1 Dynamic Staking Algorithm (DSA)

Multiple custodial and non-custodial staking solutions exist with their advantages and inconvenients, every methods can be susceptible to hacks or exploits; however, custodial solution represents a higher risk, rather than users being in control of their assets a small group or a single entity is in charge exposing all assets to higher attack surface.

The main invonvenience of non-custodial solution such as autonomous smart-contract is the trade-off between gas consumption and the reward algorithm. Knowing that EVM is a state machine, implemting such algorithms will require saving every action done by a user (without using any array to optimize gas consumption) while taking into account other users actions to solve one of the major inconvenience of smart contract staking algorithms.

Please note that from here on we assume that the staking purpose is to generate reward for the users, other's can exist but for the sake of simplicity we will use the most common ones (reward or dividend).

The biggest chalenge to over come is that users actions are dependent to one another meaning that if a user A stake, add to his stake or widthraw, the reward of user B will be affected. The main goal is to achieve a linear reward computaion for each user as demonstrated in equation 1.

We define $R^j_{K,N}$ as the reward of user j between block K and N.

$$R^j_{K,N} = R^j_{K,M} + R^j_{M,N} \tag{1}$$

where $K < M < N$.

2 Mathematical Proof

2.1 Example Use Cases

The goal of this use case implementation of DSA is to achieve a a reward distribution by block, where a reward is distributed every block between stakers following their stake value, it can be seen as an inflationary token with a monetary policy that requires users to stake if they want to get part of the newly minted tokens.

Other use cases can be implemented like:

- Future PoS pools, where stakers can make multiple deposits, without need to use an array to save their actions to compute the reward later on.
- A lending platform that distibute dividends using the proposed optimized algorithm, etc ...

More research are needed to implement a broader library. Refer to Table 2.1 for more details, another example is the deployement of future PoS dynamic pool where the reward can be disributed fairly without usage or arrays.

Table 2.1: Reward distribution example.

i	0	1	2	3	4	5	$R^j_{0,4}$
$R^0_{i,i}$	0	120	40	20	0	0	180
$R^1_{i,i}$	0	0	80	40	48	60	228
$R^2_{i,i}$	0	0	0	60	72	60	192

i is the block number

$R^j_{i,i}$ is the reward of user j at block i

$R^j_{0,4}$ is the reward sum between block $i = 0$ and $i = 4$ of user j

We suppose that at each block a reward is distributed, where BR_i for $i = 0, ..., 5$ is equal to 120 (the value of the reward per block can be dynamic following a precise monetary policy, we used a fixed reward for simplification only) tokens with a chronological order of events as follow:

- $user_0$ stake 100 tokens at block 1
- $user_1$ stake 200 tokens at block 2
- $user_2$ stake 300 tokens at block 3
- $user_0$ widthraw 100 tokens at block 4
- $user_1$ stake 100 tokens at block 5

Obtaining the results presented in Table 2.1 in a custodial or centralized solution is easy since there is no gas fee or gas block limit, however, when implementing the same algorithm in a smart contract the task is a way harder since it is nearly impossible to compute and save the reward for each user at every block due to high gas consumption when dealing with arrays (if a single user stake at a given block, the reward for all users will change).

The following subsection shows how to solve this problem by removing the usage of arrays of user interacations when computing the reward.

2.2 Demonstration

We define $R^j_{K,N}$ as the reward of user j between block K and N.

$$R^j_{K,N} = \sum_{i=K}^{i=N} A^j_i * \frac{BR_i}{S_i} \tag{2}$$

where:

- BR_i is the staking reward at block i to be devided between the stakers
- S_i is the total staked amount at block i for the total number of user P

$$S_i = \sum_{j=0}^{j=P} A^j_i \tag{3}$$

- A_i^j is the amount staked by a user j at block i

If we suppose that no staking or withdrawing activity was done between block K and M, A_i^j and S_i will remain constant on every block i where $i = K, K+1, ..., M-1, M$.

The new formulation of equation 2 will be as follow:

$$R_{K,L}^j = \frac{A^j}{S} * \sum_{i=K}^{i=L} BR_i \tag{4}$$

if we assume that any user x started staking at block M, the reward of user j where $j \neq x$ will be:

$$R_{K,N}^j = R_{K,M}^j + R_{M,N}^j \tag{5}$$

$$R_{K,N}^j = A^j * (\frac{1}{S_{K,M}} * \sum_{i=K}^{i=M} BR_i + \frac{1}{S_{M,N}} * \sum_{i=M}^{i=L} BR_i) \tag{6}$$

If we define the weighted block reward (WBR) as follow:

$$WBR_{K,M} = \frac{1}{S_{K,M}} * \sum_{i=K}^{i=L} BR_i \tag{7}$$

Equation 6 will become:

$$R_{K,N}^j = A^j * (WBR_{K,M} + WBR_{M,N}) \tag{8}$$

2.3 Algorithm

As demonstrated in equation 8, a user j reward between two blocks *(M,N)* is the sum of WBR between the same blocks multiplied by the user stake.

When a user start staking at block K we save the total sum $WBR_{0,K}$ for the specific user, once he claims, stake or withdraw at block N, we substruct the sum of $WBR_{0,N}$ from the initial sum of $WBR_{0,K}$. hence getting the reward of user j, $R^j = A^j * WBR_{K,N}$ (for a detailed definition please refer to algorithm 1).

3 Conclusion

A beta solidity implementation of this algorithm can be found here, all the conducted tests satisfied the expected results in Table 2.1, the mathematical demonstration was proved experimentaly.

To execute the tests please follow the steps described in the repository README.

Algorithm 1 Dynamic Staking Algorithm

global variables
R^j, reward of user j
A^j, stake of user j
S, total staked amount
WBR, weighted block reward
WBR^j, last weighted block reward for user j
LB, last block number where any modification was performed
BR, 120 tokens reward to distribute on each block
end global variables
procedure STAKE($value$, i, j)
$BR_{LB,i}$, sum of the reward between block LB and i
$BR_{LB,i} = (i - LB) * BR$
if $S \neq 0$ **then**
$WBR = WBR + \frac{BR_{LB,i}}{S}$
end if
$R^j = R^j + A^j * (WBR - WBR^j)$
$WBR^j = WBR$
$A^j = A^j + value$
$S = S + value$
$LB = i$
end procedure
procedure WITHDRAW($value$, i, j)
$BR_{LB,i}$, sum of the reward between block LB and i
$BR_{LB,i} = (i - LB) * BR$
if $S \neq 0$ **then**
$WBR = WBR + \frac{BR_{LB,i}}{S}$
end if
$R^j = R^j + A^j * (WBR - WBR^j)$
$WBR^j = WBR$
$A^j = A^j - value$
$transfer(j, value)$
$S = S - value$
$LB = i$
end procedure
procedure CLAIM(i, j)
$BR_{LB,i}$, sum of the reward between block LB and i
$BR_{LB,i} = (i - LB) * BR$
if $S \neq 0$ **then**
$WBR = WBR + \frac{BR_{LB,i}}{S}$
end if
$R^j = R^j + A^j * (WBR - WBR^j)$
$WBR^j = WBR$
$LB = i$
$transfer(j, R^j)$
$R^j = 0$
end procedure