Invariant Staker

March 17, 2022

1 Introduction

Staking Invariant provides with passive income generated by staked tokens, which are held in liquidity pools managed by Invariant. Users are free to stake as much as they desire. In the lack of a ticks range and concentrated liquidity, the total amount of staked liquidity is straightforward to compute in the traditional model. The liquidity mining rewards may be distributed without any technical difficulty. However, the ticks range is indeterminant in a concentrated liquidity model, determining active staked liquidity is technically difficult.

A user can stake their position to get a reward after the beginning of the incentive. Each position is simply a price range, $[p_l, p_u]$, with specific liquidity. This characteristic range is divided into tick ranges. If, and only if, the current price is at the user's active tick, a portion of the prize is charged. The number of seconds the current price was in the user's position and the amount of liquidity given by the user on a single tick are used to determine the user's share of the award program.

Users retain ownership of the position, and have the ability to collect the earned fee from trades. There may be many incentives for a particular pool, and the user may stake simultaneously their positions in any number of them to maximize their profit.

Invariant stores the `seconds_per_liquidity` parameter globally for each tick, allowing us to determine the proportions of shares of a single person to all participants in the Staker incentive.

To make it easier to understand, here is a simplified mathematical formulas for determining the proportion of shares.

$$R = t \cdot v$$

- $R-$ the sum of all liquidity mining rewards,
- $t-$ the duration of the reward distribution,
- $v-$ the speed, at which reward distribution takes place.

$$L = \sum_i \ell(i)$$

- $L-$ the sum of all liquidity mining rewards,
- $\ell(i)-$ the duration of the reward distribution.

$$r(j) = R \cdot \frac{\ell(j)}{L} = t \cdot v \cdot \frac{\ell(j)}{\sum_i \ell(i)}$$

- $r(j)-$ the liquidity mining reward for a position.

Variable	Description
`total_reward_unclaimed`	The total amount of unclaimed rewards left for an incentive.
`total_seconds_claimed`	How many full liquidity-seconds have been already claimed for the incentive.
`start_time`	When the incentive rewards began in seconds.
`end_time`	When rewards are no longer being dripped out in seconds.
`liquidity`	The amount of liquidity, assumed to be constant over the period over which the snapshots are measured.
`seconds_per_liquidity_inside_initial`	The seconds per liquidity of the liquidity tick range as of the beginning of the period.
`seconds_per_liquidity_inside`	The seconds per liquidity of the liquidity tick range as of the current timestamp.
`current_time`	The current timestamp, which must be greater than or equal to the start time.

2 Calculation

```
pub fn calculate_reward(
    total_reward_unclaimed: Decimal,
    total_seconds_claimed: Decimal,
    start_time: u64,
    end_time: u64,
    liquidity: Decimal,
    seconds_per_liquidity_inside_initial: Decimal,
    seconds_per_liquidity_inside: Decimal,
    current_time: u64,
) -> Result<(Decimal, u64)> {
    if current_time <= start_time {
        return Err(ErrorCode::NotStarted.into());
    }

    let seconds_inside =
        (seconds_per_liquidity_inside.sub
        (seconds_per_liquidity_inside_initial)).mul(liquidity);

    let total_seconds_unclaimed = cmp::max(
        Decimal::from_integer(end_time as u128),
        Decimal::from_integer(current_time as u128),
    )
    .sub(Decimal::from_integer(start_time as u128))
    .sub(total_seconds_claimed);
    let result = (total_reward_unclaimed
        .mul(seconds_inside)
        .div(total_seconds_unclaimed))
    .v
    .try_into()
    .unwrap();
    return Ok((seconds_inside, result));
}
```

Above function return two values:

- `seconds_inside` — the total seconds spent inside the position's range for the duration of the stake,
- `result` — the amount of rewards owed.