall S sends 20 Token A to U",

where Stall S is the hash of the execution code, and U is user.

To finalize this transaction, the user can at any time send it to the network as a bill, and a reference to the Job contract that produced the transaction is used as a proof of computation so that the network can easily verify the bill was a result of executing Stall S.

With Stalls, we have replicated smart contract functionality, but with lazy, off-chain and privacy advantages.

Now an **attacker** may try to cheat by spending a bill with an old rate long after it was executed, and may try this many times. But this is easily solved by the Stall adding a time limit to the resulting transaction after which the bill should be considered invalid.

3) The marketplace allows for compute bound tasks to be handled asynchronously. For example, consider a thread executing a process containing a series of compute bound tasks that are independent of each other. Although these tasks should be executed in parallel, a single CPU core is forced to execute them sequentially or at least concurrently. If a user intends to use each result the moment its available, concurrency would be disastrous as the CPU would have to complete almost every task before even one is available.

With the marketplace, each compute bound task can be offloaded to the network to execute, which allows the thread to continue working while asynchronously waiting for the result of each execution. Once the network has executed a task, the result is available immediately upon consensus, and the thread can then handle the result.

Gold Trust Token (GTT)

The GTT is the native token of the network which has a fixed total supply of 100,000,000 GTT representing the total amount of fees collected on the network, and used to invest in the marketplace community. The fee amounts to 0.1% of work pay collected from every whiteroom member on a job. Fees do not exist for normal transactions or bills on the network.

Conclusion

We have proposed a market system where nodes execute compute bound tasks and earn digital money for it. In the protocol specified as Parallel Execution Synchronous Finalization (PESF), work execution is separated from financial state changes to allow the network to perform multiple compute bound tasks in parallel, whilst still reaching consensus on the sequential order of state change on the ledger. An attacker trying to control an execution result or corrupt the ledger would have to own a majority computational power on the network or it's financial equivalent in gdc. We have also proposed a new token system where tokens can own other tokens using an account, and its worth is measured by the native cryptocurrency Goldcoin. Thank you for reading, God bless.

References

[1] Nakamoto, S. (2008). *Bitcoin: A peer-to-peer electronic cash system*. Bitcoin.org.

[2] Buterin, V. (2014). *Ethereum whitepaper: A next-generation smart contract and decentralized application platform*. Ethereum.org.

[3] Chen, J., & Micali, S. (2017). Algorand: A secure and efficient distributed ledger.

[4] Wesolowski, B. (2020). Efficient verifiable delay functions.

[5] Micali, Silvio, Michael Rabin, and Salil Vadhan. (1999). Verifiable random functions.