DejaVu Sans

Fusotao Greenbook

Fusotao Dev Team

October 17, 2021

Contents

1 Introducation

2 Technology

Generally speaking, Fusotao is a set of network protocols which are consisted of either a rule of data consistency or some certain constraints of data modification including not only a transfering constraint but also a matching verification rule. The matching verification sub-protocol, a.k.a Proofs of Matching, is the main difference from other blockchains.

2.1 Matching System

Before start introducing Fusotao Protocol, let's take a look at matching system and consider why a matcher's outputs should be verified. Matching system is a trading platform enables user to pricing orders. The core component of a matching system is a data structure named orderbook which stores all orders according to the price and time, a placed order must follow the sequence to trade if price meets while the owner of an order would be ignored. Usually, to trade on a matching system, users may hande over their account ownership so the matcher can mutate the accounts. In another words, trading on a matcher relies on human trust. Any data modifications shall obey the rules below:

Rule 1 *Mutable data shall not be shared, and shared data shall be immutable.*

Rule 2 *There shall be only one associated mutator once data shared.*

Compared to the transfering transaction, the matcher as a mutator must modify the orderbook's state for each order rather than just update the states of sender and receiver. So, a matching system is a strict serial system which can be represented by the following procedure:

$$E_i + Orderbook_{i-1} \Rightarrow Orderbook_i + R_i$$

Consider an order placed. Let x be a price, y be an amount, (x_i, y_i) be the i_{th} maker by the order of price and time.

$$\begin{aligned} mb_i = & \quad -y_i \\ tb = & \quad \sum_i y_i \\ mq_i = & \quad -x_i \times y_i \\ tq = & \quad \sum_i x_i \times y_i \end{aligned}$$

where

- $matches(x, x_i)$ is $true$

Blockchain is another typical serial system whose key rule is determining the order of accepted incoming events, e.g. the Bitcoin network uses PoW algorithm and the Longest Chain rule to ensure the unique sequence of transactions. It seems nature to build a matching system on blockchain just like a plain transfer function. But unfortunately, due to the limitation of block capacity and high latency, it is not straightfoward to do it. An order must occupy 73 bytes at least:

$$ID + Owner + Price + Unfilled + Direction = 73 bytes$$

where:

- $Price$ and $Unfilled$ are 128-bits fixed number
- ID indicates the order
- $Owner$ is the pubkey of user
- $Direction$ is 1-bit direction

Imagine there are tens of thousands of orders in a single orderbook, it is not acceptable to store all orders in the blockchain state machine. But only keeping some essential data to validate the matching results is possible. In the next section, we will introduce how to validate the matching results without holding the whole orderbook.

2.2 Global States

Sparse Merkle Tree is a full binary hash tree with fixed-depth which can be used for checking whether a node belongs to the tree. A node of Sparse Merkle Tree can be represented by the following function:

$$\psi_x = \eta(\psi_{xL}, \psi_{xR}),\ len(x) < height$$
$$\psi_x = v,\ len(x) = height$$

where

- x is the key of a node calculated by $\eta(value)$ » $height$, e.g. $0x00...a82e$

The capacity of a Sparse Merkle Tree depends on the hash function of the tree, e.g. a Sparse Merkle Tree using Sha256 has 2^{256} leaf nodes with height=256(root excluded). Given a data v, we can simply verify whether it belongs to a certain tree by calculating $height - 1$ times hash function:

```
let h = hash(v)
from 0 to 255:
    do h = hash(h, sibling_of_h)
return h == root
```

Take a look at the matching procedure, we can't hold the whole orderbook on chain, but we can encode the orderbook into a Sparse Merkle Tree and save the root hash of the tree. Once a matcher executes the i_{th} event off chain and submits the makers at $i-1_{th}$, we can simply validate like above.

2.3 Verification