MetaMoneyMarket:

A Protocol for Maximizing Yield

Version 0.1

July 2019

Author

Jacob Shiach, http://primeradiant.capital

Contents:

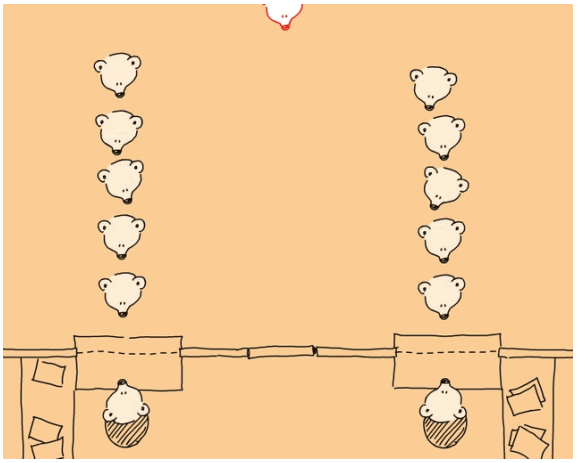
1. Introduction

MetaMoneyMarket is a protocol for maximizing yield across multiple money markets built on the Ethereum blockchain.

There are several smart-contract-based money markets in which you can deposit your tokens and then earn interest paid for by those who are borrowing your tokens. The two largest examples of this are Compound and dYdX.

The different market design decisions, yield curves, and localized demand can cause the spot interest rates to vary significantly between two markets at any one time. This makes it difficult to predict which market you should supply to in order to get the best return on your money. As what's the highest now might not be the highest for the year.

It costs gas to move funds between markets to try and chase the top rate, and unless you are a millionaire these transaction costs can surpass any potential gain from a slightly higher rate. Which can result in a scenario in which you burn a lot of gas and effort changing to the other lane, *which always seems to be faster*, just to end up with the same result as if you did nothing.

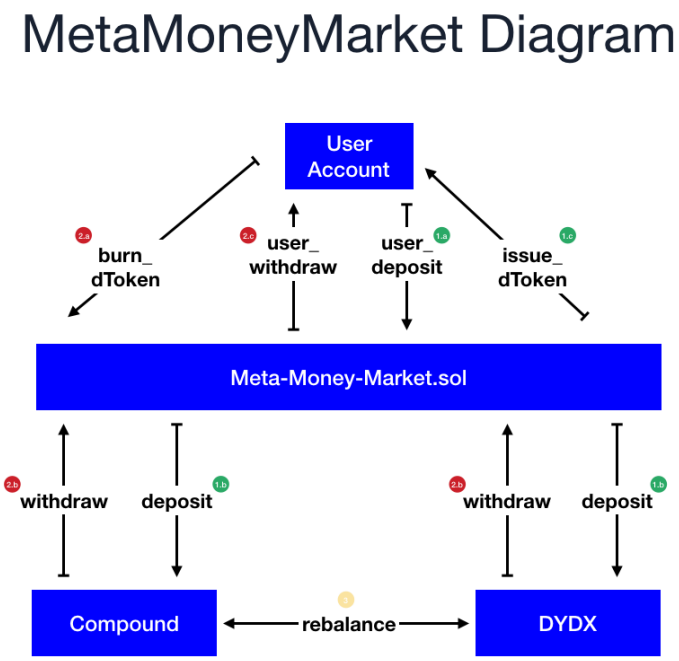


The first implementation of MetaMoneyMarket allows depositors of all sizes to earn an optimized rate by ensuring funds are deposited into the market with the best rate. And when rates move against the MetaMoneyMarket, rebalancing deposits in a cost-effective way.

Full source code is available on Github.

2. MetaMoneyMarket Contract

The MetaMoneyMarket contract is the entry point for depositing and withdrawing tokens. It works by using one or more adaptor contracts (see section #3) that allow it to interact with different markets.



Every time a user deposits some tokens, they receive some token shares that represent how much of the total deposited amount belongs to them. These token shares are minted when a user deposits and burned when a user withdraws. The minted amount is computed so that the ratio deposited tokens/minted token shares is maintained. For example, if a user deposits an amount of tokens that increase the total deposited amount in 10%, then the amount of minted tokens will be 10% of the current supply of token shares. Similarly, when a user burns some token shares, it receives an amount of tokens that preserves the ratio: if a user mints half the total amount of minted tokens, they will receive half the total amount of deposited tokens.

2.1. Deposit Token

When a user deposits some token X to the contract, to determine which Money Market to deposit tokens into, the MMM contract traverses the list of money market adapters and asks for the rate of each one using `getRate(address tokenAddress)` . It then selects the one with the highest rate and calls its `deposit(token, amount)` method.

On depositing the tokens, it will mint some "dTokens" that represent the share the user added to the supply. We'll use the `d` prefix for these minted tokens, for example, `dDAI` to represent a certain share in the amount of the DAI supply.

2.1.1.Example of token share calculation:

The MMM has a supply* of 1200 DAI and it has minted 100 dDAI so far.

- A user deposits 300 DAI. To preserve the ratio between the DAI and dDAI supply, 25 dDAIs will be minted (1200 / 100 = 1500 / 125 = 12).
- If later the same user (or a different one, it doesn't matter) wants to withdraw the equivalent of 20 dDAI, it will send them to the contract, they will be burned, and the user will receive 240 DAIs.
- The new supply of DAIs will be 1260 and the new supply of dDAI will 105.
- The ratio will then be preserved: 1260 / 105 = 12.
- With this approach, when the supply of DAI increases (because of the interest in the underlying money markets), the dDAI token will represent a larger number of DAIs.
- If the supply is 0 when a user makes a deposit, then the relation between DAI and dDAI will be 1:1.

*Note: The `totalSupply` function includes all interest accumulated

2.2. Withdraw Tokens

Withdrawing tokens is a little more involved because you can have some tokens in one money market and some tokens in the other. So `MMM.sol` withdraws from the money market with the lowest rate at the moment of the call. If there are not enough tokens, then the money market with the second-lowest rate is used next, and so on until the amount asked for withdraw is filled.

3. Adaptor Design

An adaptor is a contract that allows the MetaMoneyMarket to interact with different markets through a common interface. This common interface is described by the `IMoneyMarketAdapter` interface, and its most important methods are:

- `getRate(address tokenAddress)`: returns the supply interest rate per block in the underlying market for the given token address. This result is used by the MetaMoneyMarket contract to decide where to deposit tokens and how to rebalance.
- `deposit(address tokenAddress, uint256 amount)`: deposits the given amount of tokens in the underlying market.
- `withdraw(address tokenAddress, address recipient, uint256 amount)`: withdraws the given amount of tokens and transfers them to the specified address.

- `getSupply(address tokenAddress)`: returns the total amount of deposited tokens in the underlying market. This is used by the MetaMoneyMarket contract to know the total amount of available tokens. This total amount is used to compute the exchange rate between token shares and deposited tokens.

4. TokenShares

Each supported token has a corresponding token share, that is deployed once when support for that token is added to the MetaMoneyMarket contract. These token shares are Mintable and Burnable ERC20 tokens, and the MetaMoneyMarket is the only address that can mint new token shares. Since this is only done in its `deposit` method, it's guaranteed that the amount of tokens represented by a token share cannot be manipulated.

5. Rebalancing

Before we jump into the methods of how, let's make sure it's actually economic to rebalance. To determine if it is theoretically more profitable to rebalance vs buy and hold we must look at how the yields relate.

We can assume that in an open system arbitrageurs (and eventually this protocol) keeps the average rates between money markets roughly the same over a long enough period, because of this it's a fair assumption that any liquid enough money markets will yield equivalent return ("Ex-Post").

		Buy and Hold or Rebalance			
	Month 1	**Month 2**	**Month 3**	**Month 4**	**product**
Asset A	1.05	1.01	1.05	1.01	1.1246602500
Asset B	1.01	1.05	1.01	1.05	1.1246602500
Average	1.0300	1.0300	1.0300	1.0300	
112.466%	average of products(buy and hold)				
112.551%	Product of averages(rebalance)			*Rates are non-annualized	

If we chart an exaggerated version of that scenario(*above*) in which two markets deviate but follow a mean, we can see that the product of the averages(Rebalance) is greater than the average of the products(Buy&Hold).

Therefore, it's likely that a rebalancing strategy could lead to a higher return than holding.

5.1. Rebalance Value after Cost

At the most basic, deposits flow to the highest rate and withdrawals come from the lowest rate, so with enough activity the protocol sort of self- rebalances for free.

For active rebalancing, the simplest method would be to have a method that always moves all assets to the highest-earning Money Market. This approach however doesn't work well with Ethereum because each rebalance costs gas (~12c worth assuming 1gwei and $220ETH). So prior to rebalancing we must ensure the marginal gain in interest is worth the transaction cost.

AUM	$1,000,000	Difference in Interest Earned				
	Rates	Monthly	Daily	Hourly	Minute	Per 15s Block
Compound	10.00%	$8,333.33	$273.97	$11.42	$0.19	$0.048
DYDX	4.00%	$3,333.33	$109.59	$4.57	$0.08	$0.019
Delta	6.00%	$5,000.000	$164.384	$6.849	$0.114	$0.029

As you can see from the chart above, it would take ~$1M on deposit and roughly a difference of 6% in interest to just about break even on rebalancing 1x each minute. Since the difference between the avg monthly rates(July 2019) between compound and dydx was only 0.15% (annualized) it's necessary to take an opinionated approach to when a rebalance should be allowed.

5.2 When to rebalance?

We know that if the sum of the `Value of Deposits` multiplied by the `Delta of Rates` over *n* time period is greater than the `gas cost` then it was profitable to have rebalanced. The oracle of Delphi isn't yet on the blockchain, leaving us to solve this problem stochastically.

For the sake of brevity, I'm going to just describe a very simple potentially flawed daily rebalance solution. Using data from the last 30 days, we know that the market with the best average daily return flipped 8 times. So if we take the avg return of each rate over the last 24hrs worth of rates, and if the difference of avg daily rate changes sign, we initiate a full rebalance.

Potential Value of Daily Rebalances for 1 Yr		
@ Various AUM	Expected Value	% annualized
$100	-$8.48	-8.48%
$1,000	$18.92	1.89%
$10,000	$249.72	2.50%
$100,000	$2,989.66	2.99%

(for the sake of simplicity we assumed a static gas price)

The above table represent the additional value (minus cost) that could result from following this rebalance strategy vs. holding, assuming this past month is representative. The obvious disclaimer is these numbers aren't meant to be predictions, just to provide insight into potential practical returns.

5.3 Current Rebalance Implementation

Currently, the calculations are done off-chain and the rebalance command is sent as a list of percentages, expressed as units in 10000, indicating how to deposit the tokens in each underlying money market. The length of this array is one less than the number of money markets: the last money market will receive the remaining tokens. For example, if there are 3 money markets, and you want to rebalance so that the first one has 10.5% of the tokens, the second one 55%, and the third one 34.5%, this param will be [1050, 5500].

`function rebalance(address tokenAddress, uint256[] memory percentages)`

Note: It's percent based because if you have enough on deposit you could move the rate to your disadvantage by moving all assets over.

For now, only the owner of the contract can call a rebalance. In the next iteration of this function we would like to move the logic of the rebalance on-chain.

6. Fee structure

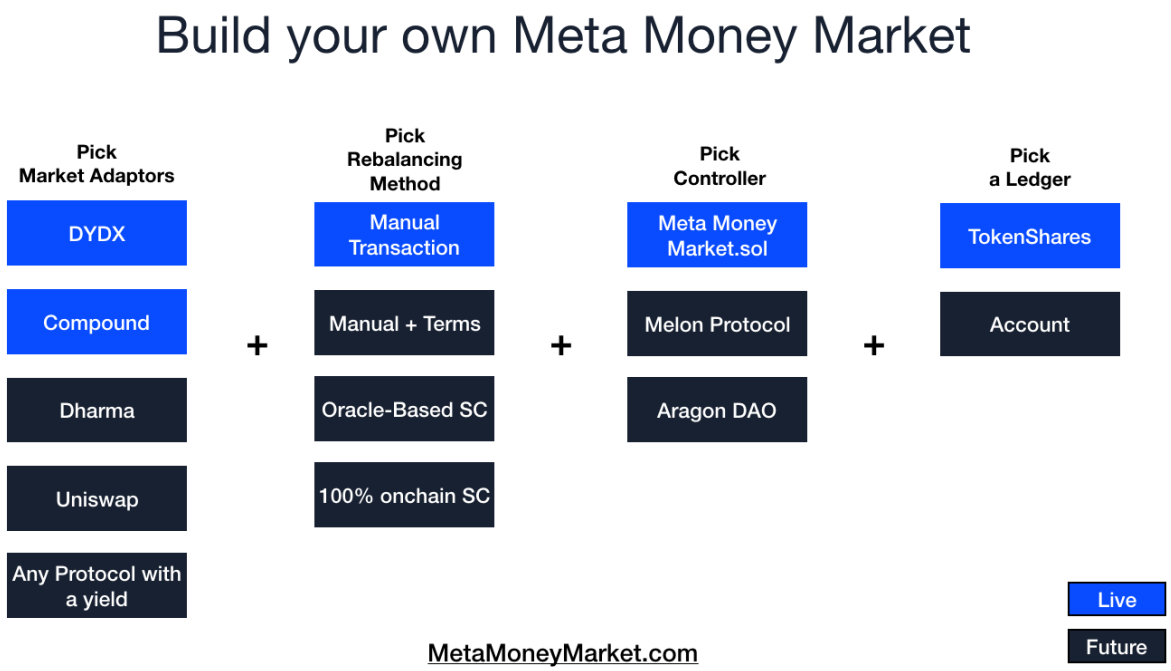
A fee structure built into MetaMoneyMarket could be useful for a few reasons: to fund rebalancing meta-transactions, depositors insurance, or sustain ongoing development. The question is how to best implement fees and do it in a way that aligns interests so that any fee is a percentage of the value created, not simply a value extraction.

Two potential options are:

1. The simplest technical and cheapest (gas) is to take a % on deposit.
    a. The only downside is an upfront fee on deposit is not the best experience. No one likes to deposit $100 into savings and instantly have less, even if it's only $0.50 less, and this might scare people from depositing.
    b. A variant of that could be to allow the fee to be rebated if deposits are held less than some period.
2. The better UX would be to charge a percentage of the interest earned leaving principal untouched. But doing that seems like it would require storing more records on-chain and it's unclear if the extra gas costs wouldn't be more expensive than a straight fee.

More exploration is needed.

7. Future Developments



In this first release, most of the effort has been geared towards a proof of concept MetaMoneyMarket deployed by us and focused on optimizing for money market returns.

It's simply not possible for a single MetaMarket to capture all of the yield generating assets, especially those with potentially negative yield. So we hope to see MMM grow beyond it's initial use case.

But the beauty of composable open-source finance is that if you need a meta-money market with a different set of adaptors, different rebalancing equation, or different ownership model, you can quickly modify and roll your own. We are especially excited about the potential of anyone, be they an individual, DAO, or Managed fund, could deploy a simplified version of MMM (mmm if you will), without the token shares overhead and with the ability to plug-n-play any adaptors in the ecosystem.

Imagine being able to invest in an on-chain fund that actively market makes on Uniswap rebalancing to Compound when price risk is too high or a DAO that's able to put its treasury to work in much the same way a corporate treasury or university endowment could.

The only thing better than a MetaMoneyMarket is multiple modular meta money markets!