Tokenizer

This whitepaper is to apply for the MVP Booster Grant (5000$).

Abstract

DeFi is growing rapidly with a lot of new protocols.

Usually an account can generate yield by staking or by providing liquidity to the protocol.

This creates a lot of manual integration for moving liquidity back and forward.

In this paper, we introduce a model to simplify what an account can do within DeFi, that is to say it can supply and then lock its liquidity, with the possibility, for advanced user, to trade.

Overview

We will describe how an account can supply liquidity in the protocol and how it is rewarded for this.

Then, we will describe a way to lock the liquidity provided.

In the last section, we will discuss about how the locked liquidity can be traded, to benefit from the changes in underlying asset.

As a conclusion, we will explain the expected profitability of the protocol and how liquidity will be used

Supply Liquidity

Provision of liquidity is the first and simplest mechanism offered to accounts.

Each account can provide liquidity in one or more of the available tokens and will receive as a reward an amount calculated based on the rate applied during the supply period.

The rate is variable over time and the supplied liquidity can be withdrawn at any time.

Lock Liquidity

//TODO

Furthermore, liquidity can be blocked and this entitles the account to an extra reward percentage calculated on the amount of liquidity blocked (both that relating to your position and the overall pool) and the duration of the block.

This liquidity block guarantees the account an additional reward that is the sum of the fixed extra reward in force at the time of the block plus the dynamically reward calculated with the various parameters as specified before.

This is called tokenization because it returns to the requesting account a Principal Token (fungible) and a Yield Token (non-fungible) that can be released after the specified date or exchanged independently of each other.

Trade Interest Rate Changes

After the liquidity lock operation the account can wait for the end of the locking period or execute a swap, which will be rewarded with a higher or lower value based on the current interest rate.

This swap can be performed both on this platform and other platforms, as both the fungible and non-fungible tokens can be withdrawn by the account.

Before expiry, the tokenized liquidity can be traded on the market, as their value will fluctuate based on changes in the reward percentages in force.

If the interest rate changes after the liquidity freeze, the value of the tokenized liquidity will also change. This opens up opportunities for trading by those speculating on future price movements

When an account wants to trade its tokenized position, the 'extra_reward' is compared to the current 'extra_reward'. If the current 'extra_reward' has increased, the account will receive a lower value. Conversely, if it has decreased, the account will receive a higher value. This dynamic opens up numerous trading opportunities on the market.

Profitability of the protocol

The described protocol employs liquidity in known instruments either directly or through asset management.

The liquidity collected will mainly be used for staking protocols, while other liquidity will be used for lending protocols.

The percentage of reward that will be recognized to suppliers will be slightly lower than what the protocol will obtain from its use, this difference will constitute its profitability together with fees.

Procotol details

Some formula have already been created for the Scrypto challenge, but to review the protocol for presentating to the Booster Grants some of these will be modified and improved.

In particular, those relating to trading interest rates changes and those related to rewards for locking liquidity.

New formulas to trading interest rates changes

https://www.investopedia.com/terms/m/macaulayduration.asp

To understand how a bond's price moves in response to changes in interest rates, we will use the Macaulay duration along with the concept of modified duration.

This bond-like locked liquidity has a maximum length of 1 year and since it does not pay an annual coupon it is like a zero-coupon bond, therefore the calculation of the Macaulay Duration simplifies significantly because

it is simply the time to maturity, which in this case is 1 year.

Macaulay Duration for a Zero-Coupon Bond

Calculate Multipliers:

- For a zero-coupon bond, the bond price P is given by

$$ P = { F \over (1 + y)N } $$

where:

```
F: F is the face value of the bond

y: y is the yield to maturity

N: N is the time to maturity (in this case, N = 1 year)
```

Simplified Macaulay Duration:

- Since there are no periodic coupon payments, the Macaulay Duration DM is simply the time to maturity:

$$ DM = { N = 1 year } $$

Modified Duration:

- The modified duration $D\ mod$ can be calculated using the Macaulay Duration:

$$ $D\ mod$ = { DM \over (1 + y) } $$

- Since DM = 1

$$ $D\ mod$ = { 1 \over (1 + y) } $$

Estimate the Price Change:

- The percentage change in bond price ΔP for a change in yield Δy is:

$$ {ΔP \over P} = - $D\ mod$ * Δy = - { 1 \over (1 + y) } * Δy $$

Explanation with Example Values

Given:

- Face value (FF) = $1000
- Yield to maturity (y) = 4% (0.04)
- Change in yield (Δy) = 1% (0.01)

Calculate the Bond Price:

$$ P = { 1000 \over (1 + 0,04) } = { 1000 \over (1,04) } ≈ 961.54$ $$

Calculate the Modified Duration:

$$ $D\ mod$ = { 1 \over (1 + 0,04) } = { 1 \over (1,04) } ≈ 0,9615 $$

Estimate the Price Change:

$$ {ΔP \over P} = - $D\ mod$ x Δy = { −0.9615×0.01 } = ≈−0.009615 or −0.96% $$

Estimated Dollar Price Change:

$$ {ΔP} ≈ { 961.54 × −0.009615 } ≈ −9.24 $$

The estimated dollar price change is approximately -$9.24.

New formulas to rewards for locking liquidity

A new formula needs to be implemented for calculating rewards dynamically based with respect to various factors

Liquidity Lock Reward Calculation

To calculate the reward for an account based on the amount of liquidity locked, the relative amounts within the account and the pool, and the duration of the lock, we use the following data and formula:

- The total supply received by the account.
- The amount locked by the account relative to its total supply received.
- The amount locked by the account relative to the total pool supply.
- The duration for which the liquidity is locked.
- A base reward expressed as an interest rate over time.

The reward formula will be structured as follows:

- Calculate a base reward using the interest rate over time.
- Calculate a multiplier based on the percentage of liquidity locked by the account and the percentage of the total pool locked by the account, as well as the lock duration.
- Combine these to determine the final reward.

Variables:

- totalSupplyReceived: The total supply received by the account.
- liquidityLocked: The amount of liquidity locked by the account.
- totalPoolSupply: The total supply in the pool where the account is supplying tokens.
- baseInterestRate: The base interest rate applied to the amount locked over time.
- lockDuration: The duration for which the liquidity is locked (in days, for simplicity).

Calculate Multipliers:

- Account Lock Percentage Multiplier: $$ lockedPercentageAccount = { liquidityLocked \over totalSupplyReceived } $$

- Pool Lock Percentage Multiplier:

$$ lockedPercentagePool = { liquidityLocked \over totalPoolSupply } $$

- Lock Duration Multiplier:

$$ timeMultiplier = { lockDuration \over 365 } $$

Update the Interest Rate:

- Combine the multipliers with the base interest rate to get the updated interest rate:

$$ interestUpdated = baseInterestRate × (1 + lockedPercentageAccount+ lockedPercentagePool + timeMultiplier) $$

Final Reward Formula:

- Use the updated interest rate in the reward formula:

$$ reward = { (liquidityLocked * interestUpdated * lockDuration) \over 36500 } $$

Where:

- `liquidityLocked` is the amount of liquidity locked by the account.
- `baseInterestRate` is the base interest rate (annual).
- `lockedPercentageAccount` is the percentage of liquidity locked by the account
- `lockedPercentagePool` is the percentage of the total pool locked by the account
- `timeMultiplier` is the duration multiplier

Example Implementation

```
def calculate_reward(total_supply_received, liquidity_locked, total_pool_supply,
base_interest_rate, lock_duration):
    # Calculate the percentage of liquidity locked by the account
    locked_percentage_account = liquidity_locked / total_supply_received

    # Calculate the percentage of the total pool locked by the account
    locked_percentage_pool = liquidity_locked / total_pool_supply

    # Determine the time-based multiplier
    time_multiplier = lock_duration / 365

    # Update the interest rate
    interest_updated = base_interest_rate * (1 + locked_percentage_account +
locked_percentage_pool + time_multiplier)

    # Calculate the final reward
    reward = (liquidity_locked * interest_updated * lock_duration) / 36500

    return reward

    # Example usage
    total_supply_received = 1000  # Example total supply received by the account
    liquidity_locked = 500  # Example liquidity locked by the account
```

```
    total_pool_supply = 10000  # Total supply in the pool
    base_interest_rate = 5  # Example base interest rate (5% annual)
    lock_duration = 30  # Lock duration in days

    reward = calculate_reward(total_supply_received, liquidity_locked,
total_pool_supply, base_interest_rate, lock_duration)
    print(f"The reward for the account is: {reward}")
```

Explanation with Example Values

Calculate Percentages and Multipliers

Let's suppose than an account lock 500 tokens out of it supplied 1000 tokens in a pool of 10.000 tokens at a 5% APY for 30 days.

- Account Lock Percentage: $$ lockedPercentageAccount = { (500) \over 1000 } = 0,5 $$
- Pool Lock Percentage: $$ lockedPercentageAccount = { (500) \over 10000 } = 0,05 $$
- Time Multiplier: $$ timeMultiplier = { (30) \over 365 } = 0,082 $$
- Update the Interest Rate: $$ interestUpdated = { 5 * (1 + 0,5 + 0,05 + 0,082) = 5 * 1,632 = 8,16 percent } $$
- Calculate Final Reward: $$ reward = { (500 * 8,16 * 30) \over 36500 } = 3,35 units $$

At the maturity date the account gets a reward of 3,35 token, netting a 8,16% API.

Conclusion

This protocol is still under work, in particular it is outside the scope of this whitepaper to describe how the collected liquidity will be used, a topic that will be discussed later.