PPLNS with Job Declaration

Lorenzo Bonazzi, Filippo Merli

August 29, 2024

1 Introduction

Right now, payout schemes for miners account only for provided hash power. One of the biggest bitcoin issue is the centralization of transaction selection, this issue is solved by Stratum v2 [5]. Given that Stratum v2 let the miner select their own transaction (this feature is known as *Job Declaration*), and that transactions fees might become the most important part of the mining reward, we need a system that calculate the payouts based both on selected transactions and hash power provided. The system proposed expands PPLNS (see [4] and [3]), so that miner will get paid in a fair way given the hash power provided, and the fees in the mined job. The core idea is that a miner perfoming 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. The system proposed can be implemented as an extension of Sv2 [2] and it is easly accountable, so that miners' trust in pool operator is minimized.

We define a fair payment schema that splits the share payout in two parts. The first comes from the coinbase and pays accordingly to the difficulty score of the share, in the same fashion as PPLNS. The second comes from the block fee and pays accordigly to share difficulty score and the share fee 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.

2 Slices and windows

In order to pay shares in a fair way with respect to fees, they must be scored somehow and the payout of the block fee must be redistributed accodingly to this score. Doing so, it is necessary to divide them in groups and compare scores only for shares belonging to the same group. Indeed, does not make any sense compare two shares taken randomly in the PPLNS window, because the mempool maximum extractable fees (MMEF) may change consistently. Indeed, it is difficult even to compare shares produced between a block and the following (we call this 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. As we seen in the previous paragraph, a slice must not cross two mining rounds. With other words, all shares in one slice must

have the same prevhash. It is needed an introduction before giving the precise definition of a slice.

Note that within a mining round, MMEF can not decrease. Indeed, the only way to remove high fees 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 prevhash (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 starts the following one, it is enough define when a slice begins. Note that the slices are created by the pool and recall that they are used to calculate the score based on fees. To each slice is associated a reference job. Fix f the fees of a particular reference job. Let $\delta > 0$ 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 if one of the two event happen:

1. the pool receive a share for a job that have fees bigger or equal then $f + \delta$, where f the actual slice reference job fee. The new slice reference job is this job.

2. the pool receive a share for a job that have a prevhash different from the actual slice reference job. The new slice reference job is this job.

Every share received by the pool that do not have a job that meet condition 1. or 2. is put in the actual slice. The above means that we create a new slice every time that the MMEF change: 1. is when MMEF grows and 2. is when a new block have been found and MMEF is lower or equal.

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 windows will be scored based on difficulty and the full block reward (coinbase and block fees) will be redistibuted accordingly. This is the ground idea of PPLNS and it is based on the fact that the work selection is performed by the pool, so all the miners are mining on the same job. If we add the Job Declaration feature, this type of scoring is not anymore fair. Indeed, suppose that there are two minewrs with the same hashpower, and both perform Job Declaration. The first is very effective in doing that, and the second not so much. So, the shares produced by the first will have higher fees than the second. On the

other hand, the difficulty of the shares of the two miners will be equal, so the payout. This is not fair, because the first miner should be paid a bit more than the second.

In this section we aim to solve this issue. The idea of our proposal is to split the payout of each share, and redistribute the coinbase reward and the block fee separately. Since the coinbase reward is independend on the other transactions in the block, it is also independent 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 coinbase reward will be redistributed in the same fashion as standard PPLNS.
For the block difficulty, we must introduce a method to weight shares based on the fees. We must neither lose the dependency on the difficulty, because in the future miners are supposed to be paid mainly with fees. So, the block fees are supposed to be redistributed amongs miners based on a score $score_d(s_i)$ that depends both difficulty and the fees of every share.

In the final subsection it will be given the precise definition for the payout of s_i.

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 (for example, [3] suggest to adopt N such that the sum at the denominator eight times the bitcoin difficulty).
Remark.

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

and that this score depends only on difficulty.

3.2 Fee-based score

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 i-th share, call $\bar{d}_i = score_d(s_i)$ its score difficulty (calculated as above) and f_i the fees of the share (which depends on the transactions chosen in the template). For this share, define the score relative to the fee

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

Remarks.

1. As remaked earlier, the dependence on difficulty is intentional. Otherwise, if there are two shares with the same fee, the one with lower difficulty is paid the same as the other, while requiring less computational effort to be produced.

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 $score_f(s_i) \geq 1/n$. This sentence is proven by contraddiction. 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. if there is $c > 0$ such that $f_i = cf_j$ for some s_i, s_j two shares in the slice S, then $payout_f(s_i) = c \cdot payout_f(s_j)$. This means that if a miner becomes c times more capable of finding a more profitable template, his shares will be paid proportionally.

4. similarly, if $\bar{d_i} = c\bar{d_j}$, then $payout_f(s_i) = c \cdot payout_f(s_j)$ this similarly represents the fact that a miner is paid proportionally with respect his hashpower.

5. similarly to difficulty score, if the slice S contains shares from k_1-th to k_2-th, we have
$$\sum_{i=k_1}^{k_2} score_f(s_i) = 1.$$

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 the coinbase reward necessarly depend only on d_i, because it is guaranteed even in empty blocks, so it is independent from work selection. We can see that the payouts of all shares add up to the right amount. Suppose that the slices are $S_1, \ldots, S_t$, and suppose that there are $1 = k_1 \leq k_2 \leq \cdots \leq k_t$ such that the m-th slice

contains shares $s_{k_m} \dots s_{k_{m+1}}$. Note also that $F_S = F/t$. So:

$$\begin{aligned}\sum_{i=1}^{N} payout(s_i) &= r \sum_{i=1}^{N} score_d(s_i) + F_S \left(\sum_{j=k_1}^{k_2} score_f(s_j) + \dots + \sum_{j=k_{t-1}}^{k_t} score_f(s_j) \right) \\ &= r + F_S \cdot t \\ &= r + F \\ &= R\end{aligned}$$

Hence, we can see that all the payouts add up to the total of funds available.

4 Shares' accountability

A pool that uses this payment 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 contains infos about:

- The number of shares contained in the slice
- The sum of the difficulty of the contained 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 arrived to the pool are valid. Indeed, if the pool is also a miner, there are some incentives to pay more itself then other miners. A method for doing so is to produce fake shares and pay them as they were valid. For avoiding that that, a miner can *challenge* the pool:

1. randomly select some slice that he is willing to check; for each selected slice randomly select some share in the slice
2. fetch the job and the transactions that are not in the cache for each selected share
3. verify that each share is valid
4. verify that merkle path of each share + share hash = root in the Slice
5. verify that the sum of the difficulty verified shares is not bigger then 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, the pool answers with the actual reference job so the miner can verify if the ref job fees + delta fees are higher then the share's job fees.

If this system is implemented as an Sv2 extension, will be very easy for a miner to prove that a pool is cheating, since Sv2 is authenticated the miner have only to publish the Sv2 session with the handshake messages.

5 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.

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, we suppose that there are 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 prevhash, all the share that have ℓ as prevhash consists of a single slice.

We calculate the payout per share of a group of miners, assuming firstly 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 hashrate.
- the miners produce 100 share per minute and once a block is found, the payout goes back 8 blocks. Then we can caluculate 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 coinbase reweard is fixed with both methods, we assume it to be zero and we consider only the redistribution of block fees subsidy F. Let $payout_f(s_i)$ be the payout of the 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 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 belong the share (and that contains all the share to a specific prevhash). 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 hashpower.

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

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

Suppose furthermore that there are 100 shares received by the pool in the time of the block to which belong the share s_i (that coincides with the time frame of the slice S). It is no loss to assume that the distribution of these shares in this timeframe 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. Nevertheless,

even this form of pool hopping is not tolerable. This led us to the introduction of slices. Within a slice, MMEF can be approximized as flat, so it is fair to compare a shares' fees. It is worth pointing out that in PPLNS-JD in which the slices consists of all the shares with the same prevhash (therefore produced wihtin 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 hashrate. 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.

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 5https://github.com/stratum-mining/stratum/