Stake Pool Ranking in Cardano

Alexander Byaly
alexander.byaly@iohk.io

Jared Corduan
jared.corduan@iohk.io

May 5, 2022

1 Introduction

The purpose of the stake pool ranking algorithm is to provide a systematic and unique ordering for stake pools, based on the expected long-term rewards that each stake pool will produce. The pool ranking is provided to users (wallets, people who own ADA, etc.), and can be used as a single metric to guide delegation decisions. The ranking is based upon past performance but should be a useful predictor of future performance.

An obvious, but naïve, approach to the ranking problem would order pools based on the rewards that are currently being paid out to their delegators. There are, however, a number of problems with this approach.

1. The naïve approach is *short-sighted*, in the sense that it does not take into account the rewards that a newly formed stake pool would yield when it reaches saturation ([BKKS18] describes this in detail). The specific relationship to the Cardano design is explained in the IOHK delegation design document [SL-D1, Section 5.6].
2. A stake pool will sometimes have the opportunity to produce blocks, but fails to do so due to issues such as a poor network connection or untimely downtime. The rate at which a pool uses their opportunities to make blocks, which we will call its *hit rate*, affects rewards. It needs to be taken into account.

Our aim with the pool ranking algorithm is to address all these concerns.

2 Hit Rate

As described above, the *hit rate* for some time period T is the ratio of expected to actual blocks that the pool produces during that time period.

$$
\begin{aligned}
Hit\ Rate_T &= B_{Produced,T}/B_{Leader,T} \\
&\textit{where} \\
B_{Produced,T} &= \text{The actual number of blocks that a pool has produced in time T} \\
B_{Leader,T} &= \text{The number of blocks during time T in a pool's private leader schedule}
\end{aligned}
$$

Note that the pool operator is the only one who knows $B_{Leader,T}$, since only they have access to their internal VRF calculation. Other actors can only estimate it using public knowledge of the pool's relative stake and the number of blocks that it has produced.

2.1 Hit Rate Estimation

Each slot, a stake pool with relative (active) stake σ_a has probability

$$1 - (1 - f)^{\sigma_a}$$

of producing a block, where f is the active slot coefficient. We can use this to estimate the hit rate. We represent such an estimate as a probability distribution on the interval $[0, 1]$.

After each epoch, we refine our estimate into a more accurate one as follows. Assume that we can observe all the blocks that a stake pool produces during an epoch, that we have a previous estimate of the blocks that we expected to produce during this epoch, w_e; we know that the epoch e has a total of E slots; the pool produces n_e blocks; and we calculate that on a given slot the chance that the pool is allowed to produce a block is t_e. The *likelihood function*, L_e, is then shown below, where $L_e(x)$ is the probability of producing exactly n_e blocks, and $S_e(x)$ is the probability that the pool produces a block at a given slot, under the assumption that the *hit rate*, p, is equal to x.

$$L_e(x) = \binom{E}{n_e} S_e(x)^{n_e} (1 - S_e(x))^{E - n_e},$$

$$
\begin{aligned}
S_e(x) &= P(\text{made block} | p = x) \\
&= t_e \cdot P(\text{made block} | p = x,\ b) + (1 - t_e) \cdot P(\text{made block} | p = x,\ \neg\, b) \\
&= t_e x + (1 - t_e) \cdot 0 \\
&= t_e x
\end{aligned}
$$

$$
b = \text{can make a block}
$$

Our refined hit rate estimate is then $w_{e+1}(x) = w_e(x) L_e(x)$.

This formula will generally not produce a proper proability distribution. The associated probability distribution is

$$q_e(x) = \frac{w_e(x)}{\int w_e(x)},$$

which is the function used to construct rankings for epoch e.

The data we store is the product

$$\mathcal{L}_e(x) = \prod_{i=1}^{e} L_i(x),$$

with the initial prior distribution w_0 included at the time a ranking is produced.

We have

$$q_e(x) = \frac{w_0(x)\mathcal{L}_e(x)}{\int w_0(x)\mathcal{L}_e(x)}. \tag{1}$$

2.2 Implementation

We approximate a *probability density function*, g, using a sequence of sample points,

$$A(g) = \big(\ln(g(0.005)),\ \ln(g(0.015)),\ \ldots,\ \ln(g(0.995))\big).$$

This allows us to use multiplication rather than exponentiation for most of our calculations.

Because of the normalization step in Equation (1), any scalar multiple of g will ultimately yield the same result. It follows that any constant shift of $A(g)$ will also yield the same result. In order to reduce the magnitude of the numbers that we are dealing with, we therefore use the logarithmic normal form. [1]

$$A'(g) = \big(\ln(g(0.005)) - m,\ \ln(g(0.015)) - m,\ \ldots,\ \ln(g(0.995)) - m\big),$$

where ln is the *natural logarithm* and m is the minimal sample point. We use g^i to denote the i'th term of $A'(g)$. For each stake pool, we compute $A'(L_e)$, add it to $A'(\mathcal{L}_{e-1})$, and normalize the result to approximate $A'(\mathcal{L}_e)$.

To calculate the final probability distribution function that we need for pool ranking, we approximate Equation (1). We use

$$q_e^i \approx \mathsf{exp}(w_0^i + \mathcal{L}_e^i)/S,$$

where

$$S = \sum (0.01)\mathsf{exp}(w_0^i + \mathcal{L}_e^i).$$

In the ranking calculation, we use the j'th percentile of this distribution for some j. We do so by computing the minimal i such that the sum of $[q_e^0,\ \ldots, q_e^i]$ is at least j. Then $i/100$ is the j'th percentile.

Putting it all together, given some prior distribution, w_0, a percentile, j, and a cumulative likelihood approximation $A'(\mathcal{L}_e)$, we estimate the hit rate to be $i/100$.

2.3 Example

Assume that Alice's pool has been active for two epochs, there are 432,000 slots per epoch, and the active slot coefficient is 0.05 (these are the mainnet values at the time of writing).

- During the first epoch (epoch 1), the relative stake was 0.001 and 10 blocks were produced.
- During the second epoch (epoch 2), the relative stake was 0.01 and 200 blocks were produced.

[1] There may be tradeoffs, such as the computational cost of logs, that we have not yet explored.

We calculate $\ln(L_1(x))$ for the first epoch as follows. With relative stake 0.001, the probability of being allowed to produce a block on each slot is $t_1 = 1 - (1 - 0.05)^{0.001}$, and $S_1(x) = t_1 x$. Since 10 blocks were produced, the real likelihood function is

$$L_1(x) = \binom{432,000}{10} (t_1 x)^{10} (1 - t_1 x)^{431,990}$$

and we will use

$$\begin{aligned} \ell_1(x) &= \ln\left((t_1 x)^{10} (t_1 x)^{431,990}\right) \\ &= 10 \ln(t_1 x) + 431,990 \ln(1 - t_1 x) \end{aligned}$$

to record information about Alice's pool's performance this epoch. In particular, we record

$$A'(L_1) = [\ell_1(0.005) - m_1,\ \ell_1(0.015) - m_1,\ \ldots,\ \ell_1(.995) - m_1]$$

where $m_1 = \min\{\ell_1(0.005), \ldots, \ell_1(0.995)\}$

For the second epoch,

$$\begin{aligned} t_2 &= 1 - (1 - 0.05)^{0.01} \\ \ell_2(x) &= 200 \ln(t_2 x) + 431,800 \ln(1 - t_2 x). \end{aligned}$$

Then for the second epoch we store:

$$A'(L_2) = [\ell_2(0.005) + \ell_1(0.005) - m_2,\ \ldots,\ \ell_2(.995) + \ell_1(.995) - m_2]$$

where $m_2 = \min\{\ell_2(0.005) + \ell_1(0.005), \ldots, \ell_2(0.995) + \ell_1(0.995)\}$.

3 Ranking and Saturation

We now describe how the hit rate estimation can be used to calculate the stake pool ranking, as described in [BKKS18] and [SL-D1, Section 5.6]. First we must turn the sampling of the likelihood functions into a concrete value. Given a percentage, z, we will calculate the z'th percentile[2]. For a given stake pool P, let h_z denote the z'th percentile. Following [SL-D1,

[2] At present we use the 10th percentile (i.e. $z = 10\%$), but we ultimately aim for this to be user configurable.

Section 5.6.1], we can then compute the stake pool's rewards when it reaches *saturation* (it has the optimal level of stake) using:

$$\tilde{f}(s, \overline{p}) := \hat{f}(s, z_0, \overline{p}) = \frac{\overline{p} R}{1 + a_0} \cdot (z_0 + \min(s, z_0) \cdot a_0) \,.$$

where $\bar{p}$ is the *apparent performance* of the stake pool, as defined in [SL-D1, Section 5.5.2]. We will substitute our estimate of the hit rate, h_z, for the apparent performance, $\bar{p}$, Given a pool, P with pledged owner stake s, costs c and margin m, the *desirability function* of [SL-D1, Section 5.6.1] becomes:

$$d(c, m, s, h_z) := \begin{cases} 0 & \text{if } \tilde{f}(s, h_z) \leq c, \\ \left(\tilde{f}(s, h_z) - c\right) \cdot (1 - m) & \text{otherwise.} \end{cases}$$

From here, we proceed exactly as in [SL-D1, Section 5.6.2] and [SL-D1, Section 5.6.4].

The desirability function above can be used to provide an overall ranking over stake pools, ordering the pools in terms of their *non-myopic rewards* for some stake. For a stake pool with pledged owner stake s, total stake σ, rank r and member stake t, we define the *non-myopic stake* σ_{nm} as:

$$\sigma_{\text{nm}}(s, \sigma, t, r) := \begin{cases} \max(\sigma + t, z_0) & \text{if } r \leq k, \\ s + t & \text{otherwise,} \end{cases}$$

where k is the parameter that is defined in [SL-D1, Section 5.2], and which represents the desired number of stake pools in an equilibrium state. The *non-myopic pool member reward* for a member that contributes member stake t to a stake pool with hit-rate estimate h_z, is then:

$$r_{\text{member,nm}}(c, m, s, \sigma, t, r, h_z) := r_{\text{member}}\Big(\hat{f}\big(s, \sigma_{\text{nm}}(s, \sigma, t, r), h_z\big), c, m, t, \sigma_{\text{nm}}(s, \sigma, t, r)\Big),$$

where

$$r_{\text{member}}(\hat{f}, c, m, t, \sigma) := \begin{cases} 0 & \text{if } \hat{f} \leq c, \\ (\hat{f} - c) \cdot (1 - m) \cdot \dfrac{t}{\sigma} & \text{otherwise.} \end{cases}$$

Delegators can use $r_{\text{member,nm}}$ to select a pool that maximizes their rewards.

References

[BKKS18] L. Bruenjes, A. Kiayias, E. Koutsoupias, and A.-P. Stouka. Reward sharing schemes for stake pools. Computer Science and Game Theory (cs.GT) arXiv:1807.11218, 2018.

[SL-D1] IOHK Formal Methods Team. Design Specification for Delegation and Incentives in Cardano, IOHK Deliverable SL-D1, 2018. URL https://hydra.iohk.io/job/Cardano/cardano-ledger/delegationDesignSpec/latest/download-by-type/doc-pdf/delegation_design_spec.