Incentive-based ledger protocols for solving machine learning tasks and optimization problems via competitions

David Amar
ddam.am@gmail.com

Lior Zilpa
AtoZLabs
Tel Aviv, Israel
lior.zilpa@atozlabs.ai

Abstract

We propose incentive-based protocols that use competitions and public ledgers to solve optimization problems. We introduce Proof-of-Accumulated-Work (PoAW): miners compete in costumer-submitted jobs and accumulate recorded work on which they are later remunerated. These new competitions replace the standard hash puzzle-based competitions. A competition is managed by a dynamically-created small masternode network (dTMN) of invested miners, which improves scalability as we do not need the entire network to manage the competition. Using a careful design of incentives, our system preserves security, avoids attacks, and offers new markets to the miners. Finally, we illustrate how the new protocols can be used for implementing machine learning competitions.

1. Introduction

A *persistent decentralized* ledger is based on an underlying data structure that keeps the transactions in an ordered collection of data blocks. The data structure is immutable in that past transactions cannot be modified, which can be used to record digital truth [1]. Specifically, in a blockchain data are kept in a linked list of blocks that can be extended only by adding a new single block as the head of the chain [2]. Decentralization in this context means that there is no single entity that keeps the information and to which all trust is given. Alternatively, there is a network of peers, also called miners, such that each one holds a copy of the data. This is accompanied with a protocol that lists rules on how a new block is added.

Data are accessible to all miners and other users of the network. They can get multiple copies of the ledger and use the protocol to decide which one is correct. The accepted majority version is called the consensus. Blocks are added in a controlled rate to guarantee that there is enough time for the network to reach a consensus when the data structure changes. Miners both keep a copy of the blockchain and can compete for the right to add new blocks. For the network to consider a new block, miners must prove their credibility. For example, in public blockchains that are used to implement currencies, miners have to prove that a certain resource was spent in the process [2, 3]. The proof system makes it expensive to achieve exclusive ownership on block addition.

Proof-of-Work (PoW) are proofs that guarantee that a certain amount of computing work was performed [4]. This can be implemented using hash-based computational puzzles, see the **Appendix** and [5] for details. PoW protocols are designed such that the expected number of blocks that miners sign is proportional to their relative weight compared to the total computing power of the network. As a result, only a miner whose total computing power is larger than 50% of the combined power of the network can obtain (partial) control over it. An alternative to PoW is called virtual mining [6, 7, 8, 9]. It is based on the idea that money itself can be treated as a form of PoW. In other words, the act of acquiring coins proves that peers had invested from their own resources to become a part of the system. Therefore, we can use money, either owned or deposited, as the underlying proof system of the blockchain. Such systems are often called Proof-of-Stake (PoS) or Proof-of-Deposit (PoD).

In this work we propose methods that utilize the resources of ledgers for solving optimization problems. By formulating such user-submitted tasks as competitions we both utilize the miners' resources and achieve a community-based solution. On the other hand, we show that in the case that the blockchain mints coins, miners benefit both from what the users offer and from the blockchain itself, all by preserving security using an incentive-based mechanism. Specifically, our system is designed to address the following attacks:

Definition 1.1 An $O(1)$ attack of a protocol P is enabled if P either allows or enables a state with a positive incentive for an adversary to: (1) post problems for the optimal answer is known, and (2) immediately propose solutions to these problems.

Definition 1.2 A suboptimal solution adversary (SSA) is a miner that knowingly posts suboptimal or intermediary solutions for an optimization task M. The protocol P is SSA-enabling if there is no negative result for posting such solutions.

2. Related work

Hash puzzles require vast computation resources. Previous works have proposed ways to either replace or reduce the dependence on hash functions. For example, Oliver et al., 2017 [10] proposed solving NP complete problems (NPC) as an additional proof system. Note, however, that these problems do not fully replicate the properties of hash puzzles (e.g., they are only hard in a worst-case analysis) and while NPC solutions can be validated to be true one cannot guarantee that a negative proposition (e.g., there are no cliques of size k in the graph G) unless NP=coNP. In addition the $O(1)$ and SSA issues described above are not fully addressed and the system remains sensitive in that an adversary can take over many blocks without performing computations. Another challenge of the approach above is that difficulty of instances of NPC problems is hard to determine in advance and dynamically while solving a problem (even though rough bounds can be computed). Several previous papers have presented ways to preserve hash properties by substantially limiting the set of practical problems that can be solved. Ball and colleagues [11] presented a proof system based on solving the orthogonal vectors problem, whereas [12] proposed primecoin: a system based on looking for Cunningham numbers.

3. The proposed protocol

3.1. Outline

We start by introducing the ideas of Proof-of-Accumulated-Work (PoAW): a system that accumulates past useful work in a way that is secured thanks to protocol-enforced constraints and rewards. Second, we introduce dynamic Task Masternode Networks (dTMNs): dynamically created subnetworks of invested miners that manage user-submitted tasks.

To achieve our goals we also utilize: (1) virtual mining, (2) replacing competitions for solving hashes with competitions over computational tasks, and (3) storage blockchains. Virtual mining acts as the glue between the different concepts. Public competitions are the places in which users submit tasks and miners can propose solutions. The storage chain keeps the information about the computational tasks. Miners of the distributed storage can constitute the dTMN, which also validates suggested solutions.

We also avoid the need to precisely evaluate the difficulty of the users' computational tasks. Alternatively, we let users offer the tasks to the system and their payment. Thus, we let the market regulate the value of each optimization problem that is submitted. The resulting protocol achieves three important goals. First, the network keeps the cryptographic power of the blockchain components intact. Second, most of the computational power of the system can be utilized to solve optimization problems. Finally, miners earn from both serving the system and solving computational tasks. Moreover, we prove that joining our network as a computational node is more profitable than either renting out hardware in a contract-based system or standard PoW mining.

3.2. The virtual mining system

3.2.1 Virtual stakes

We use virtual mining to obtain two goals: approve new blocks, and as a device for implementing PoAW. Miners that solve computational tasks get *virtual stakes* (vstakes) after a competition is sealed. Thus, there is a special transaction that creates a certain amount of locked virtual money that the miner can use as stakes only. These are not coins that the user can send or spend: these are rather specialized tokens that the miner accumulates from the system in return for solving tasks. *Vstakes* provide a way to later remunerate miners for their work, which already earned from solving fees. We propose using the PoS system of Decred [8], but other PoS solutions can be used.

3.2.2 Decred augmentation

Decred [8], which shares conceptual similarities to both DASH [7] and Peercoin [6], is based on a lottery system in which tickets are purchased and then deposited for a wait period before they can be spent. Selection of tickets from a pool is done using a pseudorandom algorithm that is based on information in the head of the new block. This allows (pseudo) random selection of tickets from the ticket pool. Each ticket gives the PoS miner a single vote, but ticket prices are determined by the protocol automatically such that the total number of tickets is maintained at a desired limit. This process is the Decred PoS equivalent of a standard hash difficulty determination algorithm. For a new block to be accepted into the blockchain five votes must approve it. The PoS miners receive both their tickets worth and additional payment. Note that there is an approximately 5% chance that a ticket will end up deleted without producing additional reward.

A tickets lifetime goes through the following stages. It is created when a miner pays the *ticket price* + *ticket fee*. A block can create up to 20 new tickets. Tickets are mined into a block such that those with higher fees get preference. Once accepted into the blockchain, a ticket must wait 256 blocks before it enters into the ticket pool. The ticket pool size is kept at 40960. If a ticket is not selected after 142 days it expires and the miner is paid back the ticket price. After a ticket is consumed, it enters into another 256 blocks period of wait after which its associated funds are released.

Voting is done by transforming tickets into vote transactions. A vote represents the ticket validation of the last seen block. The new block must contain a majority of the votes to be added into the blockchain (which implies that votes can be missed). Only when the previous block is validated by the votes in the new block the blocks funds in the UTXO set (set of remnants funds and rewards to the miner) are allowed. The process presents the validation result of the last block in a single bit that summarizes the result. PoS miners must remain constantly active as they may otherwise miss their voting turn.

Decred is similar in spirit to DASH in that they both establish a masternode network that takes over a desired task. In both, becoming a member of this network costs (either in tickets in Decred, or by depositing 1000 coins in DASH). Note that DASH is not a PoS system in that there is a single deposit and amounts are not staked. Nevertheless, the idea of substantial initial deposit can be easily extended into the Decred protocol. Also, note that the main goal of Decred is to strengthen decentralization and open governance while promoting community input. These needs stem from past weaknesses observed in Botcoin and past criticism of the PoS idea.

In our augmentation of Decred vstakes are given to the winners of a computational competition at the last transaction, called $Seal_{CT}$. Let fee_{solve} be the amount that the client wishes to pay for a task CT (a formal definition of a task and its associated transactions is given below). We define a protocol parameter called P_{vstake}. The $Seal_{CT}$ transaction contains a special minting command that gives the following amount of vstakes to each winner:

$$\frac{(P_{vstake} * fee_{solve})}{N_W}$$

where N_W is the number of miners that won the competition.

3.2.3 The PoS higher profit invariant

We define an additionl constraint that is essential for preventing $O(1)$ attacks. Let x be the amount of virtual stakes that are required to purchase a ticket. Our *PoS higher profit invariant* (simply called PoS invariant) determines that x cannot yield extra profit from the system.

Formally, let $y > x$ be the price of a ticket and let $r > 1$ is the expected profit factor of the PoS mining. For example, in a system that is designed to produce an expected 10% profit on PoS spent money $r = 1.1$. In our system only $y - x$ is entitled to receive r where x is entitled to exactly 1. That is, the PoS miner will receive from the system an expected reward of $x + r(y - x)$. Another way to interpret this constraint is that PoW miners are not asked by the protocol to share their profits with the PoS rights that result from virtual stakes.

In our protocol each new block also mints new coins. Note that these new coins are not entitled to virtual stakes. Thus, PoW miners are paid once for their work. However, they do not get all minted coins as in Decred, some of these sums go to the PoS miners as a reward.

3.3. Computational competitions

3.3.1 Preliminaries and notations

Tasks. A computational task $CT =< Type, F_D, D, C, S >$, is a five-tuple where $Type$ is a bit specifying if this is a maximization or a minimization problem, F_D is the scoring function we wish to optimize that depends on a dataset D, which can be null if F_D is self-contained. C is a set of constraints, and S is a set of additional search space parameters on top of the ones implied by the definition of F_D. For each computational task CT we define its slim version $CT^s =< Type, F_D, H(D), C, S >$ which is exactly as CT except for the data element: we keep the hash result of the data instead of D itself. Thus, CT^s requires much less space as compared to CT.

Transactions. We define a set of competition-related transactions. $Publish_{CT}$ is a bid offer by the client to the storage. $Stored_{CT}$ is a transaction that marks the acceptance of CT and its associated dataset into the storage. It also marks the acceptance of CT into the system as a new competition. $Solve_{CT}$ is a transaction that contains an offered solution. $Validate_{CT}$ is a transaction in which the *dynamic Task Masternode Network (dTMN)*, which is a group of storage peers that govern data storage and validation, is asked to validate and choose the winner, which is finally marked using a transaction called $Seal_{CT}$.

As common in blockchains, the transactions above may offer some fees to promote their execution or addition to the ledger. Generally, our transactions have two mechanisms by which service fees are paid. First, we have regular fees that are sums that party A sends directly to party B. These are generally denoted as Fee_{tr}. Second, we have *promised fees (PFs)*, which are sums paid later than the block if a certain criteria was met.

Formally, a promised fee $PF =< o, p, b >$ is a contract stating that if a payment o is carried then a certain fraction $p \in [0, 1]$ is sent after at least b blocks. Formally, all competition related transactions are tuples and each transaction has a unique identifier which can be denoted as $ID(Tr)$ of a transaction Tr (of any type). We define several transactions below. For ease of notation we omit the transaction ID and its Fee_{tr}.

- The first transaction is called $Publish_{CT}$ and it gives all the needed information from the client. $Publish_{CT} =< CT^s, Fee_{sub}, Fee_{solve}, PF_{solve}, PubID_c >$, where CT^s is the slim version of CT, $Fee_{sub} > 0$ is a storage submission fee. Fee_{solve} is the total offered solving fee, PF_{solve} is a a data structure that details the promised fees taken from Fee_{solve} as reward to the infrastructure maintaining miners. Note that PF_{solve} specifies amounts that are paid only if the competition successfully outputs a result.This issue is further discussed in **Section 3**.

- $Stored_{CT} =< CT^s, ID_{publish}, PubID_c, PubID_{masternode} >$, where CT^s is the slim version of CT, $ID_{publish}$ is the ID of the $Publish_{CT}$ transaction previously submitted by the client. $PubID_{masternode}$ is a set of storage miners (i.e., their public keys) that are committed to store the data of CT, such that $|PubID_{masternode}| > r_s$, where r_s is a protocol parameter added to ensure that there are enough copies of CT in storage.

- $Solve_{CT} =< PubID_{miner}, Sig(Sol), Score, PubID_{CT}, ID_{storage} >$, where $PubID_{miner}$ is the public key of the submitting miner, $Sig(Sol)$ is a digital signature of the submitted solution that acts as a commitment of the miner that his real solution will have the score $Score$. $PubID_{CT}$ is the identifier of the associated $Publish_{CT}$ transaction. $ID_{storage}$ is the public id of the storage miner from which data were retrieved.

- $Validate_{CT} =< E_{dTMN} >$ is a transaction in which each member of the dTMN of CT adds an encrypted list of past $Solve_{CT}$ transactions that were successfully validated.

- $Seal_{CT} =< D_{dTMN} >$ contains the decryption specification for interpreting the $Validate_{CT}$ transaction. By knowing this information the winner of the competition is declared.

Competition parameters. Let $BC = B_1, B_2,$ be the ordered set of blocks in the blockchain. Denote $BC_{(i,j)} = B_i, , B_j$ to be a subset of BC starting at block i and ending in block j. A computational competition CC_{CT} is associated with a single computational task CT and has five phases: (0) store, (1) freeze, (2) compete, (3) validate, and (4) seal. Each phase has a defined number of blocks that mark its duration. We denote this number as NB_x, where x is one of the competition phases. Let $I(Tr)$ be the block index of a transaction Tr in the blockchain BC.

3.3.2 Competition flow

A computational competition acts as a microenvironment in the system and is illustrated in **Figure 1**. A competition over a task CT starts when $Stored_{CT}$ is added to the blockchain. This is defined as a part of the protocol and every user or a miner that observes the addition of $Stored_{CT}$ to the block $B_j = I(Stored_{CT})$ can now compute the duration of each phase of the competition.

Addition of $Stored_{CT}$ to the blockchain requires a pre-competition process that establishes replicates of data in storage, see the **Appendix**. Briefly, this process starts when the client submits D and its accompanying information to the storage chain in a transaction called $Publish_{CT}$. After a certain amount of time D has a required set of copies in the storage chain, one copy per miner. This set of miners becomes the dTMN: a subnetwork of peers that are tasked with keeping D and later validating proposed solutions. Throughout the competition, the peers continuously send out ping messages notifying the network that they are active. After validation has ended, which is through a consensus reached by the dTMN, they share a certain percentage of the fees that CT entails.

The next phase is freeze: from block $I(Publish_{CT})$ to block $I(Publish_{CT}) + NB_{freeze}$ no $Solve_{CT}$ transaction is allowed into the blockchain. This step is one of our tools against an $O(1)$ attack. In this case, the attacker has no guarantee that her predefined solutions will be immediately profitable. Moreover, other miners will now have time to solve the published task. Next, the compete phase is the main place in which the blockchain stores proposed solutions for CT. For ensuring security of their solutions the solving miners do not send out their solution. Instead they present a digitally signed commitment. The reason for this commitment is to avoid a case in which the block signing miners will present the observed solutions as their own.

An important point in our protocol is that all PF fees are accumulated across sealed competitions, each in its designated pool, and are distributed every B_{distr} blocks. We maintain three profit pools that are designed to reward the infrastructure components of the competitions: (1) the main chain (i.e., the public ledger), (2) the storage, and (3) the storage-main chain interaction. Note that the last pool is a derivative of pool (2): it is a defined percentage P_{SMPool} of sums that the storage pool pays the PoW miners for accepting the storage infrastructure transactions into the ledger (e.g., bid, ask, or deal).

For every pool, funds are distributed in a way that is proportional to the amount of work that was done by the miners, which can also be weighted by the transaction importance. For the main chain this is kept in expected value (i.e., on average over a stochastic process), whereas the other pools give out rewards in a deterministic fashion that is directly proportional to the amount of work done. In the Appendix, we present the algorithm of the main chain pool, which is based on enumerating the $Solve_{CT}$ transactions from the winning miners. The algorithms of the other pools are very similar and can be easily derived from it. Briefly, let PF be the total sums in the main chain pool from the promised fees of the sealed competitions during the block range $B_i, , B_j$ such that $j - i + 1 = B_{dist}$. We assign a weight w_k for each block B_k such that: $\sum_{k=i}^{j} w_k = 1$. The weight w_k is set to be proportional to the number of $Solve_{CT}$ transactions in B_k that won their competition. The weight allocation algorithm is given below. For a block B_k with a weight w_k let $S_1, , S_u$ be the set of $Store_{CT}$ transactions in B_k that won their competition and let $M(B_k)$ be the miner that signed B_k. Given the output of the algorithm below, $w_k\ PF$ is given to $M(B_k)$.

The next step of a competition is validation. Validation entirely depends on the dTMN, whose members wish to perform work, reach consensus, and seal competitions to gain more profits as discussed above. The storage miners here receive means to go back and validate the solution commitments submitted during the competition. The validate transaction holds for each dTMN member an encrypted list of $Slove_{CT}$ transaction ids that were successfully validated (for example, we can use hash functions). Upon publication of a validation transaction the dTMN publishes the last transaction that seals the competition: $Seal_{CT}$. Here, the dTMN members present the decryption algorithm for extracting their results in the $Validate_{CT}$ transaction. This defines a consensus solution and the competition winner(s) as the optimal solution that appeared first in the ledger wins. To emphasize the importance of maintaining pools, consider the proposition below.

Proposition 3.1. (Nash Equilibrium feasibility) Assume that the mean block reward per solve transactions is higher than the mean of regular non-competition transaction fees. The competition rules constitute an equilibrium in which the best strategy of hash block signers is to gather as many $Solve_{CT}$ transactions as possible, whereas computational task solving miners best strategy is to send their solutions. The optimal strategy of the storage miner is to answer retrieve requests of computational miners and send them the data. The optimal strategy of the block assembly miners is to enable $Solve_{CT}$ transactions to be added.

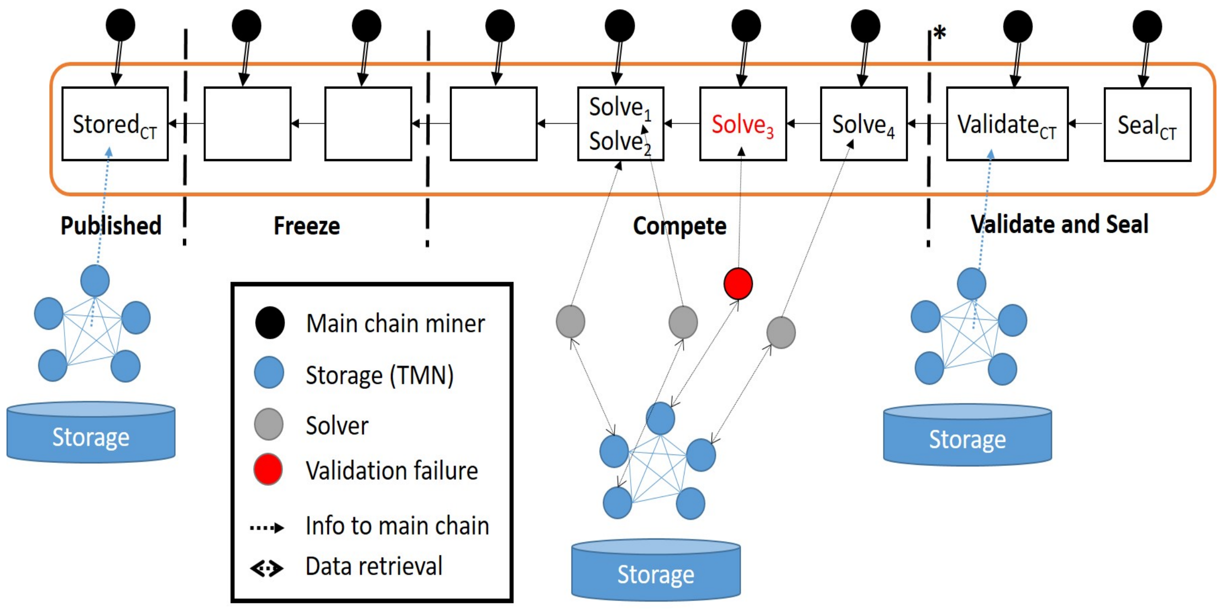

Figure 1: Illustration of the competition. Storage dTMN publishes the task to the network. After a freeze phase solving miners (gray) send out their solutions. In the validation phase the dTMN gets the solutions from the digital commitments given in the competition phase. In this case, the third proposed solution is marked as irrelevant (shown in red) in the validation transaction due to failure to fulfill the commitment. The asterisks marks that the validation phase may take a few blocks for the dTMN for reaching a consensus. Finally, a sealing transaction is added and profits are distributed.

Proof. Block signing PoW miners know that by including more solve transactions they increase their expected relative weight in the pool. Thus, when they assemble blocks the best strategy for increasing the expected profit is to include the solutions with the best declared scores. The exact number depends on three parameters: (1) how many competitions are alive, (2) the difference between the mean reward of standard transactions and solve transactions, and (3) the probability that the submitted commitment will fail. Storage miners are paid proportionally to their work. They would like to have as many computational miners as possible whose data copies were retrieved from them. On the other hand, the computational miners, which aim to win competitions by submitting their results, know that the submission associated fee is a PF. Thus, they only pay it in case they are selected to be winners and therefore they have incentive to submit solutions □

3.3.3 How to keep the solution safe

An important question at this point is how can we make sure that upon sending a solution an attacker cannot take it and claim it as is own? We propose two solutions. The first solution is based on cryptographic commitments. Here, the miner sends the hash of the solution and a nonce. After the competition is over the winner can prove his commitment by sending the real solution together with the nonce in order to finalize competition. The second solution is based on randomly partitioning the solution into shards. Here, the miner submits a few shards to each member of the dTMN. Each shard has a randomly assigned number, taken at random from a large space that the miner does not share with anyone. Only when all shard numbers are visible can a dTMN member order the shards and assemble the correct solution. This solution removes the need of the solving miner to stay alive after a competition, but is sensitive to collusion of the dTMN members.

3.4. The storage chain

Our storage component can be easily built upon existing storage blockchains (and even centralized storage solutions). Roughly, these systems address the following issues: (1) closing a deal in a reliable platform, (2) payment for storing the data, (3) how miners prove they hold a valid replica of the data, (4) payment for retrieving the data, (5) data encryption, and (6) system maintenance. Our requirements are only a subset of what these systems address. It is crucial however, that for a denentralized solution an off-chain micro-payment will be available: when two parties transfer data we use a protocol that ensures the transfer but keeps most of the information off chain (proofs and certifications are kept for tracking). As exemplary systems that provide all functions above we consider Filecoin [13] and Storj [14]. We briefly discuss the storage component below, for a thorough explanation see the **Appendix**.

Payment to the storage are done only after the solution was kept for some time factor (letting the client pull the solution). Only storage miners that validate the answer correctly (vote as the majority) on all the solutions will be paid. Per storage service, the storage miners are paid from the storage infrastructure pool. Their fees are extracted automatically only for services that were proved. As a result, each storage miner of the dTMN will be payed per given service proportionally to the amount of his work.

4. Protocol properties

4.1. Forging via 51% attacks

Carrying a 51% attack in our system, by definition, requires achieving 51% of both the PoS and the PoW components. This is a useful property of Decred: hash solving miners propose new blocks, whereas PoS miners approve past blocks. An adversary that reaches only 51% in the hash component still needs to get the proposed blocks approved. An adversary that holds 51% of the stakes cannot suggest new blocks. Thus, when there are many miners of both types, it becomes very difficult and resource consuming to try such attacks. This valuable property is not violated in our system thanks to the accumulation of past useful work.

4.2. $O(1)$ attacks

The competition's freeze phase makes $O(1)$ attacks that involve low difficulty problems unlikely because the adversary has no guarantees on winning back the submission fees. As a result, $O(1)$ attacks with notable outcomes require acquiring many coins, submitting problems with a very high difficulty, which will also results in locking them for long time periods. This is, in essence, equivalent to performing PoS because the adversary first needs to acquire resources in order to submit a new task (a client that has no coins cannot offer fee_{solve}), and then these coins are locked until the competition rewards are distributed after $> B_{dist}$ blocks. On the other hand, our *PoS higher profit invarnace* guarantees that the same $O(1)$ adversary is better off with performing PoS, which does not even require computational resources.

Under the constraints above we are still able to set the protocol parameters such that the honest miner that won a competition gets more than fee_{solve} (submitted by the client). Why is this even possible? Keep in mind that we have a gap: using vstakes to purchase tickets is not equivalent to using money to buy tickets. The former is not entitled to the extra rewards that the pure PoS miner expects. This gap allows earning $> 100\%$ compared to the client-submitted fees, as we shall show next.

Theorem 4.1. For a ticket price y there exists a set of parameter assignment to r, p_{pools}, p_{vstake} such that the expected profit of a pure PoS mining is higher than that of an $O(1)$ adversary.

Proof.

Assume that the adversary submitted a problem with an offer to pay y. By definition, $p_{pools}y$ is paid to the pools that reward the storage and main chain infrastructure. He then gets $y(1-p_{pools})$ back by solving his own problem and $p_{vstake}y(1-p_{pools})$ in virtual stakes. These are minted new coins and there is no added reward. The total reward of the adversary is:

$$y(1-p_{pool}) + p_{vstake}(1-p_{pools}) = \\ (p_{vstake}+1)(1-p_{pools})y$$

Performing PoS in the same time by buying a ticket in price y would have resulted in an expected reward of:

$$Pr(ticket\ is\ selected)ry + \\ yPr(ticket\ is\ not\ selected) = \\ 0.95ry + 0.05y = y(0.95r + 0.05)$$

Setting $r = 1.1$ we get an expected pay back for PoS of $1.095y$. We can now easily set the other parameters such that:

$$1 < (p_{vstake}+1)(1-p_{pools}) < 1.095$$

For example, $p_{vstake} = 0.25, p_{pools} = 0.15$ yields 1.0625. Alternatively, if we want to mint less coins per virtual stake then: $p_{vstake} = 0.15, p_{pools} = 0.1$ yields 1.035. $\square$

We proved the theorem above by example. However, note that the analysis above defines an infinite number ways to select parameters. To see this, assume we wish to keep a given percentage ϵ_{pos} difference between the expected PoS reward and the reward of a computational miner. Assume we wish to maximize the computational miners profit. This amounts to the following constrained problem:

$$argmax_{p_{vstake}, p_{pools}}(p_{vstake}+1)(1-p_{pools})$$

s.t.

$$(1) \quad (0.95r + 0.05) - \epsilon_{pos} = \\ (p_{vstake}+1)(1-p_{pools}) \\ (2) \quad 0 \le p_{vstake}, p_{pools} \le 1$$

Figure 2 below illustrates how for $\epsilon_{pos} = 0.01$ we get an infinite set of ways to adjust the parameters for different expected PoS rewards.

4.3. Suboptimal solutions

An honest miner that did substantial work is likely to offer a small transaction fee. If the PoW miners are bombarded with random solutions, the honest miner looks at the ledger and knows that his solution is better (because solution scores are not encrypted). He is then likely to offer a transaction fee that is similar to that of an average non-competition transaction fee. In this case the SSA (suboptimal solutions adversary) that tries to overwhelm the system with solutions will have to pay substantial funds. This concept is similar to early suggestions for fighting spam emails: the spammer needs to pay a lot to succeed, whereas the honest users pay negligible sum per transaction [4].

4.4. The dTMN and fault tolerance

Byzantine fault tolerance (BFT) is the dependability of the system in failure of components (and the identification thereof) [15]. We discussed above several mechanisms to prevent colluding attempts (e.g., make storage miners be more

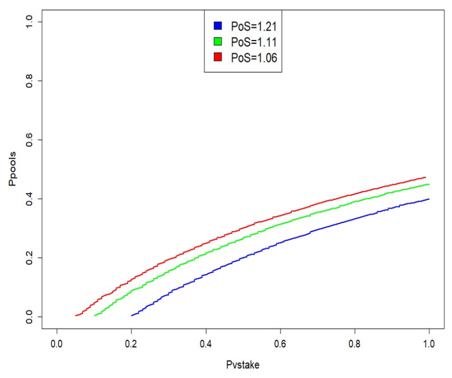

Figure 2: Computational miner reward parameters under the PoS constraints. For different PoS expected reward factors and a desired margin of $\epsilon_{pos} = 0.01$ each line shows the set of p_{vstake} and p_{pools} combination that satisfies the constraint in which PoS is more profitable than $O(1)$ attack.

like DASH masternodes). However, for BFT, note that our system always depends on both reaching a consensus and getting it approved. For the competition validation phase we give vote management to the block miners by letting storage miners submit commitment of their votes. Thus, there is no need for dTMN members to coordinate their response and in case some miss their chance due to failure, there are still others that can reach consensus. The dTMN members are expected to send their results separately but consistently until the validate and seal transactions are published. This is similar for requesting masternode members or PoS right owners from other protocols to be present in a given time otherwise they lose their rights.

4.5. Block withholding attacks

Block withholding, also called selfish mining, is an attack in which a large mining pool withholds mined blocks in hope to mine another one and let the network pursue an orphan block. By doing so, the attacker can present its longer chain later and make the network accept it as the new consensus [16]. Several analyses had deepen our understanding of such attacks, but were mainly confined to Bitcoin itself or to other very similar protocols [17, 18]. Nevertheless, this attack still requires large computing power, and is currently not suitable for hybrid PoS systems, especially ones similar to Decred in which the PoS approves blocks. Here, withholding a block and letting the network focus on a shorter chain is in fact not beneficial once the PoS mechanism approved the shorter chain. In these cases, the shorter chain is accepted as the consensus because it is the only one whose block before the current head was approved. Even though our system aims to deviate PoW power to the computational tasks and thus reduce the hash difficulty, it does not come at the expense of this property.

4.6. Can miners collude?

Given that the computational task miners all compete for rewards from the client, there is a certain risk of colluding. For example, miners may agree to send out random solutions without doing any work. Note however that we only need a **single honest miner** per computational task to render the entire colluding effort barren. This property weakens the conspiring adversaries substantially as they do not necessarily know that all miners had agreed, which is very difficult if the network is large. Moreover, even under such agreement the entire system is still under the prisoner's dilemma: even rational colluding miners are not likely to cooperate as betraying the partners results in a greater reward. Finally, we can consider active mechanisms that can be added to prevent such attacks (if required in practice). First,selecting tasks at random and submitting the solution at a random time point dismantles the attack and provides a reasonable solution to the costumer. Second, having a public reputation board will be useful in any case. However, this is a more involved solution that will require a thorough examination of the incentives.

5. Discussion

Previous suggestions aimed for alleviating the dependence on hash puzzles with a completely new mechanism. This created a problem with the need to correctly estimate the difficulty of each instance from each considered problem. For most problems, this task is currently not feasible as complexity analysis usually involves obtaining bounds for the worst-case running time. Our approach is radically different: we let the client decide how much each task is worth. Thus, the clients evaluate the cost by comparing to other alternatives in the market and then suggest what they consider a fair proposal. Of course, this will need to be balanced between the goal of saving costs (compared to other alternatives) and attracting miners.

Our protocol, like any decentralized blockchain protocol, is bound to have redundancy. First, the blockchain itself has many copies. Second, each member of the dTMN has a copy of the data of a task. Finally, the computational miners compete for gains from the same problems. The first two redundancies result from the need to reach consensus between agents in the system, which is crucial for decentralization. We therefore accept these two as needed constraints. The third redundancy,however, is a much more delicate issue.

To make our discussion more formal we compare our system to an optimal centralized system with no reliability or connection issues. Let n_m be the number of miners in the system and n_t be the number of tasks. We make the following simplifying assumptions: (1) n_m and n_t are the same in both systems (2) all miners have the same computing power, (3) all miners have the same same energy consumption per computing unit per time, (4) in our system miners select tasks at random, uniformly over all tasks, (5) all tasks require the same resources exactly, they take one hour to run in a single machine (to make time discrete and standardized), and (6) all tasks are all worth the same value v. Let e be the ratio between the income per hour in the centralized system and the expected income in our system.

Clearly, the worst-case scenario for our system is when there are only a few tasks and many miners solve the same problems using the same deterministic algorithm. In this case, the expected number of solved tasks in our system is $\frac{n_t}{n_m}$. Assume $n_t < n_m$, then: $e = \frac{vn_t}{vn_t/n_m} = n_m$ which means that the system is inefficient in a way that is linear in the number of miners.

However, this analysis is irrelevant as our goal is not to replacing current centralized cloud systems. Alternatively, we seek to solve computing and modeling bottlenecks that are widely common in large optimization problems. Moreover, the efficiency gap illustrated above quickly reduces when the user's perspective is taken into consideration. Most importantly, for most optimization tasks in AI one of the following is true: (1) users need to rerun their learning tasks several times (e.g.,when the algorithm depends on a random starting point and hyperparameters), and (2) the problem can be distributed into many small tasks that can run in parallel.

These two points directly affect e in a way that creates a trade-off. First, distributing a large task into many small ones means that both the difficulty of each task can be decreased and n_t can be much larger than n_m. In these cases, the overall income of the system depends on how many tasks can a miner take on. In any case, distribution can bound the number of miners per competition. Second, the true cost for users is a function of the total number of times they need to run the algorithm in practice. Moreover, even with different attempts there is no guarantee that the solution converged to the best one (or even a useful one) and no guarantee that the search space was thoroughly examined. In contrast, using our competition-based paradigm there is a single run of the competition and the powerful miners compete by testing different options. Thus, the costumer can propose a much lower cost as compared to current estimated costs.

5.1. Machine learning competitions

In this work we focused on solving optimization problems. Clearly these are useful for many AI tasks. However, note that for the goal of training models, we can slightly adapt the protocol to mimic how current machine learning competitions work. This requires having the validation phase run by the dTMN run the trained models on new data, which can be provided by the user after all solutions were submitted. Thus, our system can be used for implementing competitions en masse. We next briefly discuss machine learning competitions.

The lack of one master algorithm for all machine learning or optimization problems is a known result, also known as the *no free lunch theorem* [19, 20]. Empirical difficulties over time had led the community to establish competitions and challenges in order to achieve progress. There are many examples for successful competitions in Machine learning and we cannot go over them all here. These competitions are especially useful for advancing applications that had proved difficult over time. We focus here on two examples that historically led to substantial progress. For example, the Netflix challenge that led to a marked progress in Collaborative Filtering as tool for creating recommendation systems [21]. As another example, we consider Deepl Learning, the recent renaissance of neural networks that was accelerated due to the high success of this methodology in the ImageNet competition [22, 23].

Based on the empirical observation that competitions are a useful way to enhance machine learning, our system is expected to improve upon state of the art in many tasks. Specifically, large Deep Learning (DL) tasks seems suitable as the training relies both on inspection of many hyperparameters, some of which may determine how to utilize many GPUs, and performing (possibly asynchronous) stochastic gradient descent in an extremely large space [24, 25, 26, 27, 28].

5.2. Network maintenance costs

As a final point for discussion, we reiterate the set of fees that the user submits for a task: (1) fee_{tr}: same as any other transaction, (2) fee_{sub}: a bid for paying for storage, (3) retrieval fee: used for paying the storage for sending data, (4) validation: used for validation payment, (5) maintaining: used for paying the main chain for holding all other transactions, and (6) fee_{solve}: the payment to the winning solver. Note that the promised fees above can be predefined percentages of fee_{solve}. Moreover, the protocol can define a set of constraints, e.g., that the total sum of payments to the main chain pool is at least 15%. Optimal assignment of these parameters is still under research.

References

[1] P. Godsiff, "Bitcoin: Bubble or blockchain," in *Smart Innovation, Systems and Technologies*, vol. 38, pp. 191–203, 2015.

[2] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," *Www.Bitcoin.Org*, p. 9, 2008.

[3] V. Buterin, "Ethereum Whitepaper," 2015.

[4] C. Dwork and M. Naor, "Pricing via Processing or Combating Junk Mail," *Crypto*, pp. 139–147, 1992.

[5] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and Cryptocurrency Technologies Introduction to the book.* 2016.

[6] S. King and S. Nadal, "PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake," *Ppcoin.Org*, 2012.

[7] E. Duffield and D. Diaz, "Dash: A Privacy-Centric Crypto-Currency."

[8] C. Jepson, "DTB001: Decred Technical Brief," 2015.

[9] A. Chepurnoy, "PoS forging algorithms: multi-strategy forging and related security issues," 2015.

[10] C. G. Oliver, A. Ricottone, and P. Philippopoulos, "Proposal for a fully decentralized blockchain and proof-of-work algorithm for solving NP-complete problems," *CoRR*, vol. abs/1708.0, 2017.

[11] M. Ball, A. Rosen, M. Sabin, and P. N. Vasudevan, "Proofs of Useful Work," *IACR Cryptology ePrint Archive*, vol. 2017, p. 203, 2017.

[12] S. King, "Primecoin: Cryptocurrency with Prime Number Proof-of-Work," *King, Sunny*, vol. 4, no. 2, p. 6, 2013.

[13] ProtocolLabs, *Filecoin: A Decentralized Storage Network*. PhD thesis, 2017.

[14] S. Wilkinson, T. Boshevski, J. Brandoff, J. Prestwich, G. Hall, P. Gerbes, P. Hutchins, and C. Pollard, *Storj: A Peer-to-Peer Cloud Storage Network*. PhD thesis, 2016.

[15] H. Kirrmann, "Fault Tolerant Computing in Industrial Automation," *Lecture notes ABB Corporate ResearchETH*, 2005.

[16] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8437, pp. 436–454, 2014.

[17] S. Bag, S. Ruj, and K. Sakurai, "Bitcoin Block Withholding Attack: Analysis and Mitigation," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 8, pp. 1967–1978, 2017.

[18] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal selfish mining strategies in bitcoin," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9603 LNCS, pp. 515–532, 2017.

[19] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[20] D. H. Wolpert, "The Lack of a Priori Distinctions between Learning Algorithms," *Neural Computation*, vol. 8, no. 7, pp. 1341–1390, 1996.

[21] M. Wu, "Collaborative Filtering via Ensembles of Matrix Factorizations," *Biological Cybernetics*, no. October, pp. 43–47, 2006.

[22] A. Krizhevsky, I. Sutskever, and H. Geoffrey E., "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems 25 (NIPS2012)*, pp. 1–9, 2012.

[23] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[24] Y. A. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[25] I. Goodfellow, Y. Bengio, and A. Courville, "Deep Learning–book," *MIT Press*, vol. 521, no. 7553, p. 800, 2016.

[26] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. A. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large Scale Distributed Deep Networks," *NIPS 2012: Neural Information Processing Systems*, pp. 1–11, 2012.

[27] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," in *International Conference on Learning Representations Workshop Track*, 2016.

[28] Z. Wei, S. Gupta, X. Lian, and J. Liu, "Staleness-Aware Async-SGD for distributed deep learning," in *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2016-January, pp. 2350–2356, 2016.