Feedlot: trading with safe, low latency price oracles [*]

The Feedlot working group

> *feed* — Something supplied continuously.
> *lot* — One or more items auctioned or sold as a unit, separate from other items.

Purpose of this document This article is a feasibility study for a class of automated market makers (AMMs) that passively provide liquidity at a price based on the uniform clearing price (UCP) of a competitive batch auction.

It is not a whitepaper, specification, or proposal, and it makes no claim to exhaustiveness or completeness on any of the topics discussed — particularly security. Would-be implementers of a feedlot AMM are advised to pursue thorough further investigations of these issues in the context into which they hope to deploy.

[*]This research was funded by the Gnosis DAO.

1 Introduction

Feedlot AMMs should satisfy the following constraints:

- Feedlot AMM LPs should enjoy cheap portfolio management, some yield, and protection from adverse selection.
- Feedlot AMM traders should enjoy low, predictable fees, control over execution time, and favourable prices at least for trades in the 'correct' direction.
- It should not be economical to manipulate the UCP of the CoW batch auction in order to trade at favourable prices on feedlot.

  Moreover, it should not be possible to substantially offset the cost of manipulating the CoW UCP (for any reason) by trading on a feedlot AMM.
- Wherever possible, feedlot should use incentive-compatible mechanisms to ensure correct operation, without needing to fall back on social adjudication procedures.

2 General design principles

2.1 Definitions

Our blockchain model is based on Ethereum, and should apply to any blockchain-based state machine with similar principles. In particular we point out the following assumptions:

- Accounts model that tracks token balances with a common interface (e.g. ERC20) and such that token transfers satisfy usual invariants (i.e. transfers must preserve total supply and balances cannot be negative).
- Transactions (a.k.a. messages calls) must be initiated by an off-chain entity.
- Transactions, once committed, cannot be rolled back. That is, we do not consider the risk of blockchain forks.

By a *smart contract system* we mean any kind of on-chain entity or aggregation of entities. In particular, it may mean a single smart contract (e.g. a Uniswap pool) or a structured collection of interacting smart contracts (e.g. Uniswap as a whole).

- *Liquidity pool.* Smart contract system that custodies the assets of *liquidity providers* and tracks withdrawal liabilities.
- *Liquidity provider (LP).* An agent that deposits funds in a liquidity pool.
- *Automated market maker (AMM).* Smart contract system that provides pricing and passively settles orders by trading against a liquidity pool.
- *Batch auction.* Order settlement system that accumulates orders in a buffer and settles them with a uniform clearing price. Prices are supplied by *solvers* who compete to optimise an objective function defined in terms of the price vector and the set of orders.
- *Uniform clearing price (UCP).* Price vector against which a batch auction is settled.
- *Solver.* Agent that provides quotes in competition to settle a batch auction.
- *Constant function market maker.* AMM whose pricing function depends only on reserves.
- *No-arbitrage price.* A price p such that it is not possible to to make a

In this article, the pricing provided by an AMM will be a function of state at the time of settlement. In principle it is possible for it to also depend on calldata in the message that triggers the settlement. For simplicity, we will mainly consider only pools with two assets A and B whose balances are denoted $(x, y) \in [0,\infty)^2$. We take token A for the numéraire (also called the 'quote token'), so that prices are for token B in terms of token A. If a trade occurs of Δx A tokens for Δy B tokens, the execution price is $\Delta x/\Delta y$.

2.2 Constant function market makers

By far the most widely used type of AMM is a constant function market maker parametrised by a function $f(x,y)$ of the pool reserves. The associated marginal pricing function is

$$p(x,y) = \frac{f_x(x,y)}{f_y(x,y)}.$$

If reserves are in state (x_0, y_0), the amount of B tokens paid out in exchange for Δx A tokens is

$$\Delta y = \int_{x_0}^{\Delta x} p(x, y(x))dx - y_0$$

where $y(x)$ is determined by the corresponding differential equation $y'(x) = p(x, y(x))$ and the initial conditions $y(x_0) = y_0$.

A practical implementation of market orders on a CFMM must have a computationally efficient algorithm for computing Δy given Δx. On the other hand, to decide if a given swap $(\Delta x, \Delta y)$ will be accepted by a CFMM, it is enough to compute the invariant $f(x + \Delta x, y + \Delta y)$.

2.3 Liquidity provider costs

The CFMM quote is not automatically updated when changes occur on other markets. Rather, price updates are supplied by arbitrageurs who trade on the CFMM in such a way as to push the pricing into alignment with external markets.

This trading activity also has the effect of adjusting the balance of assets in the pool, computed in terms of external market prices, towards an equlibrium that depends on the invariant. For example, with Uniswap's constant product formula $f(x,y) = xy$, the equilibrium balance is 50-50, that is, $px = y$.

LPs effectively pay for this adjustment service by trading at unfavourable prices. The amount they pay depends only on the external price movement. In particular, there is no room for price updaters to compete with one another on the basis of reducing cost to LPs; rather, all arbitrageur competition is at the infrastructure layer and excess rewards tend to be absorbed either by consensus nodes or by middleware services that assemble blocks or partial blocks. $\boxed{\text{cite}}$

2.4 Automated market makers with a price feed

The issue of uncontrolled LP losses to adverse selection could be alleviated if price updates from external markets could be supplied by another channel, so that the marginal pricing function could depend on inputs other than reserves:

$$p = p(x, y; p_{\text{ext}}).$$

In blockchain applications, such channels are called *price oracles.* The most obvious choice for such a function is $\boxed{\text{cite}}$

$$p(x, y; p_{\text{ext}}) = p_{\text{ext}},$$

so that the AMM simply quotes the oracle price. If we black-box the data source and assume that it provides no-arbitrage prices, then such an AMM would indeed be protected from adverse selection.

However, as is well-known, oracles in the field are fallible. They are vulnerable to bugs and deliberate attacks at each stage of the data pipeline. Oracles that provide prices that anyone can trade against are a particularly attractive vector for sabotage, both because of the potential for financial gain and because, in the case of a manipulated data source, of the difficulty in establishing that an attack has even occurred. $\boxed{\text{cite}}$

3 Architecture

Our model for a batch auction system is the CoW protocol on Ethereum. The CoW protocol consists of a set of off-chain entities that we collectively refer to as *CoW services* and a system of Ethereum smart contracts called the *CoW contracts.* The details of the internal architecture of these aggregations is not discussed here.

Schematically, the CoW algorithm runs as follows:

1. Traders send orders to an off-chain order book server where they are tracked in a database.
2. A set of offchain entities called *solvers* query the database and attempt to construct a *solution*, that is, an Ethereum transaction that collectively settles all user orders at a fixed price vector $\vec{p}$ (the uniform clearing price). This transaction may contain arbitrary Ethereum message calls.
3. Every batch interval, solutions are validated by simulating against a recently observed chain state and ranked according to a utility function. The utility function measures the total marginal utility of user orders filled by the solution in terms of the difference between the limit price and the fill price. Utilities of orders denominated in different tokens are aggregated using prices on external markets.

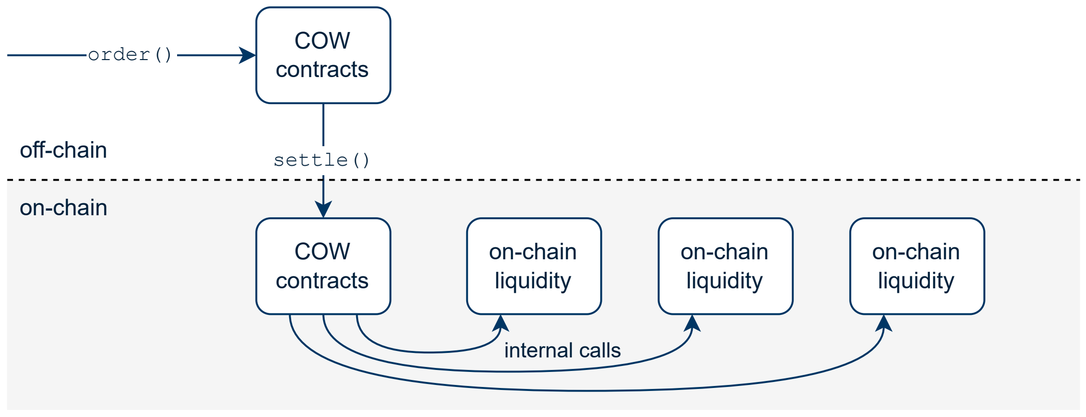

Figure 1: CoW procotol execution. CoW services accumulate orders and batches are settled (at most) every 30 seconds.

4. The solver with the winning solution calls the `settle()` function on the CoW settlement contract, which executes their solution.

5. Misbehaviour is assessed socially. Slashing punishments can be triggered by a vote of the CoW DAO.

An AMM whose quote depends on the UCP of the CoW batch auction needs to have at least the following structures:

- A way for traders to submit orders.
- A way for would-be LPs to add or remove liquidity.
- A communication channel $\mathcal{O}$ connecting Feedlot with the CoW contracts or services along which can be communicated the UCP.

Since the last item is the distinguishing feature of Feedlot AMMs, in this section we focus on the design of this channel.

3.1 Synchronisation

A batch auction system needn't be synchronised with the blockchain. There may not be an auction every block, and even if there is, not every token pair traded on the BAS need necessarily appear in the batch. This is the case for CoW: auctions are roughly every 30 seconds, and sometimes no auction occurs for hours at a time. Therefore, a fresh batch auction price might not be available at the point that a trader wishes to trade on feedlot.

If the Feedlot quote price is to depend on the price of a CoW batch auction, then, there are only two possibilities:

1. The quote function uses a stale CoW price (poll model).

2. The market only settles trades at the same time as the batch auction (subscription model).

These two options are analogous to what in the blockchain oracles literature have been called the *pull* and *push* models, respectively **heiss2019oracles**; **muhlberger2020foundational**.

In either case, CoW services can guarantee zero latency updates of the UCP by writing into the channel atomically together with the batch execution transaction. This would require minor modifications to CoW's offchain components so that they call a wrapper contract that both triggers the CoW settlement and publishes a price to the on-chain oracle. It does not require any changes to CoW's onchain components. Whether the UCP is updated before or after the settlement affects only how Feedlot behaves for orders placed in the winning solution itself, with the details depending on the synchrony model for Feedlot; see §3.2. Orders that arrive at Feedlot outside the batch are not affected by this ordering.

What could be considered a third option is to fall back on another algorithm in case no price is available. Since that essentially amounts to not trading on Feedlot, and could easily be implemented at a higher layer, we don't discuss that here.

3.1.1 Poll model

In this model, CoW writes the UCP vector to the channel every time there is a batch. Meanwhile, Feedlot AMMs query the price whenever they have to settle an order. There is no synchronisation between these two tasks. One write is needed per batch per product, and one read is needed per trade on Feedlot. These operations being relatively cheap, it would be reasonable to implement this type of channel as an on-chain oracle. A variant approach is to aggregate UCPs to produce, for example, time-weighted average prices. This is the approach taken by Uniswap's oracle service **adams2020uniswap**.

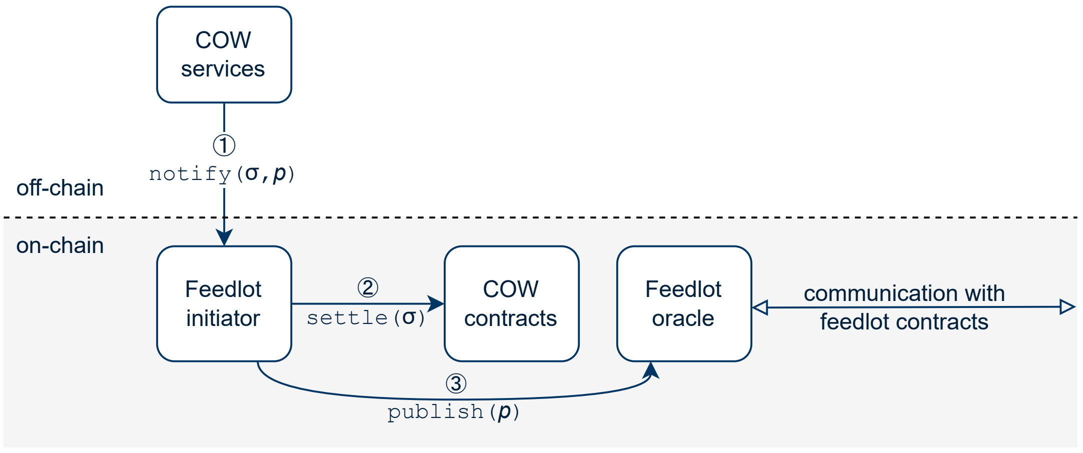

Figure 2: Atomic execution of the batch solution and publication of the UCP. The ordering of (2) and (3) can be reversed.

[PLACEHOLDER]

Figure 3: Polling oracle. Feedlot AMM executions are not synchronised with CoW batches.

3.1.2 Subscription model

In this approach, orders arriving at Feedlot are accumulated in an order book. All trades are executed when there is a CoW batch; for atomicity guarantees, the CoW batch and the Feedlot queue should be executed within the same message call.

As with any order book based protocol, the order book itself would potentially be quite expensive to maintain on the Ethereum chain. It is beyond the scope of this report to discuss approaches for optimising trust model and cost-effectiveness of off-chain order books. Pragmatically, a reasonable approach would be to maintain the order book in the same place as the order book for the CoW batch auction, which at time of writing is a set of servers maintained by the CoW team.

3.2 Can the batch itself settle on Feedlot?

Certain configurations would allow the solution to both settle orders on Feedlot and quote it a price. The solver's objective is to get the most favourable price possible for its users, so naturally it will quote the least favourable price that Feedlot will accept. Naturally, allowing an entity to trade against the AMM at any price it chooses is trivially exploitable to the detriment of LPs, and so special care has to be taken to address this case.

3.2.1 Poll model

In the poll-based model,

- If the UCP is updated *after* the batch, the solution trades with a stale price.
- If the UCP is updated *before* the batch, the solution trades at the price quoted by the solver.

The interesting case is the second one. Assuming solutions are simulated exactly as they would run on-chain, a solver could quote any price at all and have it execute successfully against Feedlot. The solver's objective is to get the most favourable price possible for its users, so naturally it will quote the least favourable price that Feedlot will accept. Naturally, allowing an entity to trade against the AMM at any price it chooses is trivially exploitable to the detriment of LPs. Thus, if this approach were to be pursued, a special mechanism would have to implemented where Feedlot is specifically excluded from simulation.

If Feedlot's pricing function is bounded below independently of the UCP — for instance if it imposes a lower bound from a hardcoded preference curve or aggregate of external market prices — then this actually eliminates the dependence of pricing on the UCP.

[PLACEHOLDER]

Figure 4: Subscription oracle. Feedlot AMM executions are triggered when a new price is published.

3.2.2 Subscription model

Batch solutions must settle immediately, so the success of the solution cannot depend on trading on an AMM that enqueues orders for execution at a later time. However, a batch can be used to atomically execute any message call. A solver could therefore use the following strategy:

1. Fill orders using private liquidity.
2. Enqueue orders in the opposite direction on Feedlot.

Financially, this amounts to hedging the trade in step 1. with a forward contract with expiry at the next batch. In this way, Feedlot could act as a kind of repo market for solvers.

4 Cryptoeconomic security

A price oracle is a data source $\mathcal{O}$ from which a smart contract may query prices for a market order

$$p \leftarrow \mathcal{O}(t_{\text{in}}, t_{\text{out}}, \sigma, r)$$

where $t_{\text{in/out}}$ denotes the in/out token, $\sigma \in \{\text{BUY}, \text{SELL}\}$ is the direction, and r is the amount (of t_{in} for SELL orders, t_{out} for BUYs). The oracle is said to be *truthful* if the prices thereby obtained are free from arbitrage.

Assuming correct implementation, oracles can fail in two ways:

- Inaccurate or unreliable data source.
- Failure to faithfully communicate the value from the data source.

In the present context, fidelity of the communication of value is guaranteed by non-falsifiability of on-chain execution.

4.1 Truthfulness of pricing data

We say that the price p is *realisable* at a volume V of the in token and chain state ϕ if it is possible to buy V A tokens for pV B tokens in state ϕ (for whom?). This should be a realisation of the notion of 'truthfulness' discussed in **heiss2019oracles**.

If CoW solver prices are not truthful, the solver must settle the trade with their own private liquidity held in bond by CoW DAO. Hence, a posteriori, CoW solver prices are truthful for the solver at the state immediately before the batch and at the volume traded in the batch.

It is worth noting that merely simulating executions on-chain (and reverting all changes at the end) is sufficient to verify realisability of the price vector.

4.2 Manipulation resistance

A price oracle based trading system is Δ-*manipulation resistant* if

$$V \cdot (\tilde{p} - p) + \Delta < C(\mathcal{O}; p, \tilde{p})$$

where $C(\mathcal{O}; p, \tilde{p})$ is the cost to effect a change in the oracle price from p to $\tilde{p}$. In order to make statements about this notion of manipulation resistance, it is necessary to at least be able to lower bound the cost of manipulation with a concrete model.

It is important to note that, intuitively, a manipulated price may still be *realisable*: a manipulator genuinely offers liquidity at off-market prices in order to effect certain behaviour in systems that depend on price feeds from these markets. It is generally not straightforward to even define price manipulation in general, as this requires insights into the utility function of the alleged manipulator.

Hence, for D-manipulation resistance, at least the following inequality must be satisfied:

$$\Delta_p \cdot \text{vol}_{\text{COW}} > \Delta_p \cdot \text{vol}_{\text{feedlot}} + D$$

where:

- Δ_p is the difference between the fair market price and the manipulated price.
- vol_{COW}, resp. $\text{vol}_{\text{feedlot}}$ is the total volume on COW, resp. Feedlot, for the pair.
- P_{ext} is the profit to be made from manipulating the price by Δ_p on external markets.

It is not feasible to estimate $P_{\text{ext}}(p)$ for all time, and the developers of COW protocol can not reasonably expect to have any control on how COW price feeds are used in offchain markets. The best we can hope to do is to ensure that

$$\text{vol}_{\text{COW}} \gg \text{vol}_{\text{feedlot}}$$

and that it is difficult to guarantee atomic execution on both the COW batch and feedlot markets in opposite directions.

5 The inventory problem

Apart from trading fees and price updates, Uniswap or Balancer arbitrageurs provide a kind of portfolio management service to LPs. This approach to portfolio management may be highly efficient, but it suffers from some problems:

- There is no room for competition on price to deliver this service: the cost to LPs is fixed in terms of the CFMM invariant, the initial reserves, and the price change. Arbitrageurs compete to rebalance reserves on the basis of speed or bribes; the proceeds of this competition go to infrastructure providers rather than LPs.
- Portfolio management and price updates are bundled together in the same operation. There is no scope for these services to be priced or managed separately.

So far, we have not discussed any analogous mechanism for feedlot LPs.

6 Conclusion

This was a success.

References

adams2020uniswap Hayden Adams, Noah Zinsmeister, and Dan Robinson. *Uniswap v2 core*. 2020. URL: `https://uniswap.org/whitepaper.pdf`.

heiss2019oracles Jonathan Heiss, Jacob Eberhardt, and Stefan Tai. "From Oracles to Trustworthy Data On-Chaining Systems". In: Atlanta, GA, USA. Atlanta, GA, USA: IEEE, 2019, pp. 496–503. ISBN: 978-1-7281-4694-2. DOI: `10.1109/Blockchain.2019.00075`.

muhlberger2020foundational Roman Mühlberger et al. "Foundational oracle patterns: Connecting blockchain to the off-chain world". In: *Business Process Management: Blockchain and Robotic Process Automation Forum: BPM 2020 Blockchain and RPA Forum, Seville, Spain, September 13–18, 2020, Proceedings 18*. Springer. 2020, pp. 35–51.