Proof of Balanced Work: The Theory of Mining Hash Products

October 30, 2023

Coin Fu Master Shifu

***Abstract*—We propose a new method to multiplicatively combine different PoW mining algorithms into a new PoW like mining algorithm such that hashrate improvements of all individual algorithms are relevant for the final hashrate of the combination. This implies that for efficient mining all combined algorithms need to be efficiently mined in a balanced way. Hence our proposed method can serve as a new tool to carefully craft new algorithms for specific hardware requirements and for ASIC-resistance. For example a combination of CPU and GPU algorithms would require miners to use both, CPU and GPU hardware at the same time in order mine efficiently.**

I. Introduction

Cryptocurrencies need a mechanism to distinguish the consensus chain from an arbitrarily generated chain. There are several approaches to this problem, Bitcoin's Proof of Work (PoW) [1], Ethereum's Proof of Stake (PoS) [2] and Solana's Proof of History (PoH) [3] are just some examples.

In this work we consider PoW, the oldest and most battle-proven approach. There is a debate on the environmental implications and ethical aspects of PoW cryptocurrencies [4] in which opponents claim PoW mining is burning energy resources without actual use, thus driving global warming while proponents on the other hand argue that the traditional banking system, gold mining and more could be replaced to a large extent by cryptocurrencies such that the negative implication is marginal or even turned into a positive effect, especially due to the facts that the replaced industries also require a substantial amount of energy and fossil fuels, and PoW mining is carried out in places where electricity is cheap and often would just be lost if not used.

From a technical perspective securing a blockchain via PoW has several benefits such as good decentralization due to the parallel nature of PoW mining, natural and fair distribution according to hashrate, which is particularly important at launch phase of new projects, the simplicity of the approach which helps to avoid bugs and unforeseen mining issues, and a canonical way to scale the difficulty. Therefore PoW is often considered as the gold standard method to distinguish the consensus chain of a cryptocurrency [5], [6]. The trust it conveys and receives from investors is one of the main reasons why Bitcoin has been the undisputed leader of all cryptocurrencies by market capitalization ever since.

In this paper we propose *Proof of Balanced Work (PoBW)*, a novel method to combine different PoW hashing algorithms in a balanced way such that individual relative improvements in hashrate efficiency of each component are relevant for the mining efficiency of our proposed combination, and even when slow and fast hashing algorithms are combined there is no bottleneck effect in mining.

This paper is structured as follows. Section II explains mining, difficulty and retargeting in the classical PoW context. These concepts are then extended in Section III where we specify our proposed PoBW mining algorithm in detail. In Section IV we analyze and demonstrate methodology on how to most efficiently mine PoBW. Our findings are finally summarized in Section V. Mathematical proofs and auxiliary results are deferred to the appendix.

II. Classical Proof of Work

A. Mining

On an abstract level PoW *mining* is the process of iteratively computing a hash function on different inputs until a certain criterion to mine a block is met. A proper hash function generates a finite-length pseudo-random byte sequence which can be seen as a uniformly distributed finite precision binary number u in the interval $[0, 1)$ while the criterion $M_\tau(u)$ is usually just a simple check whether this number u is within a certain subinterval $S_\tau = [0, \tau]$, $\tau \in [0, 1)$, i.e.

$$M_\tau(u) = \begin{cases} 1 \text{ if } u \in [0, \tau] \\ 0 \text{ else} \end{cases} . \tag{1}$$

For simplicity we will ignore the fact that u has finite precision and therefore model u as a random variable U with uniform distribution on $[0, 1]$ when sampled as output of the hash function, hence the probability of the criterion M_τ being met is

$$\mathbf{P}(M_\tau(U)) = \mathbf{P}(U \in [0, \tau]) = \tau. \tag{2}$$

By adjusting τ we can control the probability that a random hash satisfies the criterion.

Usually the finite-precision representation of τ is called *target* and encoded in the block header in compressed form. For example this compressed form is called *nBits* and located at bytes offsets 72-75 in Bitcoin's block headers.

B. Difficulty

To every target τ corresponds a *difficulty* d which states how many hashes are necessary on average to satisfy the criterion M_τ. We will now elaborate the relation between τ and d. Similarly as above we model multiple hashes on different inputs as independent random variables $U^{(1)}, U^{(2)}, \dots$ with

uniform distribution on $[0,1]$. The first index I of these random variables satisfying the criterion M_τ is $I = \min\{i \in \mathbb{N}_+ | M_\tau(U^{(i)} = 1\}$ and we know that for $k \in \mathbb{N}_+$

$$\mathbf{P}\left(\min\{i \in \mathbb{N}_+ | M_\tau(U^{(i)}) = 1\} = k\right) = (1-\tau)^{k-1}\tau$$

by equation (2) and independence such that I is geometrically distributed. Since the difficulty is the expected required number of tries to satisfy the criterion M_τ for the first time, it is equal to I's expectation

$$d = \mathbf{E}[I] = 1/\tau. \tag{3}$$

For historical reasons Bitcoin scales the reported difficulty by a factor of 2^{-32} but in this work we stick to the above definition of d being the expected number of hashes necessary to mine a block.

C. Retargeting

An important ingredient of PoW blockchain technology is the *retargeting* which describes the adjustment of the difficulty according to the total hashrate of miners in order to ensure blocks are mined at constant time rate. If blocks are mined too slowly, the blockchain might become unusable. If they are mined too fast blockchain size could outgrow the hardware capabilities of nodes and network infrastructure. Furthermore an approximately constant mining rate is important for investors and miners alike because it provides confidence about future issue rate of new coins.

If the total hashrate doubles, the difficulty also has to double, if it halves the difficulty also has to halve in order to keep block generation rate at a constant level. However the current total hash rate is not directly measurable and needs to be estimated from past block timestamps[1].

Cryptocurrencies usually implement rules which makes faking block times by a large offset almost impossible. For example in Bitcoin there is a hard-coded rule that enforces that a block time must be larger than the median of the past 11 block times and another rule making peers ignore mined blocks with times later than 2 hours in the future such that by game theory no one will risk mining a block with a future time stamp and wait until it is time that it can be published without being ignored. This implies that block timestamps can be considered to some extent as a proxy of the real block creation time and can therefore be used for estimating the total hashrate.

In Bitcoin retargeting happens every 2016 blocks and the predefined rate is one block every 10 minutes, i.e. every 600 seconds. At this rate the 2016 blocks correspond to 2 weeks, for an increased/decreased hashrate this timespan will be shorter/longer respectively. Bitcoin retargeting works roughly as follows.

1) After a batch of 2016 blocks the timestamps of the first block and the last block[2] within that batch are used to compute an approximation of the time it took to mine these 2016 blocks.
2) This time is divided by 2016 to obtain an approximation of the average time t in seconds required to mine one block.
3) The difficulty is scaled by factor[3] $600/t$ which corresponds to scaling the target by the inverse $t/600$.

After retargeting the expected time required to mine future blocks is 600 seconds as desired, given that the future hashrate is equal to the average hashrate of the past 2016 blocks.

In general retargeting works by dividing the difficulty by the quotient q of theoretical time it should take for a specific amount of blocks to be mined by the estimated time the network actually needed. From equation (3) we know that the target t must be multiplied by the quotient q to achieve this desired effect on the difficulty.

III. Proof of Balanced Work

A. Motivation

It is an interesting question whether and how PoW mining algorithms can be combined to form new PoW mining algorithms and what new properties could be realized by such hybrids. The most basic way to form a new PoW algorithm as a combination is to apply classical PoW for a new hash function which is formed as a concatenation other hash functions. In such a setup the component hash function which is slowest to compute will always be the bottleneck for the computation of the concatenation. When these component hash functions are computed on different devices the faster devices' hash rates cannot be efficiently used.

For example when a very fast GPU specific hash function and a CPU specific hash function are concatenated and mined on a rig, most likely the CPU will be the bottleneck. If the CPU is reasonably capable to also compute the GPU hash function the concatenation of the two hash algorithms will be CPU-mineable and GPUs would just be waiting for the CPU bottleneck.

Therefore such simple concatenations are of limited use. In contrast we have developed a PoW mining methodology combining hash functions multiplicatively which

- will *not* suffer from such bottlenecks,
- requires each component hash function to be efficiently computed and
- can therefore be used for example to create new mining algorithms that require *both* a capable CPU *and* GPU for efficient mining.

This means our contribution opens the door to a completely new class of PoW mining algorithms which can be freely composed from established hash functions.

B. Definition

While a classical Proof of Work requires the *one* hashed value to satisfy some criterion, we define a Proof of Balanced Work to require a *hash product* of $m \in \mathbb{N}$ hash functions $h_1, \ldots, h_m$ evaluated on the *same argument* to satisfy a criterion.

[1] Usually, block chains contain a Unix timestamp in each block header specifying block creation time

[2] off-by-one error

[3] t will never be 0 due to the timestamp monotonicity rule involving the running median of the latest 11 timestamps enforced for every block.

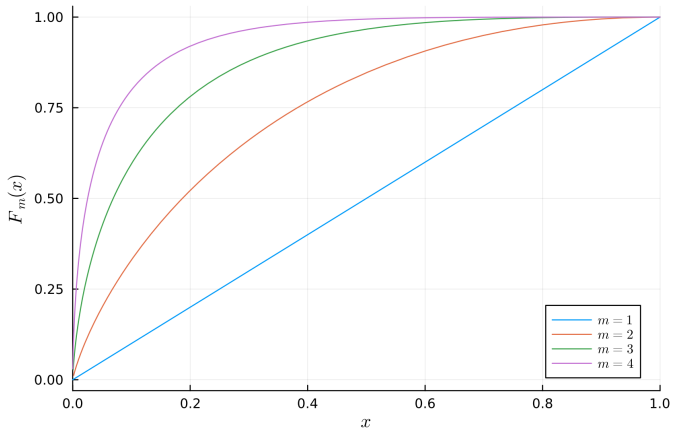

Figure 1. Cumulative distribution function F_m of the product of m independent uniform random variables on $[0, 1]$ for $m = 1, 2, 3, 4$.

A hash product can be canonically defined when each hash is considered as a binary number in the interval $[0, 1)$ as we already did in Section II-A. For $m \in \mathbb{N}$ such hashed values $u_1, \ldots, u_m \in [0, 1)$ obtained as the outputs of the hash functions $h_1, \ldots, h_m$ evaluated at the same argument, the PoBW criterion M_τ to mine a block is

$$M_\tau(u_1, \ldots, u_m) = \begin{cases} 1 & \text{if } \prod_{i=1}^m u_i \in [0, \tau] \\ 0 & \text{else} \end{cases} \qquad \text{for } \tau \in (0, 1].$$

Classical PoW is a special case of PoBW for $m = 1$.

If the hashed values are modeled as independent uniformly distributed random variables $U_1, \ldots, U_m$ the probability that the criterion M_τ is satisfied is

$$\mathbf{P}\left(M_\tau(U_1, \ldots, U_m)\right) = \mathbf{P}\left(\prod_{i=1}^m U_i \leq \tau\right) = F_m(\tau) \qquad (4)$$

for a given target τ and F_m being the cumulative distribution function of the product of m iid uniform random variables on $[0, 1]$. Lemma A.2 provides an explicit formula and Figure 1 depicts F_m for different values of m.

C. Difficulty

Similarly to Section II-B we define the difficulty d for PoBW to be the expected number of random hash products that are necessary to compute in order to satisfy the criterion M_τ. Let $(U_1^{(i)}, \ldots, U_m^{(i)})$, $i \in \mathbb{N}_+$ be independent groups of m independent uniform random variables. Then the hash products $X_m^{(i)} := \prod_{j=1}^m U_j^{(i)}$, $i \in \mathbb{N}$, are independent, each with cumulative distribution function F_m. The first index I of these random variables satisfying M_τ is $I = \min\{i \in \mathbb{N}_+ | M_\tau(X^{(i)}) = 1\}$ and it holds for $k \in \mathbb{N}_+$ that

$$\mathbf{P}\left(I = k\right) = \left(1 - F_m(\tau)\right)^{k-1} F_m(\tau)$$

by equation (4), that is, I is geometrically distributed with expectation equal to the difficulty

$$d = \mathbf{E}[I] = 1/F_m(\tau) \qquad (5)$$

for any target $\tau \in (0, 1]$. Here again we observe that classical PoW is a special case of PoBW for the case $m = 1$ since $F_1(\tau) = \tau$.

D. Retargeting

The inverse relation of the difficulty d on the target τ from classical PoW is no longer true in PoBW as can be seen in equation (5). Instead the difficulty involves the cumulative distribution function F_m which leads to a skewed inverse relation of d on τ.

Therefore the retargeting strategy from Section II-C cannot be used in general for PoBW. To illustrate the problem we compare the effect of halving the target τ on $F_m(\tau)$ for $\tau = 0.4$ and a small value $\tau = 4.0 \times 10^{-22}$ in Tables I and II.

Table I
HALVING THE TARGET $\tau = 0.4$ FOR $m = 1, \ldots, 4$.

m	$F_m(\tau)$	$F_m(\tau/2)$	$\frac{F_m(\tau/2)}{F_m(\tau)}$
1	0.4	0.2	0.5
2	0.77	0.52	0.68
3	0.93	0.78	0.84
4	0.99	0.92	0.93

Table II
HALVING THE TARGET $\tau = 4.0 \times 10^{-22}$ FOR $m = 1, \ldots, 4$.

m	$F_m(\tau)$	$F_m(\tau/2)$	$\frac{F_m(\tau/2)}{F_m(\tau)}$
1	4.0×10^{-22}	2.0×10^{-22}	0.5
2	2.0×10^{-20}	1.0×10^{-20}	0.51
3	5.1×10^{-19}	2.6×10^{-19}	0.51
4	8.5×10^{-18}	4.4×10^{-18}	0.52

We observe that halving the target τ *does not* halve the probability to mine a block for $m \geq 2$, which is a manifestation of the skew effect mentioned above. However we have proven that this effect becomes irrelevant for small τ:

Lemma III.1. *For a number $m \in \mathbb{N}$ of independent uniform random variables $U_1, \ldots, U_m$ on the interval $[0, 1]$ consider their product $X := \prod_{i=1}^m U_i$. The conditional distribution of the scaled random variable X/ε given $X \leq \varepsilon$ converges weakly to a uniform distribution on $[0, 1]$ as $\varepsilon \to 0$.*

The lemma implies that

$$\forall \underline{c} \in (0, 1), \bar{c} > 1 \quad \lim_{\tau \to 0} \sup_{a \in [\underline{c}, \bar{c}]} \left| \frac{F_m(a\tau)}{F_m(\tau)} - a \right| = 0,$$

i.e. for a capped factor[4] a, scaling small targets τ by a will also approximately scale the probability to mine a block by that factor, similarly to classical PoW. In Table II, we observe this in action for $a = 0.5$.

For block chain applications using Proof of Balanced Work, this means that despite the relation between τ and the difficulty d in equation (5) is not an exact inverse relation, in practice we can ignore this fact and still assume an inverse relation because the target is usually sufficiently small to rely on Lemma III.1. In particular we can use the retargeting logic from classical PoW.

[4] In Bitcoin the retarget scale factor is capped to values in $[0.25, 4]$.

IV. Filtered Mining for PoBW

A. Optimal Mining Approach

Since Proof of Balanced Work involves a hash product of the same argument, mining such an argument requires the evaluation of all m hash functions on candidate argument until an argument is found whose hash values satisfy the criterion M_τ for a given target τ.

The m hash operations are not required to by synced per argument, instead whenever the involved hash functions are evaluated at different hash rates, the faster hash functions outputs can be used to filter out promising arguments to be evaluated in the other hash functions.

More formally assume our compute resources allow us to compute of $n_i \in \mathbb{N}_+$ evaluations of h_i, $i \in \{1, \dots, m\}$ in one batch without loss of generality let $n_1 \geq \ldots \geq n_m$. Obviously, to achieve the highest probability of mining a block the best computational strategy is to first compute n_1 evaluations of h_1 on different arguments, then evaluate h_2 on the arguments corresponding to the smallest n_2 outputs seen as binary numbers in $[0, 1)$, then evaluate h_3 on the arguments corresponding to the n_3 smallest hash products of outputs of h_1 and h_2 etc. until we arrive at n_m different arguments for which we have computed the evaluation of all m hash functions to form the hash product and check the criterion M_τ. This optimal strategy is an m-stage iterative procedure of evaluating h_i on n_i candidates, sorting by $\prod_{j=1}^{i} h_j$ and filtering the lowest n_{i+1} candidates at each stage $i = 1, \dots, m-1$ with a final evaluation of h_m on the remaining n_m candidates to form the full hash product $\prod_{j=1}^{m} h_j$.

This approach involves sorting at the first $m-1$ stages which may not be feasible in efficient implementations. Therefore we do not consider this optimal approach in detail but introduce a slight modification to obtain a more practical approach. However we want to note that a mathematical analysis of this optimal approach would involve conditional integrals over Beta-distributions which are the distributions of order statistics of independent uniform random variables, see Lemma A.1.

B. Practical Mining Approach

In the above setting, at each stage $i = 1, \dots, m-1$, instead of sorting $\prod_{j=1}^{i} h_j$ and filtering the lowest n_{i+1} candidates we filter arguments where $\prod_{j=1}^{i} h_j$ is below a hard-coded threshold c_i which we will specify below. Note that for a uniform random variable U on $[0, 1]$ and a filter threshold $c \in (0, 1]$, conditionally on $U < c$, U is uniformly distributed on $[0, c]$. In particular each complete hash product of all m hashed values computed in this hard-coded filter approach satisfies the criterion M_τ with probability

$$\mathbf{P}\left(U_m \prod_{i=1}^{m-1} c_i U_i \leq \tau\right) = F_m\left(C\tau\right) \tag{6}$$

for independent uniform random variables $U_1, \dots, U_m$ on $[0, 1]$, target τ and $C := \prod_{i=1}^{m-1} c_i^{-1}$

At each stage $i \in \{1, \dots, m-1\}$ the number of candidates passing the threshold condition is random but we can define $c_i := \prod_{j=1}^{i} F_j^{-1}(n_{j+1}/n_j)$ such that on average n_{i+1} candidates pass stage i, which can be checked by induction. By the law of large numbers, the variance in these candidate numbers at each stage can be smoothed if many such batches are computed with work jobs queued between compute batches. In particular we can assume that in every batch this iterative hard-coded filtering approach produces n_m complete hash products of all m hashed values, mining a block with probability

$$1 - \left(1 - F_m\left(C\tau\right)\right)^{n_m}. \tag{7}$$

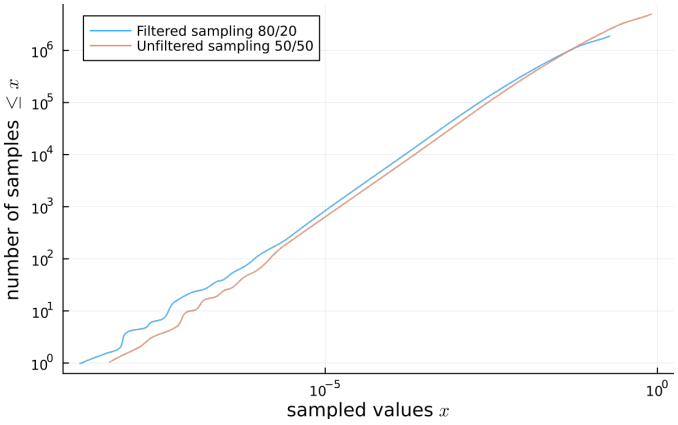

Figure 2. Logarithmic plot comparing sampled the empirical distribution of sampled hash products from the filtered and unfiltered approaches.

C. Examples

We will now provide three examples that illustrate filtered mining and demonstrate how to balance available hash rate between different component hash functions in PoBW. For simplicity we consider the case of $m = 2$ combined hash functions.

1) Simulation of Filtering: To demonstrate the positive effect of filtered mining we consider 10 million hashes. As already explained they can be regarded as binary numbers in the interval $[0, 1]$. We partition these hashes into two groups g_1, g_2 and consider them as output values of h_1, h_2 respectively. To empirically compare the naive unfiltered with the optimal filtered approach we use the following Julia code:

```
unfiltered = rand(5000000) .* rand(5000000)
filtered = sort(rand(8000000))[1:2000000] .* rand(2000000)
```

The unfiltered hash product samples are generated by two equally sized groups g_1, g_2 with $5\,000\,000$ hashes each whereas the filtered hash product samples are generated by first sampling $8\,000\,000$ hashes in group g_1, then filtering out the lowest $2\,000\,000$ hashes, and finally sampling another $2\,000\,000$ corresponding hashes for the group g_2 to form the pair-wise product. We plot the empirical distribution of both approaches in logarithmic scale in Figure 2 and observe that for lower sampled values of x the blue line is above the red line, which indicates that the filtered approach generates lower hash products than the unfiltered approach.

In particular the filtered approach generates more small hash products, and is therefore better than the unfiltered approach, which is remarkable considering the fact that in the unfiltered approach we generate $5\,000\,000$ hash products in total but only $2\,000\,000$ in the filtered approach. This means that to generate

low hash products, the positive effect of filtering out high-quality h_1 candidate arguments to apply h_2 on is stronger than the negative effect of having fewer total number of hash products in the filtered setting.

2) Balancing computations on the same device: Assume that we have a device which can compute a hash function h_1 at rate 500kh/s and another hash function h_2 at rate 125kh/s. We want to balance the computations of h_1 and h_2 on the device such that the probability to mine a block using PoBW is maximal in a batch computation during a time slot of duration t to be specified below. We denote the fraction of the available computational resources spent on hashing h_1 by α.

This optimization problem is is depicted in Figure 3. On the

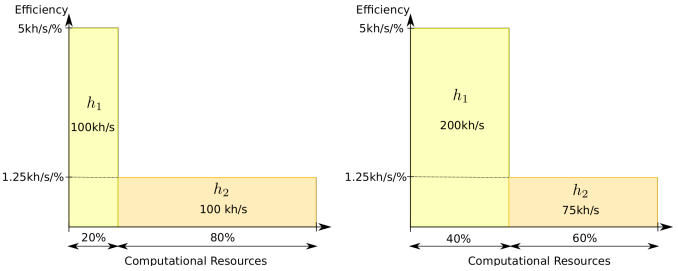

Figure 3. Different partitions of computational resources among two hash functions with different efficiencies 5kh/s/% (yellow) and 1.25kh/s/% (orange).

left hand side the balance of computational resources between h_1 and h_2 is 20/80 percent ($\alpha = 0.2$) and the overall hash rate of complete hash products is 100kh/s while the right hand side with allocation of 40/60 percent ($\alpha = 0.4$) yields a hashrate of only 75k complete hash products per second. However if we filter only the arguments of the best 75kh/s of the 200kh/s to be subsequently fed into the slower hash function, it is not clear per se which configuration is more efficient for PoBW mining.

We set the batch duration to $t = 0.1$ seconds, such that the number of hashes computed during this time is equal to $n_1(\alpha) = \alpha N_1$, $n_2(\alpha) = (1-\alpha)N_2$ with $N_1 = 50\,000$ and $N_2 = 12\,500$. Now by equation (7) the probability of mining a block within these 0.1 seconds for resource allocation parameter α is

$$P(\alpha) := \begin{cases} 1 - \left(1 - F_2\left(\tau \frac{(1-\alpha)N_2}{\alpha N_1}\right)\right)^{(1-\alpha)N_2} & \text{for } \alpha \geq \frac{N_2}{N_1+N_2} \\ 1 - \left(1 - F_2\left(\tau \frac{\alpha N_1}{(1-\alpha)N_2}\right)\right)^{\alpha N_1} & \text{else} \end{cases}$$

since $C = c_1^{-1}$ with $c_1 = F_1^{-1}(n_2/n_1) = n_2/n_1$. The case distinction comes from the fact that we assumed that $n_1 \geq n_2$, otherwise we need to swap the roles of n_1 and n_2. For the target $\tau = 10^{-8}$ this block mining probability function P is shown in Figure 4. The optimizer $\hat{\alpha}$ that and maximize P is $\hat{\alpha} \approx 0.9349$ in this case. Figure 5 shows the optimal balance parameter $\hat{\alpha}$ in other settings. It depends on the target τ, N_1 and N_2.

The probability to mine a block within one second

$$1 - (1 - P(\alpha))^{\frac{1}{t}}$$

does *not* depend on the batch duration t because N_1 and N_2 are proportional to t.

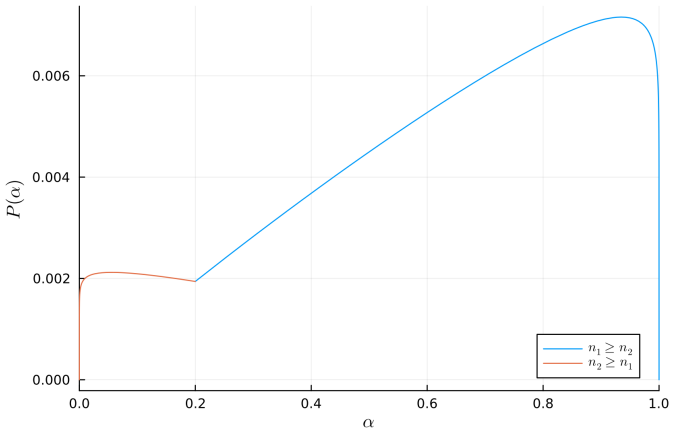

Figure 4. Cumulative distribution function F_m of the product of m independent uniform random variables on $[0, 1]$ for $m = 1, 2, 3, 4$.

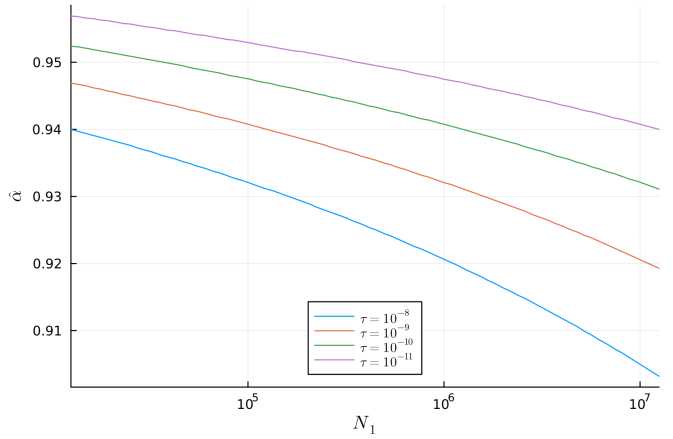

Figure 5. Optimal balance parameter $\hat{\alpha}$ that maximize P for different targets τ and $N_1 \in [N_2, 10^3 N_2]$, $N_2 = 12\,500$.

3) Balancing configurations of specialized hardware: We will now illustrate how PoBW allows the creation of PoW algorithms which can only be mined efficiently if, CPU and GPU capabilities are available and balanced. We assume that we have $m = 2$ hash functions h_{CPU} and h_{GPU} combined in our PoBW setting. For simplicity we will further assume that h_{CPU} can only be mined on CPU and h_{GPU} can only be mined on GPU.

In order to demonstrate that optimal mining requires CPU and GPU capabilities to be balanced we will compare hypothetical systems with different CPU and GPU configurations. Table III shows wattage and hashrate of 5 different hypothetical CPU and GPU configurations.

Similarly to the previous example for batch duration t, in a batch we compute $n_1 = r_1 t$ GPU hashes and $n_2 = r_2 t$ CPU hashes for GPU and CPU hash rates $r_1 < r_2$. Plugging this into equation (7) yields

$$P(\tau, t) = 1 - \left(1 - F_2\left(\tau \frac{r_1}{r_2}\right)\right)^{tr_2}$$

for the probability to mine a block in that batch. Following the same argumentation as in to the derivation of the difficulty

Table III
HYPOTHETICAL CPU AND GPU CONFIGURATIONS FOR MINING h_{CPU} AND h_{GPU} RESPECTIVELY.

CPU	Hashrate	Power	GPU	Hashrate	Power
CPU1	0.2kh/s	10W	GPU1	1mh/s	0.2kW
CPU2	1kh/s	50W	GPU2	2mh/s	0.4kW
CPU3	2kh/s	100W	GPU3	4mh/s	0.8kW
CPU4	3kh/s	150W	GPU4	10mh/s	2kW
CPU5	6kh/s	300W	GPU5	25mh/s	5kW

in equation (3) the expected number of batches to mine a block is the inverse of this quantity. Scaling this inverse by t yields the expected mining time $tP(\tau,t)^{-1}$ per block which can be multiplied by the total wattage of the CPU and GPU configuration to obtain the expected amount of energy consumption. Note that expected time to mine a block is $tP(\tau,t)^{-1} \approx r_2^{-1}F_2(\tau\frac{r_1}{r_2})^{-1}$ for t,τ small enough which holds true for practical settings such that there is only a minor dependence on the batch duration t.

In Table IV we have computed this expected energy consumption to mine a block for all 25 combinations of CPU and GPU configurations.

Table IV
EXPECTED ENERGY CONSUMPTION IN KWH PER MINED BLOCK USING POBW FOR DIFFERENT CPU/GPU CONFIGURATIONS.

	CPU1	CPU2	CPU3	CPU4	CPU5
GPU1	4.41	4.55	4.76	5.12	5.57
GPU2	4.69	4.43	4.40	4.55	4.84
GPU3	5.37	4.69	4.43	4.42	4.62
GPU4	6.11	5.02	4.54	4.39	4.51
GPU5	8.36	6.11	5.02	4.47	4.41

We can observe that a *weak* GPU configuration like GPU1 paired with a *strong* CPU configuration like CPU5 requires a suboptimal amount of energy (5.57 kWh), the same holds true for the opposite configuration of *strong* GPU5 paired with the *weak* CPU1 (8.36 kWh). Efficient mining requires CPU and GPU capabilities to be balanced. Upgrading only CPU or GPU will still improve hashrate but at the same time increase the energy consumption disproportionately.

At this point the question arises whether it is better to find a balance of weaker hardware components or should one consider an even stronger balanced CPU and GPU combination to mine more efficiently. If available CPU and GPU hashrates both scale linearly with their power drain like in this example, only the relation between CPU and GPU hashrate is relevant, the synchronous scaling of both can be accounted for by adapting the batch duration t and is therefore almost irrelevant. One can even do an continuous analysis at some fixed energy budget in this case, completely similarly to the previous example, to find the perfect balance $\hat{\alpha}$ of CPU and GPU power allocation for given hash per watt ratios.

V. SUMMARY

In this work we have presented Proof of Balanced Work which is a novel Proof of Work variant that combines *multiple* component hash functions in a multiplicative way instead of just one hash function in classical Proof of Work. Our presented method does not suffer from bottlenecks like a plain concatenation of hashes would and allows the creation of completely new mining algorithms to fit specific needs. For example it can be used to form a mining algorithm which can only be mined efficiently if *balanced* CPU and GPU capiabilities are available, while current mining algorithms are mined either on CPU or GPU but do *not* require both at the same time for efficiency.

We have formally explained the concepts of mining, difficulty and retargeting in the classical setting and then analyzed how these concepts can be transferred to PoBW.

In addition we presented an optimal and a practical approach for mining PoBW in batches. The key to avoid bottlenecks in mining different hash functions at different hash rates is to iteratively filter promising argument candidates based on the faster hash functions outputs in multiple stages. For the practical approach we provided explicit formulas for the probability to mine a block in a batch. We finally demonstrated filtered mining and showed how to balance mining setups for best efficiency.

We believe that the novel methodology, theory and examples presented in this work open the door to a new era of Proof Of Balanced Work mining algorithms due to the very favorable properties of forming hash products to combine established hash functions: simplicity, avoidance of mining bottlenecks and the possibility to target specific balanced hardware setups which can be particularly useful for the design of ASIC-resistant mining algorithms.

APPENDIX

Below we list two well-known lemmas, see for example [7] and [8] for reference.

Lemma A.1. *Consider iid uniform random variables $U_1,\dots,U_n$ on the interval $[0,1]$. The k-th smallest element $U^{(k)}$ has distribution*

$$U^{(k)} \sim \text{Beta}(k, n+1-k).$$

Lemma A.2. *The density $f_X:\mathbb{R}\to\mathbb{R}$ of the product $X := \prod_{i=1}^m U_m$ of $m\in\mathbb{N}$ independent uniformly distributed random variables $U_1,\dots,U_m$ on $[0,1]$ is*

$$f_X(x) = \begin{cases} \frac{(-\log(x))^{m-1}}{(m-1)!} & \text{if } x\in[0,1] \\ 0 & \text{else} \end{cases}, \tag{8}$$

and its cumulative distribution function F_m is therefore

$$F_m(x) = x\sum_{k=0}^{m-1}(-1)^k\frac{\log(x)^k}{k!} \text{ for } x\in[0,1]$$

by the formula $\int_0^x \log(x)^n \mathrm{d}x = x\sum_{k=0}^n(-1)^{n-k}\frac{n!}{k!}\log(x)^k$, $n\in\mathbb{N}$.

Proof of Lemma III.1. From Lemma A.2 we know the density function of X. The density of X/ε is $f_{X/\varepsilon}(x) = f_X(\varepsilon x)$ for all x. Note that for all $\tau \in (0,1]$ we have

$$\begin{aligned}
&\lim_{\varepsilon\to 0} \sup_{y\in[\tau,1]} \frac{f_{X/\varepsilon}(y)}{f_{X/\varepsilon}(1)} = \lim_{\varepsilon\to 0} \sup_{y\in[\tau\varepsilon,\varepsilon]} \frac{f_X(y)}{f_X(\varepsilon)} \\
= &\lim_{\varepsilon\to 0} \left(\sup_{y\in[\tau\varepsilon,\varepsilon]} \frac{\log(y)}{\log(\varepsilon)} \right)^{m-1} = \lim_{\varepsilon\to 0} \left(\frac{\log(\tau\varepsilon)}{\log(\varepsilon)} \right)^{m-1} \\
= &\left(\lim_{\varepsilon\to 0} \frac{\log(\tau)+\log(\varepsilon)}{\log(\varepsilon)} \right)^{m-1} = 1. \qquad (9)
\end{aligned}$$

It follows that for all $\tau \in (0,1]$

$$\begin{aligned}
&\lim_{\varepsilon\to 0} \sup_{x\in[0,1]} \left| \int_0^x \frac{f_{X/\varepsilon}(y)}{f_{X/\varepsilon}(1)} \mathrm{d}y - x \right| \\
\leq &\lim_{\varepsilon\to 0} \sup_{x\in[0,1]} \int_0^x \left| \frac{f_{X/\varepsilon}(y)}{f_{X/\varepsilon}(1)} - 1 \right| \mathrm{d}y \\
\leq &\lim_{\varepsilon\to 0} \int_0^1 \left| \frac{f_{X/\varepsilon}(y)}{f_{X/\varepsilon}(1)} - 1 \right| \mathrm{d}y \\
\leq &\lim_{\varepsilon\to 0} \int_0^\tau \left| \frac{f_{X/\varepsilon}(y)}{f_{X/\varepsilon}(1)} - 1 \right| \mathrm{d}y + \int_\tau^1 \left| \frac{f_{X/\varepsilon}(y)}{f_{X/\varepsilon}(1)} - 1 \right| \mathrm{d}y.
\end{aligned}$$

The dominated convergence theorem guarantees that the first term can be bounded by

$$\lim_{\varepsilon\to 0} \int_0^\tau \left| \frac{f_{X/\varepsilon}(y)}{f_{X/\varepsilon}(1)} - 1 \right| \mathrm{d}y \leq \lim_{\varepsilon\to 0} \int_0^\tau \left(\frac{\log(\varepsilon y)}{\log(\varepsilon)} \right)^{m-1} \mathrm{d}y + \tau \leq 2\tau$$

and that the second term converges to 0

$$\lim_{\varepsilon\to 0} \int_\tau^1 \left| \frac{f_{X/\varepsilon}(y)}{f_{X/\varepsilon}(1)} - 1 \right| \mathrm{d}y = 0$$

by equation (9). We therefore have shown that the function $x \mapsto \int_0^x \frac{f_{X/\varepsilon}(y)}{f_{X/\varepsilon}(1)} \mathrm{d}y$ converges uniformly to the identity function on $[0,1]$. This implies that the cumulative distribution function of X/ε given $X/\varepsilon \leq 1$ converges pointwise to the cumulative distribution function of a uniform distribution. The statement of the lemma now follows from the Portmanteau theorem. □

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Web document., 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] C. Schwarz-Schilling, J. Neu, B. Monnot, A. Asgaonkar, E. N. Tas, and D. Tse, "Three attacks on proof-of-stake ethereum," in *Financial Cryptography and Data Security*, I. Eyal and J. Garay, Eds. Cham: Springer International Publishing, 2022, pp. 560–576.

[3] A. Yakovenko, "Solana: A new architecture for a high performance blockchain," Web document., 2017. [Online]. Available: https://solana.com/solana-whitepaper.pdf

[4] L. Badea and M. C. Mungiu-Pupăzan, "The economic and environmental impact of bitcoin," *IEEE access*, vol. 9, pp. 48 091–48 104, 2021.

[5] L. M. Bach, B. Mihaljevic, and M. Zagar, "Comparative analysis of blockchain consensus algorithms," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2018, pp. 1545–1550.

[6] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 3–16. [Online]. Available: https://doi.org/10.1145/2976749.2978341

[7] J. Gentle, *Computational Statistics*. Springer New York, NY, 01 2009.

[8] C. Dettmann and O. Georgiou, "Product of independent uniform random variables," *Statistics & Probability Letters*, vol. 79, pp. 2501–2503, 12 2009.

Coin Fu Master Shifu prefers to stay anonymous at this time. Should he ever want to claim credit for this work may here be his PGP public key:


```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQGNBGUzn5YBDACio8JQd9kKAoKEi137Pmn4E3b65WykoK7tYg3PNeIg6EAS+OuZ
G1u6I//iv2YBuzV1Jl3W0f9uMqJodZVhkWn/CLurUg7zyGXFgN8qkgI/pkq7o/JD
txAwwuD+kTy++NNsa7+OMa5PgV/q1d+ieeYjM0XLXTb8Kk3hUQx5us/jIrAavm35
LyDK0yHqG0HesAuowk0pjPI0zU2JUQbrYSbKbh0Zmuvy4QB78bgDt+Q4eFI+cg+D
Qf4BuMKOFsYHSu9+OmOb5sFpI8U3bjWN7izhJcsUYMouu9AttY3nWptG1Bpg2/nk
1iHoEwq7j3XKaO6doG8BY6IN+YrR3a37kXlj+StIqvdOAqaDVrIrgG1+BbvGjeU+
NbaZWWMbArLGdeyyPMaCxI0thYkqyXYNku2pTGTZUNGqYVyzoG3T81MVEkHTHGZ4
LTdGYqlsVf/E1+iPxZ66mRl9LkE7RBa+huWk+h5OW10RrpBVcJuvTeefGacBESvp
Ho5YhjQvPCGy3TEAEQEAAbQUQ29pbiBGdSBNYXN0ZXIgU2hpZnWJAc4EEwEKADgW
IQTR/W1/HXMAZ+gJgX3YQBkptMgPVQUCZTOflgIbAwULCQgHAgYVCgkICwIEFgID
AQIeAQIXgAAKCRDYQBkptMgPVWJqC/wIACt8XGShBcj5n6Mhswq7oIDhMvHhiCSl
0grOGDKLbqSnnXv5MiZb8XW60qFMW1RXzVWXpQSXeSugnDPuvYEFwSKwbTb3x9xW
sW0WNpyLNJqnKAkomwaiPimCX/V00sNWE7c/SSGGPjEGk7UQx4xMcUwAf0QoPRXO
dPeTQ6R1CtuSbZpa40ytQ0ywbxQJ0AKteH4anEt6AwNuKNFmD4/Et19hB4PcCeLG
VHUjqjuTFCK8BEnWgq4fPpuAtCQjhXOY9Ynnwq+P88/7iIs14z/QS+5gclmYQ/R2
5mof/v7xBE1Ol/cU8KHdbdYPzLUgxfX5ewZnRx5hjMGzAPmjhAzuKnVnIHuuO7BY
q9JXagH1ZFYO1vT6T/R5+TNmaaVdZfU2vL231CI+EZ6Q6191NTWzdBQn7Ly+wpEU
/axyVZ6NQ4uQuwowX1otcVwcjVCsZvV5ZVTgq8hWXXWTKtrOpb70Vow/fdxy0XwC
AhBCCtBYFqscyzNr8j33QvYHF2cpCf25AY0EZTOflgEMAMX+DH0JqK3L6WKaNml1
YOf2v5D0rByQySaRfzZ5nAkmLHjwUXuok1WGh/23ZX5GIHxPlhcfaAYJflzWe1z6
U8EEetw8g0mgzZ6+sEFrs6i74vlm36Cz0zJ+YxDiaxy+dbPCTOHJg8ZR74wEhc8y
IaCeKnNHX5oK15cBFRBtYKVgs4WBLE6bg2Yt0eE8kUDhReiWvoPczxZg9bBdGdXI
Z0lgIBe+F6fIP6UW7jEQN71O3K83mNbb97Mb13DW6hjhw6OGJnCj+R6Z0+paOL0G
kPV8gTpkTtJhXSBTiG6Br7uXlB8DWR2GDMcKwr2QjKmIisrQuQvQT9cUgFCLSXf2
OcKlUQwWHA+8VTVw9s/HgF4ghfegsidmesDqtg8IUnVUbXhVl3Ta4nGoOgKf4um3
AlP2wcR0l1eD9wQtsczqUssjhfqYr56JI/hAkgm//aW4rpy1Hxd3APJhRPm2y2w6
sXhwvmzEabV1qU7fqX9hXP8cyRCzLmVi7jj55Xqe4GbnEQARAQABiQG2BBgBCgAg
FiEE0f1tfx1zAGfoCYF92EAZKbTID1UFAmUzn5YCGwwACgkQ2EAZKbTID1VqWgv/
T310IhZzRX9lzo9UgpJmoiCo5F9b4JCXf1S9O2srIRBjHzzoikWmqRUpHK1iTYnr
bODFdyW20+l/b60vzWjfotgZM7oFtHK8TNNhzx/p8judAsp6RLffPz+c7c92eOZj
4UoeQGndvIZM8cDyNEPj9VKLI2Tnk6vAgcR/fEkrGFinMCKwJ1s7Ct6QNZAPhy3+
1xgHw5hsQARv9V/xEGP14CkPvjW5m2HdFUiHL602xQ2VTzVeV3VW/Pl3sVnbMq4N
4zNpHWNFeqHKyjs2UhXPWF41qoC7/EQDFYdFqyVqT0CVMl6sD6CGmGrklm3HnDgQ
OgeSoUOOEbyg3q01XrR5mHzubDFt8nY7EA7+AEdSxXBXbHxx7MPtqZGlYUg1dQLi
l7R7mQHykxKBwWRFqn5RoQFO+xIc0jYt+agKpZ86k+GbXxPcNsYPfs7lzyZcS3c6
AACd6gaZGLO6Z3T+pdElVvvqqG247tpLJDmYSyRC2T/544e25VDnhiwH8ntQZlMU
=yyqJ
-----END PGP PUBLIC KEY BLOCK-----
```