# Architecture

haipo yang edited this page on Nov 11, 2017 · 2 revisions

## Design Purpose

1. Capacity of over 10 thousand transactions per second
2. Support trading pairs between multiple currencies
3. Distributed cluster architecture with high availability

## Technical selection

Typical Bitcoin exchanges use relational database for their trade matching machine. The advantage is that it can rapidly realize business logic and guarantee data accuracy and reliability using data-dependent index, transaction etc. mechanisms. But it would also invite problems to the database and result in poor performance. With the development of quantitative trading, an increasing number of orders are now processed systematically and most of them are high-frequency trading with large order quantity which requires high standard for transaction interface delay. When faced with these technical issues, mainstream exchanges are now aware that these traditional data-dependent database can no longer meet the growing demand of trading. In order to break through the bottlenecks of database performance, we have chosen single process to avoid spending on database transactions and locks, and memory calculation to avoid spending on data persistence in return of significant performance improvement.

Fundamentally, the mechanism of a matching machine is simple: Submit orders by time, and match trading based on preferences of price and time efficiency. User balance change indicates the trading result. Since user deposit and withdrawal will also affect account balance, therefore, the final result should be identical to operational log. This is very similar to AOF mode of Redis, which essentially is an in-memory database that relies on operational logs for data recovery. Besides, by generating data state slices periodically, the matching engine can upload slice data and then log data to complete data recovery, hence reducing time of loading historical data to restart services.

Based on calculation of Benchmark of Redis, a single write server is fully capable of supporting up to 10,000+ orders. For the common benefit of achieving high availability, matching service requires one-master multi-slave cluster and in our case, we use Zookeeper to ensure its management. In addition, database is required for asynchronous persistent storage of order history, transaction history, asset change history etc.

Clone this wiki locally

`https://github.com/viabtc/vi`

## Overall structure

- Basic system: Zookeeper, kafka, Redis Sentinel, MySQL cluster
- Matching: One-master multi-slave, Read/Write Splitting
- Market: Generate K-line data
- Quantizer: Provide basic strategy trading for users
- HTTP: Provide HTTP socket for built-in system
- WEBSOCKET: Provide WEBSOCKET for users

## Goal of phrase 1:

Implement single-instance matching services regardless of high availability