(ENG) BiFi Smart Contract: Interest Model Design

1. Introduction

BiFi is a DeFi project on Ethereum that enables users to deposit various digital assets and borrow other digital assets against the deposits as collateral. Like traditional banks, BiFi provides interest to depositors and charges interest to borrowers, but the interest rates are determined automatically by the liquidity in the market. If the amount of deposits in the market increases, the interest rates decrease, making borrowing more advantageous. If the amount of borrows increase, the interest rates increase, making depositing more advantageous. Furthermore, the interest calculation is designed to be compounding. As the frequency of use increases, the calculation of interests increasingly approaches 'per block compounding' because we implement the interest compounding with the simple-interest operations.

2. Assumptions and Terminology

Assumptions

- Interest rates for deposits and borrows are determined automatically by the liquidity in the market.
- BiFi Smart Contract provides users four types of actions in BiFi:
    - `deposit`: Deposit digital asset
    - `withdraw`: Withdraw deposit
    - `borrow`: Borrow digital asset against the deposit as collateral
    - `repay`: Repay borrowed digital asset
- When the borrowed asset exceeds the deposited asset in a particular ratio (Loan-To-Value Ratio), the deposit can be liquidated.

Terminology

Global Shared

- H_L:Block Height of the last block when an action occurred
- E_D: Exchange Rate for deposits, which is the cumulative product of interest rates for deposits.

- E_B: Exchange Rate for borrow, which is the cumulative product of interest rates for borrow.
- D_T: Total deposited amount in the market
- B_T: Total borrowed amount in the market
- $E_D[u]$: Deposit Exchange Rate for user u
- $E_B[u]$: Borrow Exchange Rate for user u
- $D[u]$: Deposit amount of user u, which is the sum of principal and interest.
- $B[u]$: Borrow amount of user u, which is the sum of principal and interest

Shared in a block

- tH_L: Temporary variable for H_L
- tE_D: Temporary variable for E_D, i.e., the latest value of E_D.
- tE_B: Temporary variable for E_B, i.e., the latest value of E_B

Constants

- $blocksPerYear$: Total number of Blocks created in a year (e.g., 2,102,400 in Ethereum)
- r_{min}: Minimum interest rate for borrowing, set by the admin (e.g., 2%)
- s: Liquidity sensitivity, set by the admin

3. Calculation Method for Amount with Interest

BiFi is implemented by several "Token Handlers" that provide deposit and borrow functions to the user. Every token supported by BiFi has its own Handler that manages the deposit, withdraw, borrow, and repay functions, and the Handler stores all relevant variables, including Market Status Variables (D_T, B_T, D for users and B for users), Interest Rate Calculation Variables (r_{min}, s) and Interest Calculation Variables (H_L, E_D, E_B, $E_D[u]$ and $E_B[u]$). The following section describes how these variables are used to calculate the interest rates and the interest.

3.1 Interest Rate Formula

Annual Borrow Interest Rate (BIR_y) and Annual Supply Interest Rate (SIR_y) are determined by D_T and B_T. BiFi uses these annual interest rates by dividing them into Per Block Interest Rates (BIR_b and SIR_b).

Utilization

- Ratio of the total amount of borrows to the total liquidity (the sum of all deposits and borrows) in the market.
- Formula: $U = \frac{B_T}{D_T + B_T}$

Interest Rates

- Annual Interest Rates
  - BIR_y = $r_{min} + U * s$
  - SIR_y = $BIR_y * U$
- Per Block Interest Rates
  - BIR_b = $\frac{BIR_y}{blocksPerYear}$
  - SIR_b = $\frac{SIR_y}{blocksPerYear}$

3.2 Calculating the Amount with Interest

While the interest rate calculations for deposits and borrows are different, their role is the same. Therefore, the Per Block Interest Rates (SIR_b and BIR_b) can both be expressed as r.

Amount with Interest

Since BiFi provides compounding interest for every block, the amount with interest (A_{i_2}) for a principal starting at block i_1 block (P_{i_1}) and ending at block i_2 is calculated as follows:

$$A_{i_2} = P_{i_1} * \prod_{i=i_1+1}^{i_2} (1 + r_i)$$

However, this requires r_i to be stored at every block, making it inefficient in terms of storage.

Exchange Rate (Cumulative Product of Interest Rates)

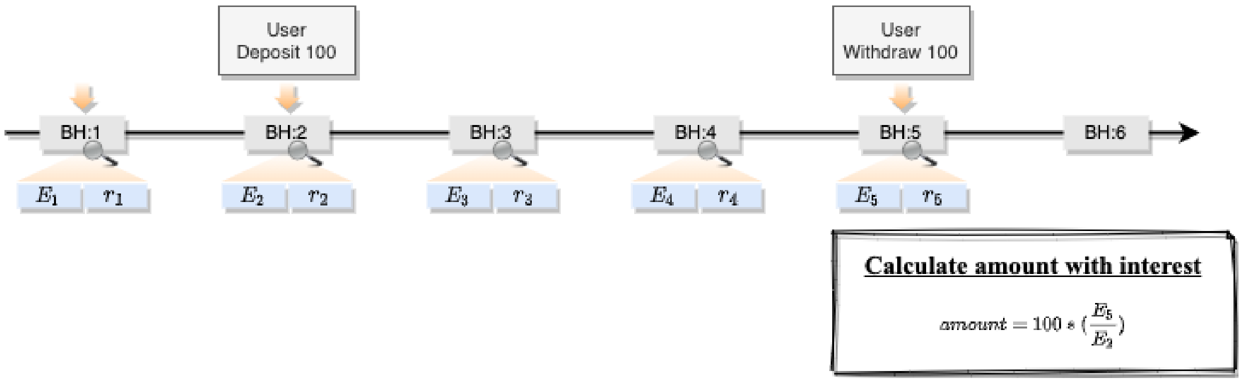
The exchange rate from the starting from block i to block j ($ExchangeRate_{i,j}$) is defined as below. BiFi Smart Contract has the variable `ExchangeRate`. In every block, it multiplies $(1 + r)$ to the existing value of `ExchangeRate` then stores the updated value in `ExchangeRate`.

$$Def)\ ExchangeRate_{i,j} = \prod_{i}^{j}(1 + r_i)$$

User is assigned the two values of `ExchangeRate`, first when initial action is taken at block i_1 and then when later action is taken at block i_2. These values are used to calculate the final amount with interest (A_{i_2}). Using the cumulative product of $(1 + r)$ eliminates the need to store the value of $(1 + r)$ at every block, thereby increasing the storage efficiency for BiFi Smart Contract and reducing the gas fee for users.

Example of Calculating Amount with Interest

In the following diagram, E_i is the `ExchangeRate`, r_i is the interest rate, and A_i is the amount with interest at the i-th block. When a user deposits 100 tokens at `BH:2`, the amount with interest at `BH:5` can be calculated as following:

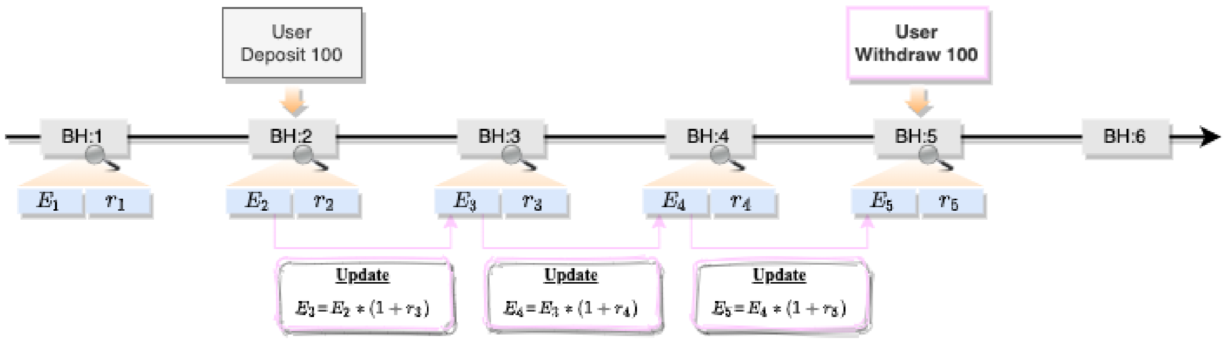


- The amount with interest for the user at `BH:5` is the value of principal (100) multiplied by (1+r_3), (1+r_4) and (1+r_5).
- Given $\frac{E_5}{E_2} = (1 + r_3)(1 + r_4)(1 + r_5)$, the amount with interest is the value of principal (100) multiplied by $\frac{E_5}{E_2}$.

3.3 Updating Exchange Rate

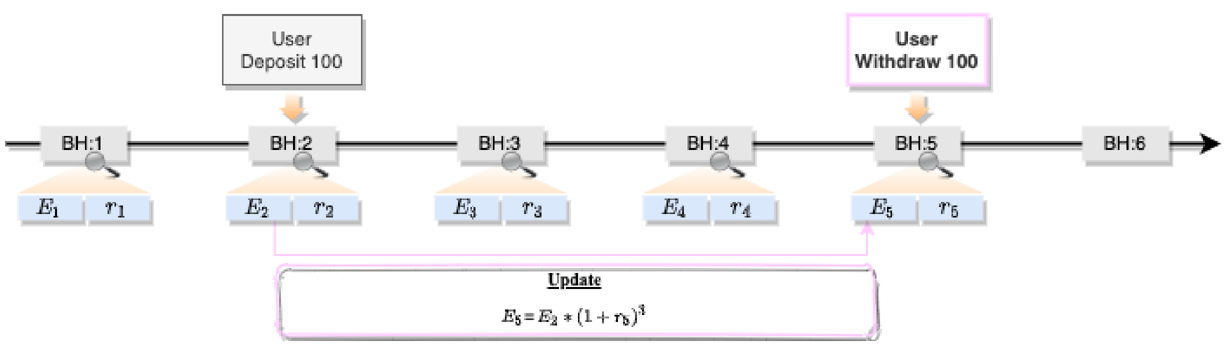
Basic Update Logic

`ExchangeRate` is updated in every block by multiplying $(1 + r)$ to the existing value, as shown below:



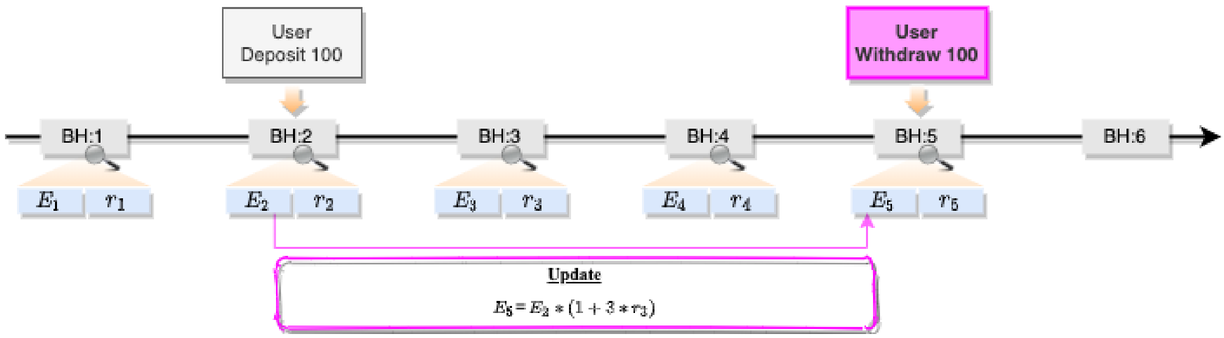
Batch Overdue Update

`ExchangeRate` cannot be updated in a block where no user action occurred to change the state of the BiFi Smart Contract. Hence the current action needs to make all the necessary updates since the last action. Since interest rate is determined by D_T and B_T, it has the same value for all blocks where no actions occurred ($r_3 = r_4 = r_5$). Therefore, the action in `BH:5` can complete the overdue updates by multiplying `ExchangeRate` by $(1 + r_5)$ three times.



Approximation Update

If no user action occurs for an extended period, the batch overdue update can have high computational cost. In order to preserve the computational efficiency regardless of the period without any actions, we opted for $(1 + 3 * r)$, instead of setting the update formula to $(1 + r)^n$. Functions $y = (1 + r)^x$ and $y = 1 + rx$ are approximately the same when the value of r is small.



3.4 Applying Liquidity Accurately

`ExchangeRate` is updated based on current liquidity (D_T, B_T), as follows:

1. Calculate current interest rate r, based on D_T and B_T.
2. Calculate elapsed time: $\delta =$ `block.number` $- H_L$, (`block.number` is a global variable provided in Solidity)
3. Update `ExchangeRate` : $E_{next} = E_{prev} * (1 + \delta * r)$.
4. Update H_L : $H_L =$ `block.number`

This process runs smoothly, if there is at most one action in a single block. However, if there are several actions in one block, `ExchangeRate` may not accurately reflect the correct balance in the market. In the process above, $\delta = 0$ for all actions except the first action in the block. This results in `ExchangeRate` not being updated for second action onwards, and thus the subsequent changes in liquidity is not reflected. Ultimately, this process violates the market design that derives interest rates based on the cumulative liquidity from all actions.

Solution: Using Shared Variable in a Block

To resolve this issue, every action within a single block must update `ExchangeRate` based on its own interest rates. To achieve this, each action must know the `ExchangeRate` and H_L stored by the last action of the previous block. In BiFi Smart Contract, the first action in the block stores the previous values of `ExchangeRate` (tE_D, tE_B) and $H_L (tH_L)$ as a temporary global variable available to all actions within the block.

4. Liquidation

4.1 Loan-To-Value Ratio

Loan-To-Value (LTV) Ratio is the ratio of the deposit amount to the borrow amount of a user, representing the status of the collateral loan of the user. A financial service must execute the liquidation process before the LTV ratio exceeds 1 to preserve assets. The LTV ratio of a user may increase based on several factors:

- Decrease in price of the digital asset deposited as collateral
- Increase in price of the digital asset borrowed
- When the value of deposit amount with interest becomes smaller than the value of borrowed amount with interest

4.2 Liquidation Target and Liquidator

Liquidation Target

BiFi defines users who exceed a certain threshold of LTV ratio as "Liquidation Targets", and their collateral (i.e., deposit) may be liquidated.

Liquidation Process and Liquidator

A decentralized financial service cannot execute liquidation by itself. Therefore, it must create an economic system that incentivizes the "Liquidators".

For example, if Liquidation Targets are defined as users with LTV ratio higher than 0.92, the Liquidator can repay the borrows on behalf of the Liquidation Target, and can take the collateral of the Liquidation Target. If a Liquidation Target with

LTV ratio of 0.92 goes through the liquidation process, the liquidator can expect approximately 8% profit, because the expected cost of repayment is 92 and expected gain is 100.