BALANCER: A PROTOCOL FOR MULTI-TOKEN AUTOMATED MARKET-MAKING

FERNANDO MARTINELLI

Abstract. Balancer is a new Ethereum-based protocol that allows any ERC20 portfolio to be continuously rebalanced while at the same time generating passive income through trading fees. Previously idle ERC20 tokens become Balancer pools, allowing anyone to trustlessly and non-custodially trade any two assets contained therein. Differently from Uniswap[1], pools can contain any number of ERC20 tokens with arbitrary value weights. They are also owned and controlled individually, as opposed to combined into single pairwise markets like on Uniswap.

1. Introduction

Investment portfolios such as Index Funds are a very common financial instrument. The first index fund became effective in 1972, its investment objective being to approximate the performance of the Dow Jones Industrial Stock Average. Ever since, investors heavily rely on different portfolio strategies to hedge risk and achieve diversification. With blockchain, digital assets have become a new class of investment, which naturally ported the index fund concept from the conventional financial world to the new blockchain context. Index funds guarantee investors a constant and controlled exposure to a basket of assets. If one of its assets out- or under-performs, it is respectively sold or bought to keep its value share of the total portfolio constant.

Both in the conventional financial system as well as in the blockchain context, Index Funds and other types of investment portfolios charge investors fees for managing and holding their funds. These fees are necessary to pay for the costs of actively rebalancing the index funds, be it by manual traders or automatic bots.

Balancer protocol, on the other hand, allows any portfolio owner to maintain their portfolio continuously and effortlessly rebalanced. This is done not only with the owner keeping full control and custody of their portfolio, but at the same time also being rewarded through trading fees, as we will see in detail in the following.

To summarize, instead of paying fees to keep a balanced portfolio, ERC20 token holders can now earn fees by using Balancer protocol to do so.

2. Present Work

There are many centralized solutions for portfolio management and for investing in index funds. We will however limit this analysis to the decentralized/non-custodial solutions in

existence, since that is a fundamental characteristic Balancer protocol was designed to have.

Melonport[2] is a protocol that allows fund managers to streamline and set up funds more easily than conventionally. Although an interesting concept, Melonport did not seem to get traction, as we believe it is still complicated to use and retail investors have to pay fees to participate in its investment funds. The existence of the MLN token also adds unnecessary friction to the process.

Set Protocol[3] is, in our view, a more interesting approach that has recently been launched and is gaining traction. It allows anyone to buy sets of tokens, i.e. tokens that represent ownership of a basket of other tokens. These sets may have embedded logic as for when the underlying tokens are swapped/exchanged for one another. For example, the 50%/50% BTC ETH set is a buy and hold strategy that rebalances monthly. Other more complex strategies include trend and range bound trading.

It is important to stress that none of the protocols mentioned above provide liquidity providers with native rewards or earnings. On the contrary, rebalancing sets within Set Protocol incurs in both trading fees as well as network gas costs. The latter is currently being paid for by Set Labs Inc., the company that develops Set Protocol.

Uniswap, Kyber[4] and Compound[5] are examples of protocols that enable liquidity providers to earn fees through providing liquidity.

Balancer Protocol achieves the objective of rebalancing portfolios as does Set Protocol at the same time rewarding liquidity providers as does Uniswap. Balancer is a generalization of Uniswap. All Uniswap exchanges can be emulated with Balancer Protocol, however the opposite is not true: arbitrary portfolios cannot be invested and continuously rebalanced in Uniswap as they can with Balancer Protocol.

3. Balancer Protocol Architecture

3.1. **Balancer Pools.** Balancer pools are the building blocks of Balancer protocol. Any Ethereum address that owns two or more ERC20 tokens can create a Balancer Pool, as illustrated in Figure 1.

The pool will have a certain total value and each of the tokens will make up a percentage of that value. These percentages are the token weights. These weights are defined upon the pool creation and can be changed at any later time by the pool owner. If not changed, Balancer Protocol will ensure that the percentage of value each token in the pool represents will remain stable, even if exchange rates between tokens may change. This is achieved through protocol incentives created for arbitrageurs to arbitrage between Balancer pools and external markets—other centralized or decentralized exchanges.

[2]https://melonport.com/
[3]https://www.tokensets.com/
[4]https://kyber.network
[5]https://compound.finance/

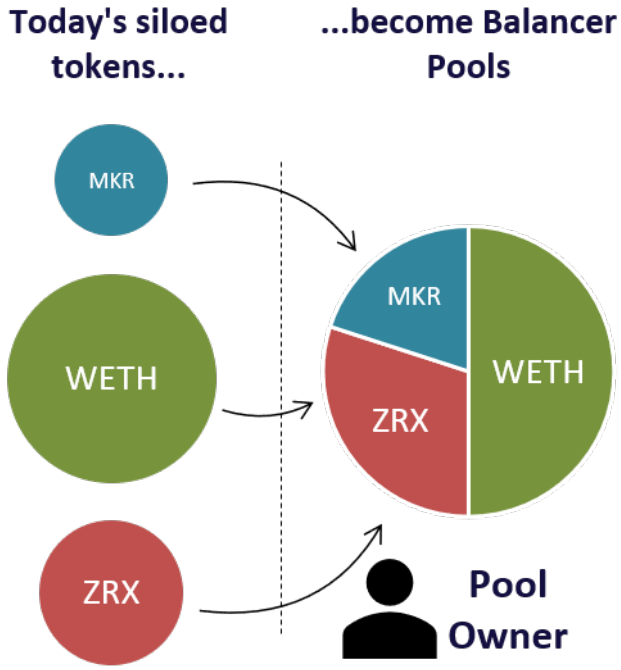

FIGURE 1. Example of Balancer Pool containing a value distribution of 50% in WETH, 30% in ZRX and 20% in MKR tokens.

3.2. **Trading with Individual Pool.** Balancer Pools are automated market-makers because they allow any Ethereum user to trade on any pair contained in the pool. In the example pool above one could trade on any of the following pairs: WETH/MKR, MKR/ZRX and ZRX/WETH. Both directions, buying or selling, are always possible.

3.2.1. *Spot Exchange Rate.* Similarly to Uniswap, we propose the exchange rates of any of the $\frac{(n^2-n)}{2}$ pairs of tokens within a pool of n tokens to be defined exclusively by their balances[6]. We propose nevertheless an addition that enables Balancer pools to assume any basket value distribution as opposed to Uniswap[7]. The spot exchange rate between any two tokens, $SpotExchangeRate_{in}^{out}$, or in short SER_{in}^{out}, is the the ratio of the token balances *normalized* by their weights:

[6]This means that the system state and its evolution is self-contained, i.e. no external feeds or oracles are necessary. This makes Balancer Protocol an excellent and trustworthy oracle, so long as there is enough liquidity available.

[7]In Uniswap each exchange pair always holds the same value in ETH as in the ERC20 token. This is a consequence of the simple invariant function $x.y = k$ used, where x represents the ETH balance, y represents the ERC20 balance and k is a value that would remain constant after any number of trades, were it not for the 0.3% fees charged.

$$SER_{in}^{out} = \frac{\frac{Q_{out}}{w_{out}}}{\frac{Q_{in}}{w_{in}}} \tag{1}$$

Where:

- Q_{out} is the balance of token *out*, the token being bought by the trader which is going *out* of the pool.
- Q_{in} is the balance of token *in*, the token being sold by the trader which is going *in* the pool.
- w_{out} is the weight of token *out*
- w_{in} is the weight of token *in*

For example, if in the pool described above we had 10 WETH, 6000 ZRX and 1 MKR, then the spot exchange rates offered by this pool would be:

$$SER_M^W = \frac{\frac{Q_W}{w_W}}{\frac{Q_M}{w_M}} = \frac{\frac{10}{0.5}}{\frac{1}{0.2}} = 4$$

$$SER_Z^M = \frac{\frac{Q_M}{w_M}}{\frac{Q_Z}{w_Z}} = \frac{\frac{1}{0.2}}{\frac{6000}{0.3}} = 0.00025$$

$$SER_W^Z = \frac{\frac{Q_Z}{w_Z}}{\frac{Q_W}{w_W}} = \frac{\frac{6000}{0.3}}{\frac{10}{0.5}} = 1000$$

The spot exchange rate in the opposite direction of the trade is just the inverse of the original:

$$SER_{out}^{in} = \frac{1}{SER_{in}^{out}} \tag{2}$$

3.2.2. *Effective Exchange Rate.* It is important to bear in mind that SER_{in}^{out} is the theoretical exchange rate for infinitesimal trades, which would incur in no slippage. In practice, the effective exchange rate for any trade depends on the amount being traded, which always leads to some slippage. If we define q_{out} as the amount of token *out* being bought by the trader and q_{in} as the amount of token *in* being sold by the trader, then we can define $EffectiveExchangeRate_{in}^{out}$, or in short EER_{in}^{out} as:

$$EER_{in}^{out} = \frac{q_{out}}{q_{in}} \tag{3}$$

And as mentioned above, EER tends to SER when traded amounts tend to 0:

$$SER_{in}^{out} = \lim_{q_{out}, q_{in} \to 0} EER_{in}^{out} \tag{4}$$

The derivation of q_{out} in terms of q_{in} and vice-versa depend on the multidimensional invariant surface described in detail in section 4.

3.2.3. *Arbitrage and External Market Exchange Rate Matching.* From Eq.1, considering weights constant during trades[8], the spot exchange rates offered by Balancer Pools only change with changing token balances. If the pool owner does not add or remove tokens to/from the pool, token balances can only change through trades. We have then a feedback loop that causes exchange rates of tokens being bought (token *out*) to decrease—i.e their prices increase—and vice-versa. One can prove that whenever external market exchange rates are different from those offered by a Balancer Pool, an arbitrageur will make the most profit by trading with that pool until its exchange rates equal those on the external market. When this happens there is no more arbitrage opportunity. These arbitrage opportunities are the reason why we see Uniswap exchange rates move in lockstep with the rest of the market.

3.3. **Off-Chain Smart Order Routing (SOR).** Since Balancer Protocol allows any number of pools to be created, there may be several different pools that contain any given trading pair. If a trader wants to trade 100 Token A for Token B and the whole trade is carried out only in a single Balancer Pool, there may be a lot of slippage. That slippage would create a difference between exchange rates amongst different Balancer Pools, allowing for arbitrageurs to profit. In other words, trading with only one pool is equivalent to leaving money on the table only for arbitrageurs.

A better approach to minimize slippage and maximize the amount of tokens B received is spreading the 100 Tokens A across multiple Balancer pools such that all trades shift the exchange rates of the pools to the same value. Finding which amounts should be traded and in what pools is a non-linear, multi-variable optimization problem. Balancerpro.com has a free off-chain Smart Order Routing algorithm that solves this problem and finds the exact amounts to be traded with the most advantageous pools available. For that optimization we take into account the gas price selected as well as the states of all available Balancer pools in the desired pair. A simplified illustration of the Smart Order Routing is depicted in Fig.2.

4. Mathematical Background and Proofs

4.1. **Value function.** The bedrock of Balancer Protocol's mathematical framework is a value function that, when kept invariant, ensures that the spot exchange rates of any two tokens are as defined in Eq.1. As we will see in section **??**, this property has as corollary that the value distribution across all tokens is kept constant. This is what makes Balancer Protocol suitable for rebalancing portfolios.

We propose V to be defined as:

$$V = \prod_k (Q_k)^{w_k} \tag{5}$$

[8]Weights can be changed at any time by the pool owner. However since trade events are atomic, weights cannot be changed by the pool owner during a trade.

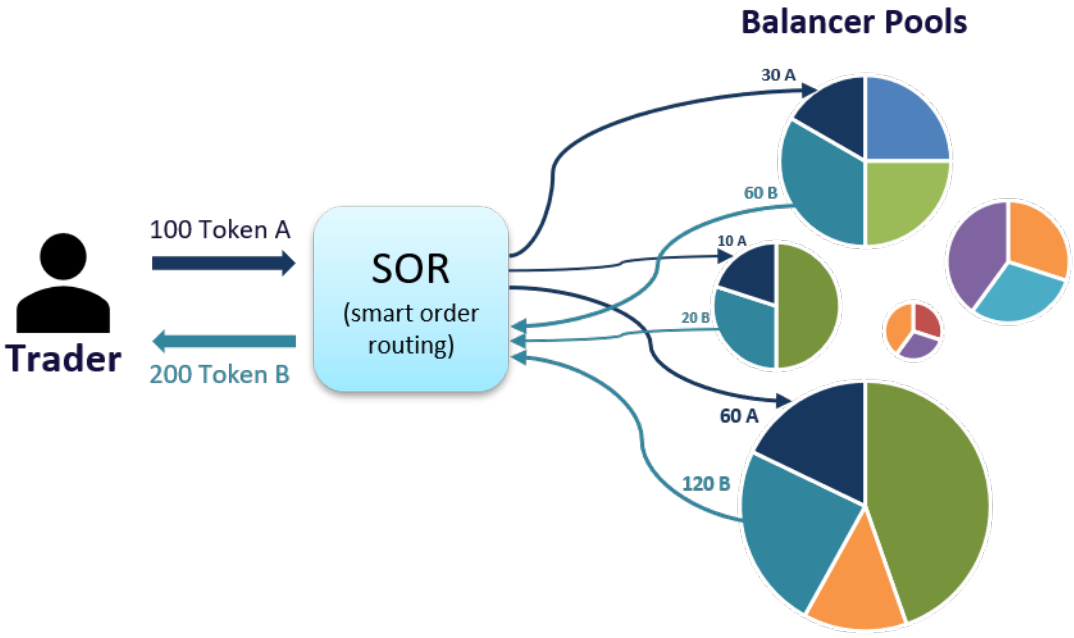

FIGURE 2. Example of trade using Balancer's smart order routing to maximize returns.

with:

$$\sum_k w_k = 1 \tag{6}$$

The value function goes strictly up with the fees charged by Balancer Pools for each trade. However, for simplicity's sake, let's assume no fees are charged and we enforce that the value function remain constant. In other words, we make V an invariant.

4.2. **Spot Exchange Rate proof.** Let's now prove that this choice of V entails Eq.1.

First of all, we know that what the trader buys, q_{out}, is subtracted from the contract's balance. Therefore $q_{out} = -\Delta Q_{out}$. Likewise, what the trader sells, q_{in}, is added to the contract's balance. Therefore $q_{in} = \Delta Q_{in}$.

Substituting in Eq.4 we get:

$$SER_{in}^{out} = \lim_{q_{out}, q_{in} \to 0} EER_{in}^{out} = \lim_{\Delta Q_{out}, \Delta Q_{in} \to 0} \frac{-\Delta Q_{out}}{\Delta Q_{in}} \tag{7}$$

This limit is, by definition, minus the partial derivative of Q_{out} in function of Q_{in}.

$$SER_{in}^{out} = -\frac{\partial Q_{out}}{\partial Q_{in}} \tag{8}$$

From the value function definition in Eq.5 we can isolate Q_{out}:

$$\begin{aligned} (Q_{out})^{w_{out}} &= \frac{V}{\left(\prod_{k\neq out, k\neq in}(Q_k)^{w_k}\right).(Q_{in})^{w_{in}}} \\ Q_{out} &= \left(\frac{V}{\left(\prod_{k\neq out, k\neq in}(Q_k)^{w_k}\right).(Q_{in})^{w_{in}}}\right)^{\frac{1}{w_{out}}} \end{aligned} \tag{9}$$

Now we use Eq.9 to expand the partial derivative in Eq.8:

$$\begin{aligned} SER_{in}^{out} = -\frac{\partial Q_{out}}{\partial Q_{in}} &= -\frac{\partial\left(\left(\frac{V}{\left(\prod_{k\neq out, k\neq in}(Q_k)^{w_k}\right).(Q_{in})^{w_{in}}}\right)^{\frac{1}{w_{out}}}\right)}{\partial Q_{in}} = \\ &= -\left(\frac{V}{\prod_{k\neq out, k\neq in}(Q_k)^{w_k}}\right)^{\frac{1}{w_{out}}} . \frac{\partial\left(Q_{in}^{-\frac{w_{in}}{w_{out}}}\right)}{\partial Q_{in}} = \\ &= -\left(\frac{V}{\prod_{k\neq out, k\neq in}(Q_k)^{w_k}}\right)^{\frac{1}{w_{out}}} . -\frac{w_{in}}{w_{out}} . Q_{in}^{-\frac{w_{in}}{w_{out}}-1} = \\ &= \left(\frac{V}{\prod_{k}(Q_k)^{w_k}}\right)^{\frac{1}{w_{out}}} . Q_{in}^{\frac{w_{in}}{w_{out}}} . Q_{out} . \frac{w_{in}}{w_{out}} . Q_{in}^{-\frac{w_{in}}{w_{out}}-1} = \frac{\frac{Q_{out}}{w_{out}}}{\frac{Q_{in}}{w_{in}}} \end{aligned} \tag{10}$$

which concludes our proof.

4.3. **Constant Value Distribution Proof.** We will now prove that:

(1) Balancer Pools maintain a constant share of value across all tokens in the pool and;
(2) These shares of value are equal to the weights associated to each token.

Let's calculate V^1, the total pool value in terms of token 1—it can really be any token in the pool since they are not necessarily ordered. Since we already know that the pool has Q_1 tokens 1, let's calculate how many tokens 1 all the other remaining tokens are worth. It does not make sense to use their Effective Exchange Rate relative to token 1 since we are not going to do any actual trade. Instead, to calculate the theoretical value we use their Spot Exchange Rates relative to token 1.

From Eq.1 we can calculate V_n^1, i.e how many tokens 1 the balance of each token n is worth:

$$V_n^1 = Q_n.SER_n^1 = Q_n.\frac{\frac{Q_1}{w_1}}{\frac{Q_n}{w_n}} = Q_1.\frac{w_n}{w_1} \tag{11}$$

We know that the total pool value in terms of tokens 1 is the sum of the values of each token in terms of tokens 1:

$$V^1 = \sum_k V_k^1 = Q_1 + \sum_{k\neq 1} V_k^1 = Q_1 + \frac{Q_1}{w_1}.\sum_{k\neq 1} w_n = \frac{Q_1}{w_1}.(w_1 + \sum_{k\neq 1} w_n) = \frac{Q_1}{w_1} \tag{12}$$

Now to calculate S_n, the share of value each token represents in the pool, all we have to do is divide the value of each token V_n^1 by the total pool value V^1:

$$S_n = \frac{V_n^1}{V^1} = w_n \tag{13}$$

which proves both that the share each token represents of the total pool value is constant and also that it is equal to the weight of that token.

5. Trading Formulae

Calculating the trade outcomes for any given Balancer Pool is easy if we bear in mind that the Value Function V must remain invariant[9], i.e. V must have the same value before and after any trade.

5.1. Amount of Token *out* for Given Amount of Token *in*.

When a user sends tokens *in* to get tokens *out*, all other token balances remain the same. Therefore, if we define q_{in} and q_{out} as the amount of tokens *in* and *out* exchanged, we can calculate the amount q_{out} a users gets when sending q_{in}:

$$\begin{aligned}
(Q_{out} - q_{out})^{w_{out}}.(Q_{in} + q_{in})^{w_{in}} &= Q_{out}^{w_{out}}.Q_{in}^{w_{in}} \\
Q_{out} - q_{out} &= \frac{Q_{in}^{\frac{w_{in}}{w_{out}}}.Q_{out}}{(Q_{in} + q_{in})^{\frac{w_{in}}{w_{out}}}} \\
q_{out} &= \left(1 - \left(\frac{Q_{in}}{Q_{in} + q_{in}}\right)^{\frac{w_{in}}{w_{out}}}\right).Q_{out}
\end{aligned} \tag{14}$$

[9]Again, this is a simplification for a Balancer Pool that does not charge any fees for trades, in practice trades are charged a fee that causes the value function to strictly increase

5.2. **Amount of Token *in* for Given Amount of Token *out*.** It is also very useful for traders to know how much they need to send of the input token q_{in} to get a desired amount of output token q_{out}. We can calculate the amount q_{in} as a function of q_{out} similarly as follows:

$$\begin{aligned}(Q_{out} - q_{out})^{w_{out}}.(Q_{in} + q_{in})^{w_{in}} &= Q_{out}^{w_{out}}.Q_{in}^{w_{in}} \\ Q_{in} + q_{in} &= \frac{Q_{out}^{\frac{w_{out}}{w_{in}}}.Q_{in}}{(Q_{out} - q_{out})^{\frac{w_{out}}{w_{in}}}} \\ q_{in} &= \left(\left(\frac{Q_{out}}{Q_{out} - q_{out}}\right)^{\frac{w_{out}}{w_{in}}} - 1\right).Q_{in}\end{aligned} \tag{15}$$

Notice that q_{out} as defined by Eq.14 tends to $SER_{in}^{out}.q_{in}$ when $q_{in} << Q_{in}$, as expected. This can be proved by using L'Hopital's rule[10] but this proof is out of the scope of this paper.

5.3. **Amount of Token *in* to New Desired Spot Exchange Rate.** For practical purposes, traders intending to use our contract for arbitrage will like to know what amount of tokens *in* – q_{in} – they will have to send to the contract to change the current spot exchange rate SER_{in}^{out} to another desired one $SER_{in}^{out\prime}$. The desired spot exchange rate will usually be the external market exchange rate and, so long as the contract spot exchange rate differs from the external market's, any arbitrageur can profit by trading with the contract and bringing the contract exchange rate closer to the external market's.

The highest profit possible by an arbitrageur is when they bring the contract spot exchange rate exactly to the external market's exchange rate. As already mentioned, this is the main reason why our design is successful in keeping track of the market prices, thus being a reliable on-chain price sensor.

It can be proved with a little math effort, which we consider out of this paper's scope, that the amount of tokens *in* – q_{in} – a user needs to trade against tokens *out* so that the pool's spot exchange rate changes from SER_{in}^{out} to $SER_{in}^{out\prime}$ is:

$$q_{in} = \left(\left(\frac{SER_{in}^{out}}{SER_{in}^{out\prime}}\right)^{\frac{w_{out}}{w_{out}+w_{in}}} - 1\right).Q_{in} \tag{16}$$

6. Specifications and Smart Contract Design

6.1. **Overview.** Here we show the main smart contracts in the system and how each actor can interact with them: pool owners can set pool balances, weights and fees; traders can swap any pair of tokens directly with one or many pools (for example using SOR).

6.2. **Functions (maybe for appendix?)** Details of input and returned params

[10] https://en.wikipedia.org/wiki/L'Hopital's_rule

6.3. **ERC20 Standard Considerations.** See this section in Set Protocol's paper: https://www.setprotoc
Interesting for us as well: - Variant ERC20 Decimals - Variant Transfer Receive Quantities
- Variant Transfer Return Values

7. References

Whitepapers, Vitalik's post about x*y = k etc.