了。趁着这个机会就先写一个 Uniswap v3 设计与实现吧。 因为 v3 版本的实现复杂度和 v2 已经不在一个量级了,难免会有理解上的偏差,本文权当是抛砖引 玉,也希望有更多的人参与讨论。因为实现比较复杂,本系列会拆分成多篇文章,持续更新。

刚看完 Uniswap v2 的代码,本来打算写一个 Uniswap v2 设计与实现,结果 Uniswap v3 就发布

本文假定读者都能理解 AMM 的基本概念, 并且阅读过 v3 的实现细节 (最好读过白皮书) 来撰写 的,因此不会具体的解释每一个概念的实现逻辑。 前置阅读资料: • 官方博客 • v3 白皮书

设计原理

官方的白皮书已经比较详尽的描述了 v3 的设计原理,这里仅对白皮书中的内容做一些补充,包含本 人对其中一些机制的理解和思考。 Uniswap v2 版本使用 $x\cdot y=k$ 这样一个简洁的公式实现了 AMM Dex, 正是由于其简洁易用性, 使

坚实的幻想

其在短短的一年的时间内迅速成长为 DeFi 领域的龙头项目。但是随着 DeFi 生态走过了「从无到

有」的阶段,因为 v2 无法满足某些特定需求,从而诞生了 Curve, Balancer 这些针对某些功能进行 改进的 AMM。 简单来说, 官方认为 v2 版本最大的痛点是资金利用率 (Capital Efficiency) 太低, v3 版本在解决这 个问题的同时,还带了了新的改进,总体总结如下:

• 可灵活选择价格区间提供流动性 • 更好用的预言机 order book 功能

提升资金利用率

• 灵活的费率

v2 的问题 我们先来看一看 v2 版本的资金利用率是怎样的,假设 ETH/DAI 交易对的实时价格为 1500 值:

DAI/USDC 范围内波动。

解决资金利用问题之前,我们可以观察到大部分的交易对的价格,在大部分时间内都只是在一个固定 范围内波动。例如 ETH/DAI 交易对,在近一个月时间内都是在 1300~2200 DAI/ETH 这个范围内波

动。更极端的例子是 DAI/USDC 这样的稳定币交易对,在大部分时间内都只是在 1.001~1.002

解决方案

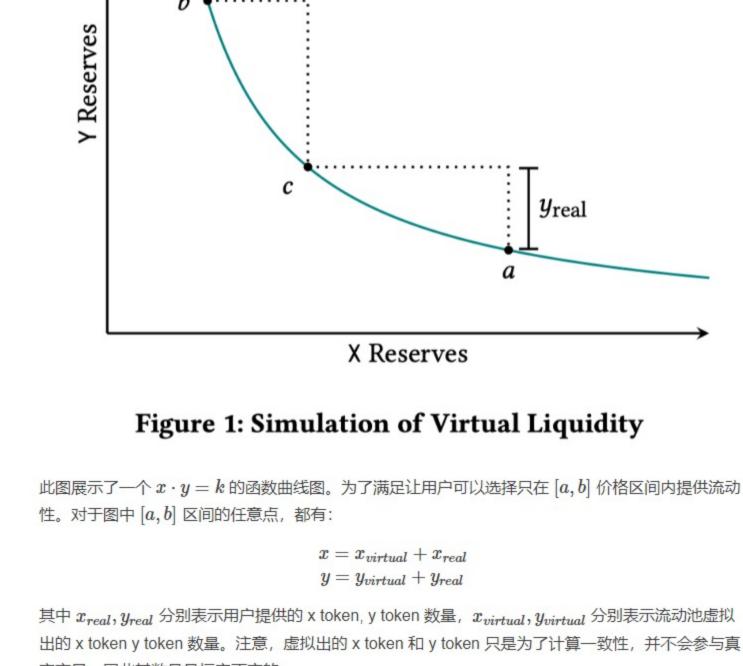
DAI/ETH, 交易对的流动性池中共有资金: 4500 DAI 和 3 ETH, 根据 $x \cdot y = k$, 可以算出池内的 k $k = 4500 \times 3 = 13500$

假设 x 表示 DAI,y 表示 ETH,即初始阶段 $x_1=4500, y_1=3$,当价格下降到 1300 DAI/ETH 时: $\begin{cases} x_2 \cdot y_2 = 13500 \\ \frac{x_2}{y_2} = 1300 \end{cases}$

得出 $x_2=4192.54, y_2=3.22$,资金利用率为: $\frac{\Delta x}{x_1}=6.84\%$ 。同样的计算方式,当价格变为 2200 DAI/ETH 时,资金利用率约为 21.45%. 也就是说,在大部分的时间内池子中的资金利用与低于25%.这个问题对于稳定币池来说更加严重。

v3 版本的解决方案是允许用户只在一段价格区间内提供流动性。如下图:

 virtual reserves $x_{\rm real}$



实交易, 因此其数量是恒定不变的。 当流动池的价格来到用户设置的零界点时 (例如图中的 a 点或者 b 点) , 用户实际提供的 x token 或

者 y token 将为 0, x 或 y 将完全由虚拟 token 组成。为了保证虚拟 token 恒定不变,当价格进一步 变动,移动到用户设定的价格区间之外时,流动池会移除这部分流动性,以保证虚拟的 x token 或 y token 数量不会减少,因此这部分虚拟的 token 只会在价格处于设定的区间内时参与价格的计算,而

不会真的参与流动性提供 (即虚拟 token 数为常数,并不会发生数量变化)。

例如,当价格到达 a 点时,用户的所有资金转换为 x,此时 $y_{real}=0, y=y_{virtual}$,当价格继续降 低时,流动池将移除这部分流动性。用户的资金状态将停留在 a 点,直至价格再次回到 a 点并进入 [a,b] 价格区间。 通过这样的设计,用户的资金只会在 [a,b]价格区间内提供流动性,并且因为虚拟 token $x_{virtual}, y_{virtual}$ 的参与,这部分流动性也满足 $x \cdot y = k$ 公式,计算价格的方式并没有产生变化。

virtual reserves (2.1) real reserves (2.2) Reserves

> X Reserves Figure 2: Real Reserves 上图展示了用户选择在价格 [a,b] 之间提供流动性时,通过虚拟 token 的参与,将曲线 f(real) (橘 红色)向右上方移动至 f(virtual) (绿色) ,实现了价格计算的一致性(即满足 $x\cdot y=k$)。 **》流动性聚合** 上面我们说了,通过引入虚拟 token 的概念,让用户可以在某一个价格区间内提供流动性。而每一个 用户提供的流动性都可能设置不同的价格区间,这样一来一个交易对的池子中就包含了多个不同的流 动性。因此从单个交易池的视角来看,Uniswap v3 实际上扮演的角色是一个交易聚合器:当发生交 易时,此交易会拆分成多个,通过池中多个不同的流动性来进行交易,最后将交易结果聚合,完成最

的流动性)。因此 v3 的交易过程会围绕价格来进行,这样可以保证所有流动性的价格一致。事实上 这就和 AMM 交易聚合器的行为一致,因此我们可以把 Uniswap V3 理解成单个交易池中不同流动性 的交易聚合器。

交易过程 v2 版本

在 v2 版本中,用户与一个交易对发生交易时,假设用户提供 x token,资金量为 Δx ,AMM 需要计

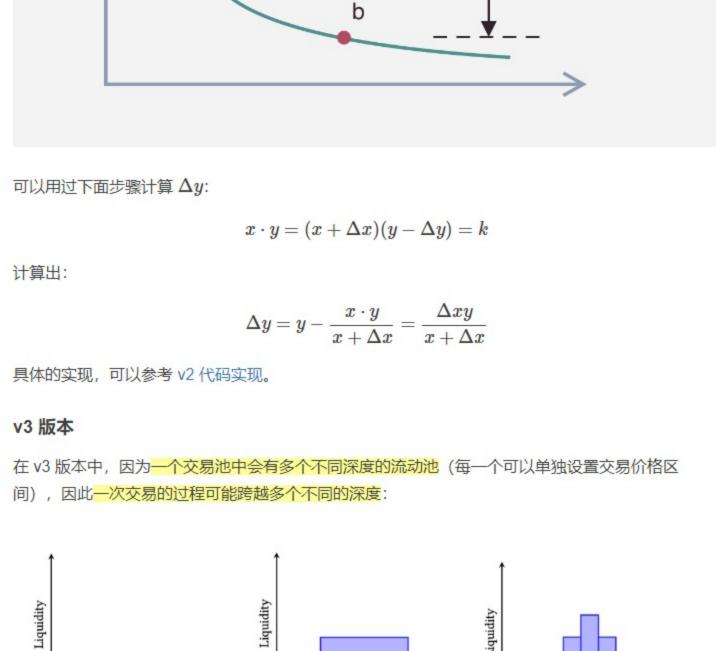
算出用户可以得到的 y token,即 Δy . 如下图所示,池中资金从 a 点随着曲线移动到 b 点:

从交易聚合器的角度看,在交易发生前,池中每一个流动性中的 token price 是一致的。那么我们需

要让交易结束后,池中每一个流动性中的 token price 仍然是一致的 (当然这里仅包含所有在区间内

终的交易过程。

а



的关系不能像 v2 版本一样用一个平滑的 bonding curve 曲线来表示。那么如何计算交易结果呢? 在前面我们说过,V3 的行为类似一个交易聚合器,它需要保证池中所有流动性的价格在交易前后一 致。因此 V3 会围绕池中的代币价格来进行。 对于一个流动性来说,流动性大小可以用 k 表示,即 $k=x\cdot y$,用 P 表示 x 的价格,即 $P=rac{y}{x}$ 对于一个流动性来说,当使用 x token 交换 y token 时,我们需要进行如下计算: • 交易至指定价格 (不可以超出此流动性的边界价格) P, 需要的 x token 数 Δx , 可以获得的 ytoken 数 Δy • 给定 x token 数 Δx (假设不会引发价格超出此流动性的边界价格) ,可以获得的 y token 数

 p_a

Price

(I) Uniswap v2

 Δy , 以及最终的价格 P

格所在区间内所有流动性大小的总和。

进行计算。这样会导致:

tick 管理

动性)。

灵活的手续费选择

手续费与 tick 的关系

uniswap-v3-core

uniswap-v3-periphery

这些合约间的关系大致如下图:

uniswap-v3-periphery 开始。

本系列所有文章:

• Uniswap v3 详解 (一) : 设计原理

• Uniswap v3 详解 (三): 交易过程

• Uniswap v3 详解 (六) : 闪电贷

Uniswap v3 详解(四): 交易手续费

• Uniswap v3 详解 (五): Oracle 预言机

PREVIOUS

使用 NEOVIM 打造一个现代化的编辑器

• Uniswap v3 详解 (二) : 创建交易对/提供流动性

core 仓库的功能主要包含在以下 2 个合约中:

peirphery 仓库的功能主要包含在以下 2 个合约:

池, 分别对应 0.05%, 0.30%, 1.00% 的手续费。

点中的某一个作为流动性的边界价格。

当 k 值不变时,根据定义:

 p_b

Price (III) A collection of custom positions

Price

(II) A single position on $[p_a, p_b]$

Figure 3: Example Liquidity Distributions

如上图最右边所示, 当价格变化时, 流动池中的总流动性也会随之变化。因此 v3 版本流动池中资金

可以推导出: $\left\{ egin{aligned} x = \sqrt{rac{k}{P}} \ y = \sqrt{kP} \end{aligned}
ight. \Rightarrow \left\{ egin{aligned} \Delta x = \Delta rac{1}{\sqrt{P}} \cdot \sqrt{k} \ \Delta y = \Delta \sqrt{P} \cdot \sqrt{k} \end{aligned}
ight.$

这样一来计算过程并<mark>不需要关注流动性中的 x token 和 y token 余额</mark>,通过 k 值和价格 P 就可以完 成交易过程的计算。 为了减少计算过程中的开根号运算,v3 合约直接存储 \sqrt{P} 的值,同时合约中没有存储 k 的值而是 存储 $L=\sqrt{k}$,通过 L 来表示池中当前的流动性大小(存储 L 还有一个好处是减少溢出的可能 性)。

在实际交易过程中,一个交易会通过多个流动性聚合完成。因此上述的公式会进行聚合完成,即使用

当前价格上的流动性总和来进行计算,流动性总和可以这样表示: $L_{total} = \sum L_{user}$, 即将当前价

同时, 一个交易还可能跨越不同的流动性阶段(即可能超出或者进入某个流动性), 因此合约需要维

护每个用户提供流动性的价格边界,当价格到达边界时,需要在总流动性上增加或移除对应流动性大 小。通过分段计算的方式完成交易结果的计算,具体的实现过程可以参考: Uniswap v3 详解 (三): 交易过程 价格精度问题

因为用户可以在任意 $[P_0,P_1]$ 价格区间内提供流动性,Uniswap v3 需要保存每一个用户提供流动性

的边界价格,即 P_0 和 P_1 。这样就引入了一个新的问题,假设两个用户提供的流动性价格下限分别

是 5.00000001 和 5.00000002, 那么 Uniswap 需要标记价格为 5.00000001 和 5.00000002 的对应

的流动性大小。同时当交易发生时,需要将 [5.00000001,5.00000002] 作为一个单独的价格区间

• 几乎很难有两个流动性设置相同的价格边界,这样会导致消耗大量合约存储空间保存这些状态 • 当进行交易计算时, 价格变化被切分成很多个小的范围区间, 需要逐一分段进行计算, 这会消

Uniswap v3 解决这个问题的方式是,将 $[P_{min},P_{max}]$ 这一段连续的价格范围为,分割成有限个离

散的价格点。每一个价格对应一个 tick,用户在设置流动性的价格区间时,只能选择这些离散的价格

Uniswap v3 采用了等比数列的形式确定价格数列,公比为 1.0001。即下一个价格点为当前价格点的

耗大量的 gas, 并且如果范围的价差太小, 可能会引发计算精度的问题

100.01%,前面我们说过 Uniswap v3 实际存储的是 \sqrt{P} ,那么下一个价格与当前价格的关系为 $\sqrt{P_{next}} = \sqrt{1.0001} \cdot \sqrt{P_{current}}$ 如此一来 Uniswap v3 可以提供比较细粒度的价格选择范围(每个可选价格之间的差值为 0.01%), 同时又可以将计算的复杂度控制在一定范围内。

 $i = \log_{\sqrt{1.0001}} \sqrt{p}$ 这样一来,只需要通过整数索引i就能找到对应的tick,并且i为0时价格为1. Uniswap 不需要记录每个 tick 所有的信息,只需要记录所有作为 upper/lower tick 所包含的流动性元 数据即可。看下面这个例子:

有一个价格为 1 的 tick。将所有 tick 通过索引来表示,定义整数 i 表示 tick 的索引:

a $\Delta L = 500$ $\Delta L = 700$ $\Delta L = -700$

两个用户分别在 [a,c] 和 [b,d] 两个区间提供了流动性 L_1 和 L_2 ,对于 Uniswap 来说它 $\frac{1}{2}$ 会在 a ,b ,c ,

d 四个 tick 上记录对应的流动性增减情况。例如当价格从图中从左向右移动时,代币池的流动性需要

做对应的增减(即从左侧 tick 进入一个流动性时增加流动性,移出流动性的右侧 tick 时减去相应的流

v3 版本内置了三种梯度的手续费率 (0.05%, 0.30%, and 1.00%) , 同时可以在未来增加更多的费率

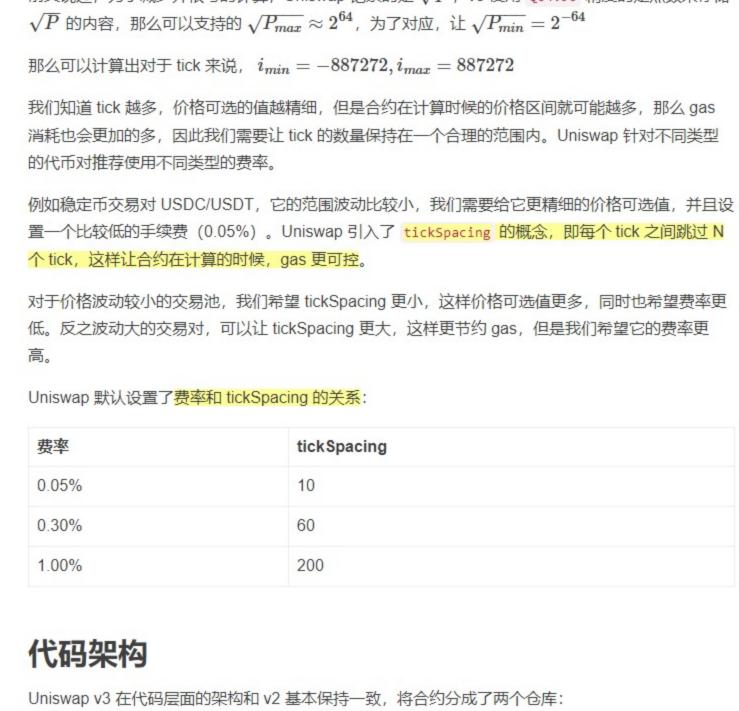
值。关于手续费的计算过程,这部分放在后文来详解(链接:交易手续费)。需要注意的是,由于需

要支持多种费率,同一个代币对 v3 版本会有多个不同的流动池。例如 ETH/DAI 代币对,会分成三个

更多的费率选择性,这样做会更加灵活,但是同时也会带来一定的流动性分裂, uni 官方表示后续可

以通过治理添加更多的费率可选值,这也势必会让流动性更加分裂。那么可能会出现一种情况是,即

 $\Delta L = -500$



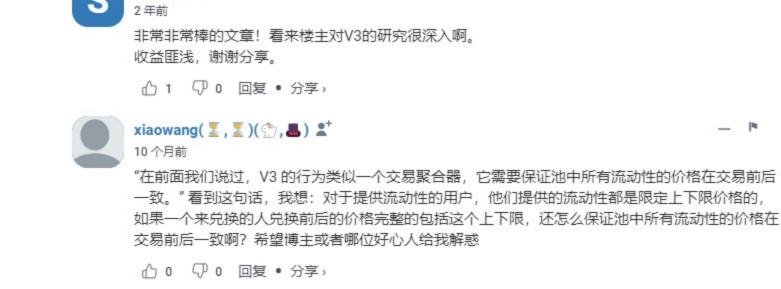
• SwapRouter: 提供代币交易的接口,它是对 UniswapV3Pool 合约中交易相关接口的进一步封 装, 前端界面主要与这个合约来进行对接。 • NonfungiblePositionManager: 用来增加/移除/修改 Pool 的流动性, 并且通过 NFT token 将 流动性代币化。使用 ERC721 token (v2 使用的是 ERC20) 的原因是同一个池的多个流动性并 不能等价替换 (v3 的集中流性动功能)。

• UniswapV3Pool: 实现代币交易,流动性管理,交易手续费的收取, oracle 数据管理。接口的 实现粒度比较低,不适合普通用户使用,错误的调用其中的接口可能会造成经济上的损失。

• UniswapV3Factory: 提供创建 pool 的接口,并且追踪所有的 pool

Update 05-23 本系列文章主要参考的是 Uniswap v3 3月底的代码,已经和其最新代码又一定差异,但是这部分差 异不大,并不会影响主体业务逻辑的理解。 Uniswap v3 详解系列

加入讨论... 或注册一个 DISQUS 帐号 ? 通过以下方式登录



1年前

Stefans 2

Xiaoxue Zhang 2 *

△ 0 ♀ 0 回复 • 分享,

△ 0 ♀ 0 回复 • 分享,

Paco 管理员 ♣ → TangHaoSuan

TangHaoSuan 2*

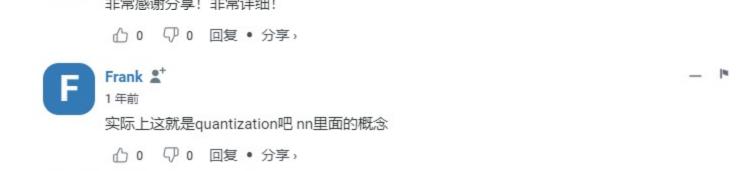
FEATURED TAGS

Ethereum Uniswap DeFi

好文

这个我还不太熟悉

△ 0 ♀ 0 回复 • 分享,





写得真好,近期看到写得最好最详细的Uniswap V3的文章

- CATALOG

设计原理

提升资金利用率

ARCHIVES

HOME

ABOUT

v2 的问题 解决方案 流动性聚合 交易过程 v2 版本 v3 版本 价格精度问题 tick 管理 灵活的手续费选择 手续费与 fick 的关系 代码架构

Update 05-23 Uniswap v3 详解系列

简单说,一个 tick 就代表 Uniswap 价格的等比数列中的某一个价格,因此每一个用户提供的流动性 的价格边界可以用 $tick_{lower}$ 和 $tick_{upper}$ 来表示。为了计算的方便,<mark>对于每一个交易对,uni 都定义</mark>

使是只使用 uniswap v3 这单个 AMM 来完成一笔交易,但是因为代币对的流通性分散在多个池子 中。那么最优的交易策略是使用交易聚合器(例如 1inch)来进行交易,即将单笔交易拆散,同时使 用多个流动性池来完成交易。就目前 uniswap v3 前端代码情况来看,官方的界面是不支持这种聚合 交易的, 其 sdk 代码中的注释也说明了这个问题: SDK 代码。 前文说过,为了减少开根号的计算,Uniswap 记录的是 \sqrt{P} , v3 使用 Q64.96 精度的定点数来存储

Frontend

本系列后续会从常用的 Uniswap v3 操作入手,讲解代码调用流程。一般来说,用户的操作都是与

12条评论 1 登录 ▼ DGCG ♡ 1 • 分享 最佳 最新 最早 tyu 💒 v3的介绍太棒了 想请教博主对在v3上的套利策略有何看法? △ 1 ♀ 0 回复 • 分享, Paco 管理员 ♣ → tyu

NEXT

UNISWAP V3 详解 (二): 创建交易对/提 供流动性



△ 0 ♀ 0 回复 • 分享, Eason 2+

(Life) (Basic) (Performance Tuning) (Operating System) (OpenStack) (Python) (Web) (Solidity

Theme by Hux | C) Star 6,486

Copyright © 坚实的幻想 2023