By @banr1 and @alphaist from Titania Research

Thanks you @cankisagun, @orest, @halo3mic and all SUAVE developers for discussing.

In this article, we delve into the implementation of aFrequent Batch Auction (FBA) on SUAVE.

The FBA mechanism involves users placing and canceling orders, with an operator executing batch fills at regular intervals.

We explore the main functions and data structures involved in this process, with a special focus on the executeFills function.

You can find the code in this GitHub repository:

GitHub

GitHub - titania-research-labs/suave-fbadex

Contribute to titania-research-labs/suave-fbadex development by creating an account on GitHub.

Overview of the FBA Process

The FBA process revolves around three main functions:

1. placeOrder

: Allows users to place orders in the order book.

1. cancelOrder

: Enables users to cancel their previously placed orders.

1. executeFills

: Executed by the operator at regular intervals to match and fulfill orders.

Key Roles and Interaction Diagram

We can illustrate the interaction between the main participants

- Users: the users who place and cancel orders
- Operator: the entity responsible for executing fills
- Contract: the smart contract that manages the order book and fills
- Storage: Confidential Data Store that stores orders and the contract storage that stores cancels

[

image

1914×1360 162 KB

](https://collective.flashbots.net/uploads/default/original/2X/d/d2b888c50826a044b257a43cbb73cc60877f095f.png)

Data Structures

The primary data structures involved in the FBA process include the Fill

, Cancel

, PlaceResult

, and CancelResult

structs.

- Order

: Represents the order with price, amount, side, and order ID.

- Fill

: Represents the filled orders with price and amount.

- Cancel

: Represents the canceled orders with order ID and side.

- FillResult

: Represents the result of a filled order with price, side, and amount.

- CancelResult

: Represents the result of a canceled order with order ID and side.

struct Order { uint256 price; uint256 amount; // 'true' for bids and 'false' for asks bool side; string orderId; }

struct Fill { uint256 price; uint256 amount; }

struct Cancel { string orderId; bool side; }

struct PlaceResult { uint256 price; bool side; uint256 amount; }

struct CancelResult { string orderId; bool side; }

The placeOrder

/ cancelOrder

Functions

The placeOrder

function allows users to submit their buy or sell orders to the order book. Depending on the order side (buy or sell), the order is inserted into the appropriate heap (bids or asks). The FBAHeap

library is used to manage the order book efficiently and stores the orders and their metadata on Credential Data Store.

function placeOrder(FBAHeap.Order memory ord) external returns (bytes memory) { (FBAHeap.ArrayMetadata memory am, FBAHeap.MapMetadata memory mm) = getMetadata(ord.side); FBAHeap.insertOrder(ord, am, mm);

```
PlaceResult memory placeResult = PlaceResult(ord.price, ord.side, ord.amount);
return abi.encodeWithSelector(this.placeOrderCallback.selector, placeResult);
```

}

The cancelOrder

function allows users to cancel their previously placed orders. The function stores the cancellation details in an array which will be processed during the executeFills

function. The cancellations are stored in the contract storage, not in the FBAHeap

.

function cancelOrder(string memory orderId, bool side) external returns (bytes memory) { cancels.push(Cancel(orderId, side));

```
CancelResult memory cancelResult = CancelResult(orderId, side);
return abi.encodeWithSelector(this.cancelOrderCallback.selector, cancelResult);
```

}

The executeFills

Function

The executeFills

function is the heart of the FBA process. It is responsible for matching and filling orders. It consists of three main parts:

1. Processing Cancel Orders

: Cancels any orders that were marked for cancellation.

1. Matching Orders

: Matches buy and sell orders based on the clearing price.

Detailed Implementation of executeFills

The executeFills

function first processes the cancel orders by deleting the canceled orders from the order book. It then matches the buy and sell orders based on the clearing price. The function iterates through the orders, updating the order book and fills array accordingly.

function executeFills() external returns (bytes memory) { (FBAHeap.ArrayMetadata memory bidAm, FBAHeap.MapMetadata memory bidMm) = getMetadata(ISBUY); (FBAHeap.ArrayMetadata memory askAm, FBAHeap.MapMetadata memory askMm) = getMetadata(ISSELL);

```
// Reset fills
fills = new Fill[](0);

////// First part: prioritize cancels
for (uint256 i = 0; i < cancels.length; i++) {
    string memory orderId = cancels[i].orderId;
    bool side = cancels[i].side;

    if (side == ISBUY) {
        FBAHeap.deleteOrder(orderId, side, bidAm, bidMm);
    } else if (side == ISSELL) {
        FBAHeap.deleteOrder(orderId, side, askAm, askMm);
    }
}
cancels = new Cancel[](0);

////// Second part: match orders
FBAHeap.Order memory bidMax = FBAHeap.getTopOrder(bidAm, ISBUY);
FBAHeap.Order memory askMin = FBAHeap.getTopOrder(askAm, ISSELL);
uint256 clearingPrice = (bidMax.price + askMin.price) / 2;
// Match orders as long as:
// 1. The highest bid is less than the lowest ask
// 2. The clearing price is less than the highest bid and greater than the lowest ask
while (bidMax.price >= askMin.price && bidMax.price >= clearingPrice && askMin.price <= clearingPrice) {
    if (bidMax.amount > askMin.amount) {
        fills.push(Fill(clearingPrice, askMin.amount));
        bidMax.amount -= askMin.amount;
        FBAHeap.updateOrder(bidMax, bidAm, bidMm);
        FBAHeap.deleteOrder(askMin.orderId, ISSELL, askAm, askMm);
    } else if (bidMax.amount < askMin.amount) {
        fills.push(Fill(clearingPrice, bidMax.amount));
        askMin.amount -= bidMax.amount;
        FBAHeap.updateOrder(askMin, askAm, askMm);
        FBAHeap.deleteOrder(bidMax.orderId, ISBUY, bidAm, bidMm);
    } else {
        fills.push(Fill(clearingPrice, bidMax.amount));
        FBAHeap.deleteOrder(bidMax.orderId, ISBUY, bidAm, bidMm);
        FBAHeap.deleteOrder(askMin.orderId, ISSELL, askAm, askMm);
    }

    // Update bidMax and askMin
    bidMax = FBAHeap.getTopOrder(bidAm, ISBUY);
    askMin = FBAHeap.getTopOrder(askAm, ISSELL);
}

return abi.encodeWithSelector(this.executeFillsCallback.selector, fills);
```

}

Conclusion

In this article, we explored the implementation of a Frequent Batch Auction (FBA) on SUAVE. We discussed the main functions involved in the FBA process, the key data structures used, and the detailed implementation of the executeFills function. For more information about the code, you can check out the GitHub repository:

GitHub

GitHub - titania-research-labs/suave-fbadex

Contribute to titania-research-labs/suave-fbadex development by creating an account on GitHub.

And this is the article on future work:

Cross chain Batch DEX [SUAVE

](/c/suave/27)

by @Alphaist & tomoki adachi from Titania research Introduction: In our previous post, we proposed buildingFrequent Batch Auction on SUAVE. The detailed implementation of the PoC for FBA on SUAVEcan also be found in this post. We demonstrated the potential for this to contribute to solving two current problems faced by DEXs: Sandwich Attacks and Loss versus Rebalancing. However, simply building an FBA on SUAVE is not significantly different from building an FBA on a Privacy L2. Of course, S…