PPLNS with Job Declaration

Lorenzo Bonazzi, Filippo Merli

August 28, 2024

1 Introduction

One of the biggest bitcoin issue is the centralization of transaction selection. Since in future transactions fees will become the largest percentage of the block reward, transaction selection will be more and more relevant for the miners payout. This issue is solved by Stratum v2 [5], which enables a miner to select their own block template and just declare to the pool the job they are mining on. This feature is known as *Job Declaration*. On the other hand, current payout schemes for mining pools account only for provided hash-rate, which is not ideal for Job Declaration. Given that, the network needs a system that calculates payouts based both on selected transaction fees and hash-rate provided.

We propose a new payment schema, called PPLNS-JD, that expands PPLNS (see [4] and [3]), so that miners will get paid in a fair way given the hash power provided, and the fees in the mined block template. The core idea is that a miner performing Job Declaration who is more capable to find the best set of transactions in terms of fee over vbytes (and consequentially in fees of the block template), should be rewarded more. PPLNS-JD splits the payout of each share in two parts. The first comes from the block subsidy and pays accordingly to the difficulty-based score of the share, in the same fashion as traditional PPLNS. The second comes from the block fee and pays accordingly to the share fee-based score in such a way that share with the highest fee in its group of shares (slice, see below) is paid at least F_S/n, where F_S is the part of the block fee reserved to that group of shares S and n is the number of shares in S (see Lemma 3.2.2).

PPLNS-JD can be implemented as an extension of Sv2 [2] and it is easily accountable, so that miners' trust in pool operator is minimized.

This article is available here [6].

2 Slices

In order to pay fairly, with respect to fees, the shares produced by a miner that performs Job Declartion, they must be scored somehow and the block fee must be redistributed accordingly. Doing so, it is necessary to divide them into groups and compare scores only for shares belonging to the same group. Comparing two shares taken randomly in a traditional PPLNS window is inaccurate, because the mempool maximum extractable fees (MMEF) may change consistently. The same is true for randomly selected shares within a mining round.

We call a group of shares whose fees can be scored (and the scores can be compared) a *slice*, which is a portion of the stream of shares. A slice must not cross two mining rounds and therefore all shares in one slice must have the same prev-hash. It is needed an introduction before giving the precise definition of a slice.

Note that the only way to remove transactions from mempool is to have them mined. Therefore, MMEF grows monotonically within a single mining round. If a slice consists of *all* the

shares with the same prev-hash (namely the slice consists of the shares produced between two consecutive blocks), there are some incentives for miners to performs a variation of pool hopping (see Appendix).

To remove this incentive, slices must be shorter than the entire mining round. Intuitively, they must be short enough to be possible for the fees to be approximated as constant. Since a slice ends when a new slice begins, it is enough define when a slice starts. The slices are created by the pool and they are used to calculate the the fee-based score.

To each slice is associated a *reference job* and the fees f of that job. Let be $\delta > 0$ chosen by the pool. Then, every share that belong to the slices relative to the reference job must have fees lower than

$$f + \delta.$$

A new slice is created and the reference job for the slice is updated when the pool receives a share in which:

1. fees are greater than or equal to $f + \delta$ (increased MMEF).
2. have a prev-hash different from the actual slice reference job (new block).

Every share received by the pool that do not have a job that meet condition 1. or 2. is put in the actual slice.

If there are t slices in the window and F are the collected fees of the mined block, assign to each slice S an amount $F_S = F/t$. Then, all the shares will be scored basing on the fees of selected transactions and F_S will be redistributed to all the shares of S accordingly with its fee score.

3 Scores

In PPLNS, once that some pool's miner finds a block, every share in the window will be scored based on difficulty and the full block reward (block subsidy and block fees) will be redistributed accordingly. This scoring assumes that all miners are mining on the same job provided by the pool, which performs transaction selection in a centralized way. With the addition of Job Declaration, this type of scoring becomes unfair. Given two miners with the same hash-rate that submit the same amount of work to the pool, but each has different block templates via Job Declaration. The first miner optimizes transaction selection for maximum fees, and the second not so much. With traditional PPLNS, these two miners are paid equally since only hash-rate (work submitted via shares) is considered. This is unfair as the miner with optimized transaction fees is working toward higher block reward for the pool, and should be rewarded commensurately.

To solve this inequity, we split the payout of each share and redistribute the block fee and the block subsidy separately. Since the last is independent of transactions in the block, we will distribute it independently from Job Declaration, and therefore will be redistributed in base of the difficulty score $score_d(s_i)$ of the i-th share s_i. So, the block subsidy will be redistributed in the same fashion as standard PPLNS.
For the block fees, we introduce a new type of scoring, which is based on difficulty and fees (we must not lose the dependency on difficulty, because in the future miners are supposed to be paid mainly with fees). So, the block fees are supposed to be redistributed among miners based on a score $score_d(s_i)$ that depends both difficulty and the fees of every share.

In the final subsection precise definition for the payout of s_i is given.

3.1 Difficulty-based score

If the window contains N shares, then for the i-th share s_i we define its score

$$score_d(s_i) = \frac{d_i}{\sum_{j=1}^{N} d_j},$$

where d_j is the difficulty of the share s_j. Note that this scoring depends only on difficulty and that all scores adds up to 1:

$$\sum_{i=1}^{N} score_d(s_i) = 1,$$

this means that each score reflect it contribution on the total. It is important to mention that N ia not specified, but is suggested to be such that the sum at the denominator is a multiple the bitcoin difficulty (for Ocean TIDES [3] this multiple is 8).

Remark 3.1.1. *It is easy to see that difficulty-based score is linear. If there is $c > 0$ such that $d_i = cd_j$ for some s_i, s_j, then $score_d(s_i) = c \cdot score_d(s_j)$. So, miner's shares are scored (and paid) proportionally with respect his hash-rate.*

3.2 Fee-based score

As mentioned above, it is important that two shares with the same fee also account for computational work needed. Without this, they would be paid equally, unfairly benefiting the one that required less computational effort to be produced. Constructing the fee-based score on top of difficulty-based score fixes the issue. Indeed, the score that we are going to introduce is linear with respect to difficulty and fees (see Remark 3.2.1).
Suppose we want to score the i-th share s_i, and suppose that this share belongs to a slice S that contains all the shares from the k_1-th to the k_2-th. For the j-th share, $\bar{d}_j = score_d(s_j)$ is the difficulty-based score (calculated as above) and f_j is the fee in the block header of that share (which depends on the transactions chosen in the template). We define the fee-based score

$$score_f(s_i) = \frac{\bar{d}_i f_i}{\sum_{j=k_1}^{k_2} \bar{d}_j f_j}.$$

Similarly to difficulty score, if the slice S contains shares from k_1-th to k_2-th, we have

$$\sum_{i=k_1}^{k_2} score_f(s_i) = 1.$$

Remark 3.2.1. *Within a slice, it is easy to see that the fee-based score is linear with respect both difficulty and fees.*

1. *If there is $c > 0$ such that $f_i = c \cdot f_j$ for two shares s_i, s_j in the slice S, then $score_f(s_i) = c \cdot score_f(s_j)$. This means that if a miner becomes c times more capable of finding a more profitable template, his shares will be scored (and paid) proportionally.*
2. *Similarly to Remark 3.1.1, if $\bar{d}_i = c\bar{d}_j$, then $score_f(s_i) = c \cdot score_f(s_j)$.*

Lemma 3.2.2. *Suppose that the slice S contains n shares, all of the same difficulty score. Then, if $f_i = f_{max}$ is the max fee of every share in the slice S, then the payout for this share is at least $score_f(s_i) \geq 1/n$.*

Proof. The Lemma is proven by contradiction. So, assume that

$$score_f(s_i) < 1/n.$$

Since all the shares have the same difficulty score, we have

$$\frac{\bar{d}_i f_{max}}{\sum_j d_j f_j} = \frac{f_{max}}{\sum_j f_j}.$$

It follows that

$$\frac{f_{max}}{\sum_j f_j} < \frac{1}{n}.$$

So,

$$1 = \frac{\sum_k f_k}{\sum_j f_j} = \sum_k \frac{f_k}{\sum_j f_j} \leq \sum_k \frac{f_{max}}{\sum_j f_j} < \sum_k \frac{1}{n} = 1$$

which is impossible. □

3.3 Payout for each share

Suppose that we want to pay the i-th share. This share will belong to a slice S, to which is reserved a portion of block fees subsidy F_S. Then the payout is

$$payout(s_i) = r \cdot score_d(s_i) + F_S \cdot score_f(s_i).$$

The redistribution of block subsidy necessarily depends only on d_i it is independent from work selection, because it is guaranteed even in empty blocks.

For a share s_i, call $payout_d(s_i) = r \cdot score_d(s_i)$ and $payout_f(s_i) = F_S \cdot score_f(s_i)$. Then we have the following important results, which guarantee that the share payout is proportional with respect of both fees and difficulties. The proof of the next Remark is a direct consequence of Remarks 3.2.1 and 3.1.1.

Remark 3.3.1. *Let s_i and s_j two shares. Let d_i, d_j be their difficulties and f_i, f_j be their fees.*

1. *If there is $c > 0$ such that $d_j = c \cdot d_j$. Then $payout_d(s_i) = c \cdot payout_d(s_j)$. If the two shares are in the same slice and $f_i = f_j$, then $payout(s_i) = c \cdot payout(s_j)$.*
2. *If the share are in the same slice, $d_i = d_j$ and there is $c > 0$ such that $f_i = c \cdot f_j$, then $payout_f(s_i) = c \cdot payout_f(s_j)$*

We can see that the payouts of all shares add up to the right amount. Suppose that N is the size of PPLNS window and that the slices are $S_1, \ldots, S_t$, and suppose that there are $0 = k_0 < k_1 < k_2 < \cdots < k_t = N$ such that the m-th slice contains shares $s_{k_{m-1}+1}, \ldots, s_{k_m}$. Note also that $F_S = F/t$. So, using the fact that scores add up to 1 :

$$\begin{aligned}
\sum_{i=1}^{N} payout(s_i) &= r \sum_{i=1}^{N} score_d(s_i) + F_S \left(\sum_{j=k_0+1}^{k_1} score_f(s_j) + \cdots + \sum_{j=k_{t-1}+1}^{k_t} score_f(s_j) \right) \\
&= r + F_S \cdot t \\
&= r + F \\
&= R.
\end{aligned}$$

Hence, all the payouts add up to the total of funds available.

4 Shares' accountability

A pool that uses this accounting schema is expected, for each block found, to publish all the slices that belong to that block window calculated using PPLNS. The published slice will contain:

- The number of shares contained in the slice
- The sum of the difficulty of the shares
- Fees of the slice's reference job
- Merkle root of the tree composed by all the shares that belong to the slice
- Id of the reference job

A miner must be sure that all the shares received by the pool are valid. If the pool is also a miner, there are some incentives to pay more itself than other miners, debasing their share value. A method for doing so is to produce fake shares and pay them as if they were valid. To avoid this, a miner can *challenge* the pool:

1. randomly select slice(s) to check; for each slice randomly select share(s) from within the slice
2. for each selected share, fetch the job and transactions that are not in the cache
3. verify that each share is valid
4. verify the Merkle tree for the slice with provided Merkle path
5. verify that the sum of (verified) shares' difficulties is not bigger than the slice difficulty
6. verify that the fees in the shares are lower than fees of the slice ref job fees + delta

Whenever a miner sends a share to the pool, it answers with the reference job, so the miner can verify if the ref job fees + delta is higher than the share's job fees.

Ackowledgments

We would like to express our gratitude to Demand [1] for their collaboration throughout the development of this project. Additionally, we extend our thanks to the developers of the open-source project *Stratum v2 Reference Implementation* ([5]), whose contributions were crucial to the success of the Sv2 project. We would like to express our thanks also to Gary and Bernardo (@plebhash) for their thorough review of the manuscript and for providing numerous valuable suggestions.

A Motivation for slices

In this appendix we show that if a slice is large enough, then there may be some incentives for a miner to perform a variation of pool hopping (see [4] for a definition of pool hopping). For doing so, assume the following two pools:

1. POOL-1 implements a classic PPLNS.

2. POOL-2 adopts PPLNS with Job Declaration, but with the slices that coincide with the rounds of blocks mining.

In POOL-2, if ℓ is a prev-hash, all the share with ℓ as prev-hash consists of a single slice.

We calculate the payout per share of a group of miners, assuming first that they mine for the POOL-1, and subsequently like if they were mining for the POOL-2. Then we will compare the results of these two calculations. Without loss of generality, we make some assumptions:

- bitcoin difficulty is constant and all the miners have the same hash-rate.
- both the miners produce 100 share per minute and once a block is found, the payout goes back 8 blocks. Then we calculate the windows size for the two pools: $N = 100 shares/min \cdot 8 \cdot 10 min = 8000$.
- the pool fee is zero and the block reward coming from the fees of mined transactions is F.

Since redistribution of block subsidy is fixed with both methods, we assume it to be zero and we consider only the redistribution of block fees F. Let $payout_f(s_i)$ be the payout of share s_i.

Question: Suppose that we want to pay the i-th share s_i. How much is paid in PPLNS-JD with full mining round sized slices compared with PPLNS? With standard PPLNS, we have that

$$payout(s_i) = \frac{F}{8000}.$$

With PPLNS-JD with large slices, on the other hand, we have

$$payout_f(s_i) = F_s \cdot score_f(s_i),$$

where F_S is part of the fee reward reserved for the slice to which the share belong (and that contains all the shares with a specific prev-hash). Since we have 8 block, we have that $F_S = F/8$. So,

$$score_f(s_i) = \frac{\bar{d}_i f_i}{\sum_{j=1}^{100} \bar{d}_j f_j} = \frac{f_i}{\sum_{j=1}^{100} f_j}.$$

Recall that all the miners have the same hash-rate.

Note. From now on we assume that the growth of fees within a slice (which coincides with a mining round) is linear, namely there are $m > 0$ and $c > c$ such that

$$F(f) = mt + c.$$

Furthermore, suppose that there are 100 shares received by the pool in the time of the block to which belong to share s_i (that coincides with the time frame of the slice S). Without loss of generality, we can assume that the distribution of these shares in this time frame is uniform. For a share s_j in S, we have that

$$f_j = m(6secs \cdot j) + c,$$

so

$$\begin{aligned} score_f(is_i) &= \frac{m(6secs \cdot i) + c}{\sum_{j=1}^{100} m(6secs \cdot j) + c} \\ &= \frac{m(6 \cdot i) + c}{m \cdot 6 \cdot \frac{100 \cdot 101}{2} + c \cdot 100} \\ &= \frac{m(6 \cdot i) + c}{100(303 \cdot m + c)} \end{aligned}$$

and

$$payout_f(s_i) = \frac{F}{8} \cdot \frac{m(6 \cdot i) + c}{100(303 \cdot m + c)}.$$

Hence, there must be a $\bar{i}$ such that $payout_f(s_{\bar{i}-1}) < F/8000 < payout_f(s_{\bar{i}})$. Therefore, for a miner it is more profitable to mine with POOL-1 until $\bar{i} \cdot 6secs$ and then join POOL-2. This is a slight variation of classic pool hopping, in which the miners jump into a pool that has just found a block and mine there for a while (this is quantified in [4]). In our case, miners will jump into this pool at the end of the mining round. This form of pool hopping is not desirable. This led us to the introduction of slices. Within a slice, MMEF can be approximate as flat, so it is fair to compare a shares' fees. It is worth pointing out that in this form of PPLNS-JD, in which the slices consists of all the shares with the same prev-hash (therefore produced within a single round of mining), a miner that remains at the beginning of the round may be disadvantaged. Indeed, at the beginning his shares worth less because of the low fees (that grow linearly with time). Then, after $\bar{i} \cdot 6secs$, when supposedly the fees are higher and his shares may worth more, many other hopping miners join the pool, rising the pool hash-rate. This will make the shares of the faithful miner to compete against many more shares, and therefore mitigating the relief of higher fees. In conclusion, there is a double disincentive for miner to be faithful to the POOL-2.

B Notations

- *share* or weak block: is a block with difficulty lower than the bitcoin's one. Miners send shares to the pool periodically for proving that they are actually mining. Each share needs some work to be produced and will be paid by the pool proportionally by its difficulty.
- *job*: the transaction list the miners are mining on.
- PPLNS *window*: the shares that are paid once that a block is found by some pool's miner. So, the windows size is N, where N is symbol used for the acronym "Pay Per Last N Shares".
- *mining round*: the shares produced in the time between a Bitcoin block and the following one or, equivalently, all the shares *with the same prev-hash*. block fees: the sum of the fees of all transactions in the block.
- *block subsidy*: the predetermined supply issuance of bitcoin in a block, this value halves every 210,000 blocks, the initial subsidy was 50 bitcoin per block.
- *block reward*: block subsidy + block fees.
- *hash-rate*: the amount of hashes per second a miner is capable to produce. Measured in H/s.

References

[1] Demand pool `https://www.dmnd.work/`

[2] Extension on share accounting `https://github.com/demand-open-source/share-accounting-ext/blob/master/extension.md`.

[3] Ocean pool `https://ocean.xyz/docs/tides`

[4] *Analysis of Bitcoin Pooled Mining Reward Systems*, M. Rosenfeld, https://arxiv.org/abs/1112.4980

[5] Stratum v2. Specifications: https://github.com/stratum-mining/sv2-spec/ and Reference Implementation https://github.com/stratum-mining/stratum/

[6] https://lorbax.github.io/pplns-with-job-declaration/pplns-with-job-declaration.pdf and https://www.dmnd.work/pplns-with-job-declaration/pplns-with-job-declaration.pdf