NG (Next Generation) v4 dYdX Orderbook

Author: Lawrence Chiu (lawrence-ext@dydx.foundation)

Twitter: https://twitter.com/LawrenceChiu14

Date: 9/7/2024

Tested on: Ubuntu Server 22.04 LTS, 16 vCPU, 64 GiB Memory

Design Considerations

1. The previous iteration of my v4 dYdX Orderbook used filesystem storage. This ran into scaling issues as the number of v4 markets grew. Processes became unstable, and would sometimes hang.
2. In addition, the prior iteration used a single program to read from indexer websocket and also process the data. As the number of v4 markets grew, this lead to frequent websocket disconnections.
3. The solution was to use PostgreSQL for storing data. To arrive at this solution, I also evaluated Redis and MySQL but ultimately chose PostgreSQL for the following reasons:
    a. Redis was fast at storing data but querying was slow. I stored price/size using hset() and collected them in a set using sadd() but reading this data using smembers() took a lot longer than I can tolerate.
    b. MySQL operated well and I could have gone with this except it used a lot more storage than PostgreSQL.
    c. PostgreSQL had the best of both worlds, it is both fast at storing and retrieving data, and also used the smallest amount of storage.
4. The second change was to implement Python multiprocessing.Pool() to build a client-server architecture where two programs work in conjunction. One program to read from indexer websocket and pass this data to the second program asynchronously (in order to achieve maximum performance), and a second program that processed this data (storing into the database)

Part 1) Setting up

1. Install PostgreSQL:

```
sudo apt-get install postgresql
```

2. Install the Python library psycopg

```
pip3 install psycopg
```

3. Create the database and required tables:

```
sudo su - postgres
psql
create database orderbook;
create user vmware with encrypted password 'orderbook';
grant all privileges on database orderbook to vmware;
exit
```

switch back to your regular user that will run the orderbook.

```
psql -h localhost -d orderbook -U vmware
create table v4orderbookindex (market1 varchar(255) not null, index1
int not null, primary key (market1));
```

Part 2) Checklist

1. There are 4 programs:
    a. v4dydxob.py (the server program that reads from indexer websocket)
    b. v4dydxobclient.py (the client program that processes the data from v4dydxob.py)
    c. v4dydxob2.py (the display program to show the orderbook)
    d. v4cleandb.py (this clears the old orderbook data, and should be executed before starting the server)

Part 3) v4dydxob.py and v4dydxobclient.py

1. Remember to run cleandb.py first if you have old orderbook data in the database. Skip this step if this is your first time.

```
nohup python3 -u v4cleandb.py
```

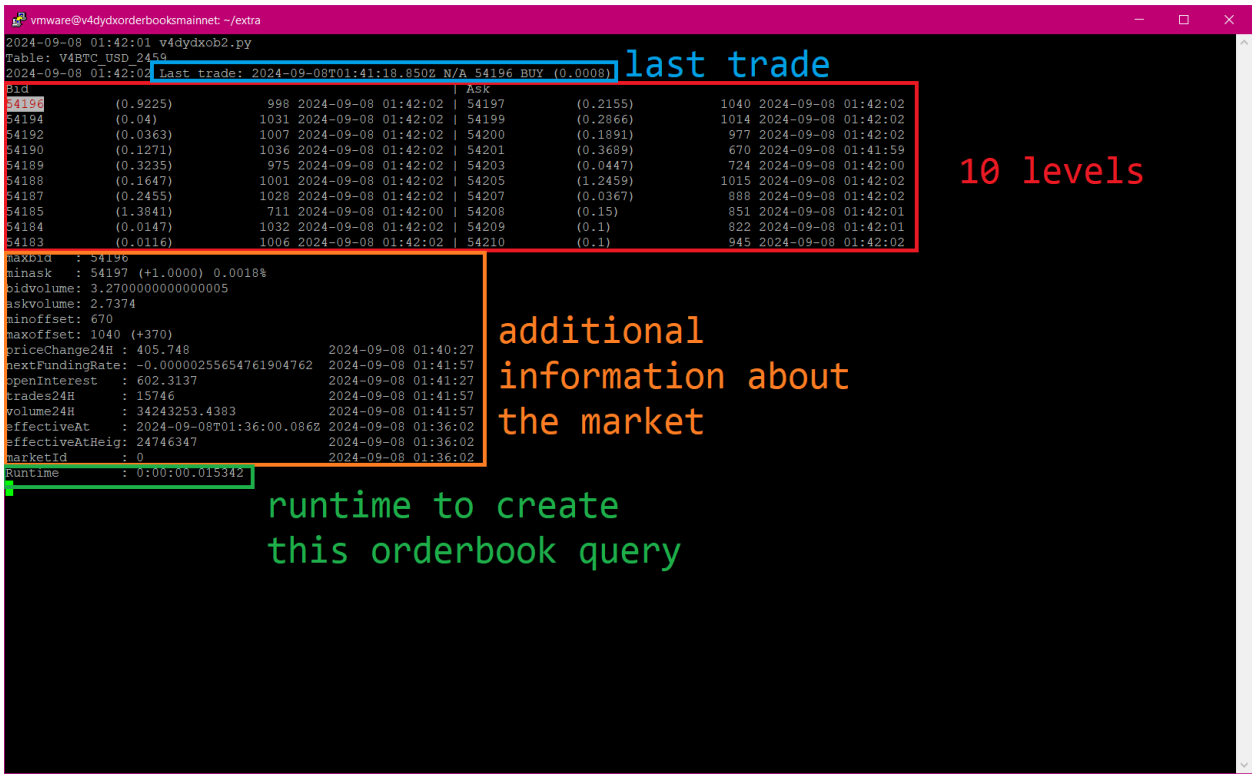
2. Run v4dydxob.py, the server program. The client program v4dydxobclient.py is handled entirely by the server. It takes only 1 argument which is the market (e.g. BTC-USD, ETH-USD, etc.)

```
nohup python3 -u v4dydxob.py BTC-USD > /tmp/v4dydxobBTC-USD.log 2>&1 &
```

Part 4) v4dydxob2.py

1. This program displays the orderbook to your screen. It takes up to 3 arguments: 1) the market, b) the number of levels to display, and 3) whether to color the output (specify noansi to disable)
2. For example, to display 10 levels and color the output:

```
python3 -u v4dydxob2.py BTC-USD 10
```



Notice that the Bid price of 54196 is in red color to show it was also the last traded price.

3. To display without color, and the output would be identical to the above except the Bid price would not be in color. This is useful if you intend to use the data in another program.

```
python3 -u v4dydxob2.py BTC-USD 10 noansi
```