Distribution Model Contract (Solidity)

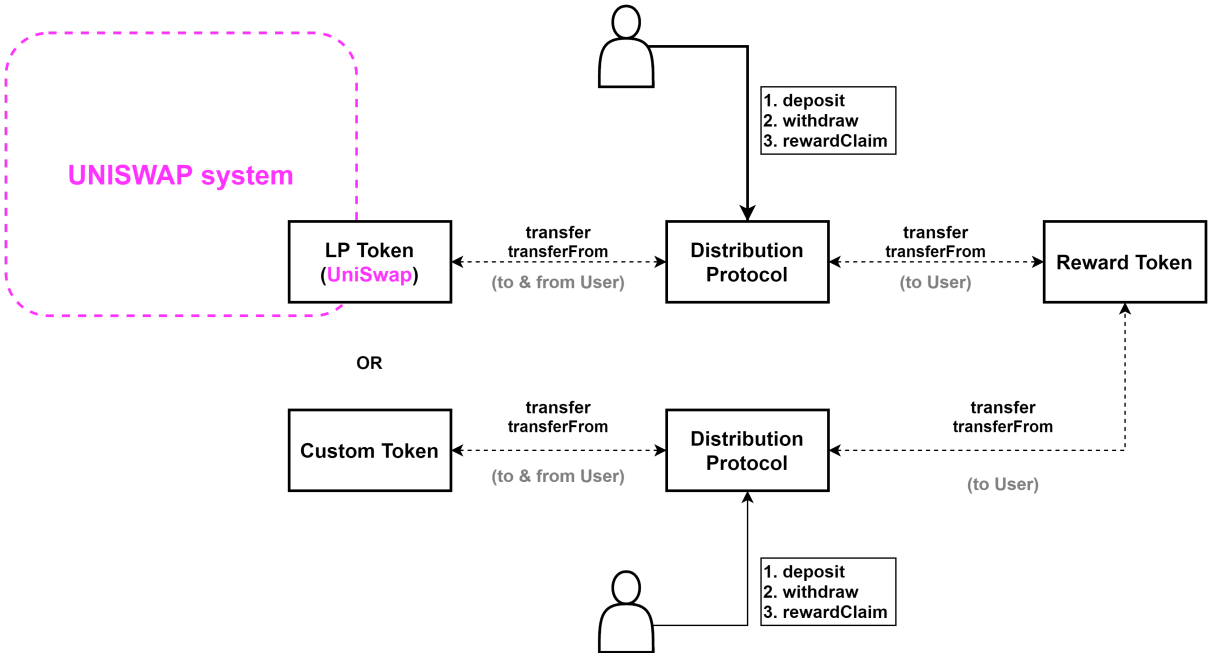
1. Introduction

This smart contract implements a token staking service. Two types of tokens are involved in the staking service: (1) Contribution Token and (2) Reward Token. When a user deposit his or her Contribution Tokens to our smart contract, the user can get earn the Reward tokens for deposited amount and period. Both the Contribution Token and Reward tokens are the ERC-20 tokens. In our service scenario, this smart contact code will be used for staking Uniswap Liquidity Pool Tokens ("LP Tokens") or BFC (our token) as for the Contribution Tokens. We will issue another tokens for the Reward Tokens.

This document describes the **Service Model** (2.1), **Contracts** (2.2), and **Inheritance relationship** (2.3) of this smart contract.

2. Architecture

2.1. Service Model



This figure depicts our service model. We aim at providing two staking services with the same smart contract code for the LP tokens and the custom tokens (e.g., our token). The "Distribution protocol" entity is our smart contract. User can perform three kinds of actions:

1. `deposit` : Deposit his Contribution tokens (i.e., LP Tokens or Custom Tokens)
2. `withdraw` : Withdraw his Contribution tokens
3. `rewardClaim` : Claim his Reward tokens

The Distribution Protocol accumulates the Reward tokens by the deposited amount and period of the Contribution tokens.

Parameter changes

The rewards (Reward Tokens) to distribute in a single block (may) keep decreasing linearly. The main parameters for this process:

1. `_rewardPerBlock` : The initial reward amount to distribute in a single block
2. `_decrementUnitPerBlock` : The decrement amount per block

Namely, the Reward Tokens at the block t is

$y[t]$ = -1 * `_decrementUnitPerBlock` * t + `_rewardPerBlock`

The administrator (owner) can set the parameters by calling `setRewardVelocity`.

However, the curve for the distribution decrement can have inflection points, where the parameters change. We call the points as "reward velocity points." The administrator can register or remove the reward velocity points via `registerRewardVelocity` and `deleteRewardVelocity`.

2.2. Contracts

2.2.1. Logic modules

The main service contract is `DistributionModelV3`. The smart contract is built up from the basic module, referred to as `internalModule`, to the full-featured module, referred to as `externalModule`.

- ./contracts
  - **DistributionModelV3.sol**
    - Inheritance: `externalModule`
    - Description: Implement the constructor of Distribution Model Contract

The following modules are in the inheritance relationship: `internalModule` → `viewModule` → `externalModule`.

- ./contracts/module
  - **internalModule.sol**
    - Inheritance: `storageModule`, `eventModule`, `safeMathModule`
    - Description: Implement internal functions that process user and admin requests (for instance, updating contract state and service logic)
      - `_deposit`
        1. Update states of global amount and user amount
        2. `emit Deposit`
        3. Call ERC-20 function for tokens to be deposited into the DMC
      - `_withdraw`

1. Update states of global amount and user amount
2. `emit Withdraw`
3. Call ERC-20 function for tokens to be withdrawn from the DMC

- `_rewardClaim`
  1. Update states of user reward amount
  2. `emit Claim`
  3. Call ERC-20 function to transfer Reward Tokens

- `_redeemAll`
  1. Call the calculation function for the DMC to update the global state variables
     - `_updateRewardLane`
  2. Calculate and update user state variable (for the accrued Reward Tokens of each user)

- `_updateRewardLane`

  Update global states of the DMC variables

  - Called by all user actions (`_deposit`, `_withdraw`, `_redeemAll`, `_rewardClaim`)
  - Called by admin action `setRewardVelocity`

  1. Load global variables to memory
  2. Check if any inactive reward distribution velocity parameters exist (loop)
     - If existing, calculate and update memory variables
       - Call `_calcNewRewardLane`
  3. Check if any inactive current distribution velocity parameters exist

        - If existing, calculate and update memory variables
            - Call `_calcNewRewardLane`
    4. Update global states of the DMC variables

- `_calcNewRewardLane`
    - Calculate new "`rewardLane`" (in the DMC document) and `rewardPerBlock`
        - Call `_meanOfinActiveLane` (described below)
        - Call `_getNewRewardPerBlock` (described below)

- `_meanOfinActiveLane`
    - Calculate the arithmetic mean of the subsequence of reward per block (R_b in the DMC Document)
        - Use the first term (`a`), common difference (in an arithmetic series) (`d`), and number of elements (`n`)

- `_getNewRewardPerBlock`
    - Calculate the last term of the subsequence of reward per block
- `_registerRewardVelocity`
    - Register new reward parameter to the array (`registeredPoints[]`)
- `_deleteRewardVelocity`
    - Remove the reward parameter at `index` from the array (`registeredPoints[]`)

- **externalModule.sol**
    - Inheritance: `viewModule`

- Description: Implement external functions that serve as the entry point for user and admin requests and calls internal functions according to the logic (for instance, wrapping internal actions of user and admin)
  - **userAction**
    - `deposit`
      1. Call `_redeemAll`
      2. Call `_deposit`
    - `withdraw`
      1. Call `_redeemAll`
      2. Call `_withdraw`
    - `rewardClaim`
      1. Call `_redeemAll`
      2. Call `_rewardClaim`
  - **adminAction**
    - `setClaimLock`
      - `emit ClaimLock`
      - Update `modifier` state
    - `setWithdrawLock`
      - `emit WithdrawLock`
      - Update `modifier` state
    - `registerRewardVelocity`
      - `emit SetRewardParams`

        - Register new reward parameters
    - `deleteRewardVelocity`
        - Remove the reward parameter at `index` from the array (`registeredPoints[]`)
    - `setRewardVelocity`
        - `emit RegisterRewardParams`
        - Set new reward parameters immediately
- **viewModule.sol**
    - Description: Implement view functions for the web-frontend components.

2.2.2. Storage and auxiliary modules

- ./contracts/module
    - **storageModule.sol**
        - Description: Define the state variables for the global and per-user information.
        - Define `struct`
            - `Account`
            - `RewardVelocityPoint`
            - `UpdateRewardLaneModel`
    - **eventModule.sol**
        - Description: Define events for user and admin state updates.
        - Define `event`

```
//user Event
event Deposit(address, uint256, uint256, uint256);
event Withdraw(address, uint256, uint256, uint256);
event Claim(address, uint256);
event UpdateRewardParams(uint256, uint256, uint256);

//admin Event
event ClaimLock(bool);
event WithdrawLock(bool);
event SetRewardParams(uint256, uint256);
event RegisterRewardParams(uint256, uint256, uint256);
```

- **safeMathModule.sol**
  - Description: Implement the primitive functions of `safeMath`
  - Implement arithmetic operations for `uint256`
  - Implement multiplication and division for `uint256` with 18 decimals

2.3. Inheritance relationship


DistributionModelV3
constructor(address, address, uint256, uint256)
externalModule
modifier onlyOwner()
modifier checkClaimLocked()
modifier checkWithdrawLocked()
function deposit(uint256) external
function withdraw(uint256) checkWithdrawLocked external
function rewardClaim() checkClaimLocked external
function setClaimLock(bool) onlyOwner external
function setWithdrawLock(bool) onlyOwner external
function registerRewardVelocity(uint256, uint256, uint256) onlyOwner external
function deleteRegisteredRewardVelocity(uint256) external
function setRewardVelocity(uint256, uint256) onlyOwner external
viewModule
function modelInfo() external view returns (uint256, uint256, uint256, uint256, uint256)
function getParams() external view returns (uint256, uint256, uint256, uint256)
function getRegisteredPoint(uint256) external view returns (uint256, uint256, uint256)
function getRegisteredPointLength() external view returns (uint256)
function userInfo(address) external view returns (uint256, uint256)
internalModule
function _deposit(address, uint256) internal
function _withdraw(address, uint256) internal
function _redeemAll(address) internal
function _rewardClaim(address) internal
function _updateRewardLane() internal returns (uint256)
function _calcNewRewardLane(uint256, uint256, uint256, uint256, uint256) internal pure returns (uint256, uint256)
function _registerRewardVelocity(uint256, uint256, uint256) internal
function _deleteRegisteredRewardVelocity(uint256) internal
function _meanOfInactiveLane(uint256, uint256, uint256) internal pure returns (uint256)
function _getNewRewardPerBlock(uint256, uint256, uint256) internal pure returns (uint256)
function _setClaimLock(bool) internal
function _setWithdrawLock(bool) internal
safeMathModule
uint256 constant one = 1 ether;
function safeAdd(uint, uint) pure returns (uint256)
function safeSub(uint, uint) pure returns (uint256)
function safeMul(uint, uint) pure returns (uint256)
function safeDiv(uint, uint) pure returns (uint256)
function expMul(uint, uint) pure returns (uint256)
function expDiv(uint, uint) pure returns (uint256)
storageModule
address owner;
bool claimLock
bool withdrawLock
uint256 rewardPerBlock
uint256 decrementUnitPerBlock
uint256 rewardLane
uint256 lastBlockNum
uint256 totalDeposited
ERC20 lpErc; ERC20 rewardErc;
mapping(address => Account) accounts;
uint256 passedPoint;
RewardVelocityPoint[] registeredPoints;
eventModule
event Deposit(address, uint256, uint256, uint256);
event Withdraw(address, uint256, uint256, uint256);
event Claim(address, uint256);
event UpdateRewardParams(uint256, uint256, uint256);
event ClaimLock(bool);
event WithdrawLock(bool);
event SetRewardParams(uint256, uint256);
event RegisterRewardParams(uint256, uint256, uint256);
event DeleteRegisterRewardParams(uint256, uint256, uint256, uint256, uint256);
Structure
struct Account {
uint256 deposited;
uint256 pointOnLane;
uint256 rewardAmount;
}
struct RewardVelocityPoint {
uint256 blockNumber;
uint256 rewardPerBlock;
uint256 decrementUnitPerBlock;
}
struct UpdateRewardLaneModel {
uint256 len; uint256 tmpBlockDelta;
uint256 memPassedPoint; uint256 tmpPassedPoint;
uint256 memThisBlockNum;
uint256 memLastBlockNum; uint256 tmpLastBlockNum;
uint256 memTotalDeposit;
uint256 memRewardLane; uint256 tmpRewardLane;
uint256 memRewardPerBlock; uint256 tmpRewardPerBlock;
uint256 memDecrementUnitPerBlock; uint256 tmpDecrementUnitPerBlock;
}