

NG (Next Generation) v4 dYdX Orderbook

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

Twitter: <https://twitter.com/LawrenceChiu14>

First release Date: 9/7/2024

Last Updated Date: 9/16/2024

Tested on: Ubuntu Server 22.04 LTS, 16 vCPU, 64 GiB Memory. Recommended machine-type on Google Cloud: t2d-standard-16

Change Log

1. 9/7/2024 First release
2. 9/9/2024: Documented remote mode for v4dydxob2.py
3. 9/16/2024: Updates, including new Part 5

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 led 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() (a set is the logical orderbook), but reading this structure using smembers() took a lot longer than I can tolerate. I require querying and displaying the orderbook to take no more than 20 milliseconds.
 - b. MySQL operated well and I could have gone with this except it used a lot more disk storage than PostgreSQL.

- c. PostgreSQL had the best of both worlds, it was both fast at storing and retrieving data, and also used the smallest amount of disk storage. It can query and display the order book in 10 milliseconds.
4. The second change was to implement Python multiprocessing.Pool() to build a client-server architecture where two separate 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. You'll also need python3-pip.

```
sudo apt-get install postgresql
Sudo apt-get install python3-pip
```

2. Install the Python library psycpg. You'll also need websocket-client, websockets, and tornado.

```
pip3 install "psycpg[binary]"
pip3 install websocket-client
pip3 install websockets
pip3 install tornado
```

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));
create table v4server (market1 varchar(255) not null, messageid int
not null, primary key (market1));
create table v4client (market1 varchar(255) not null, messageid int
not null, primary key (market1));
```

4. Configure PostgreSQL to allow network connections:

a) Add the following line to `/etc/postgresql/14/main/postgresql.conf`:
`listen_addresses = '**'`

b) Change the following line:
From: `max_connections = 100`
To: `max_connections = 1000`

c) Next, open file `pg_hba.conf` and change the following line:
From: `host all all 127.0.0.1/32 scram-sha-256`
To: `host all all 0.0.0.0/0 scram-sha-256`

Part 2) Checklist

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

You run a) `v4dydxob.sh` which runs the two other programs b) `v4dydxob.py` and c) `v4dydxobclient.py` for you. You DO NOT run the other two programs on your own.

Part 3) `v4dydxob.sh`

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 cleandb.py
```

2. Run `v4dydxob.sh`. It takes only 1 argument which is the market (e.g. BTC-USD, ETH-USD, etc.)

```
nohup ./v4dydxob.sh BTC-USD > /tmp/v4dydxobBTC-USD.log 2>&1 &
```

Part 4) v4dydxob2.py

1. (Optional) This program can run from the same server as v4dydxob.py OR a remote server. To run this on a remote server, set the environment variable with the IP address of the order book server. If this is not set, the program assumes the same server.

```
export ORDERBOOKSERVER=192.168.0.169
```

2. 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)
3. For example, to display 10 levels and color the output:

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

```
vmware@v4dydxorderbooksmainnet: ~/extra
2024-09-08 01:42:01 v4dydxob2.py
Table: V4BTC_USD_2459
2024-09-08 01:42:02 Last trade: 2024-09-08T01:41:18.850Z N/A 54196 BUY (0.0008)
bid                                     ask
54196 (0.9225) 998 2024-09-08 01:42:02 | 54197 (0.2155) 1040 2024-09-08 01:42:02
54194 (0.04) 1031 2024-09-08 01:42:02 | 54199 (0.2866) 1014 2024-09-08 01:42:02
54192 (0.0363) 1007 2024-09-08 01:42:02 | 54200 (0.1891) 977 2024-09-08 01:42:02
54190 (0.1271) 1036 2024-09-08 01:42:02 | 54201 (0.3689) 670 2024-09-08 01:41:59
54189 (0.3235) 975 2024-09-08 01:42:02 | 54203 (0.0447) 724 2024-09-08 01:42:00
54188 (0.1647) 1001 2024-09-08 01:42:02 | 54205 (1.2459) 1015 2024-09-08 01:42:02
54187 (0.2455) 1028 2024-09-08 01:42:02 | 54207 (0.0367) 888 2024-09-08 01:42:02
54185 (1.3841) 711 2024-09-08 01:42:00 | 54208 (0.15) 851 2024-09-08 01:42:01
54184 (0.0147) 1032 2024-09-08 01:42:02 | 54209 (0.1) 822 2024-09-08 01:42:01
54183 (0.0116) 1006 2024-09-08 01:42:02 | 54210 (0.1) 945 2024-09-08 01:42:02

maxbid : 54196
minask : 54197 (+1.0000) 0.0018%
bidvolume: 3.2700000000000005
askvolume: 2.7374
minoffset: 670
maxoffset: 1040 (+370)
pricechange24H : 405.748 2024-09-08 01:40:27
nextFundingRate: -0.00000255654761904762 2024-09-08 01:41:57
openInterest : 602.3137 2024-09-08 01:41:27
trades24H : 15746 2024-09-08 01:41:57
volume24H : 34243253.4383 2024-09-08 01:41:57
effectiveAt : 2024-09-08T01:36:00.086Z 2024-09-08 01:36:02
effectiveAtHeig: 24746347 2024-09-08 01:36:02
marketId : 0 2024-09-08 01:36:02
Runtime : 0:00:00.015342
```

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

4. 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
```

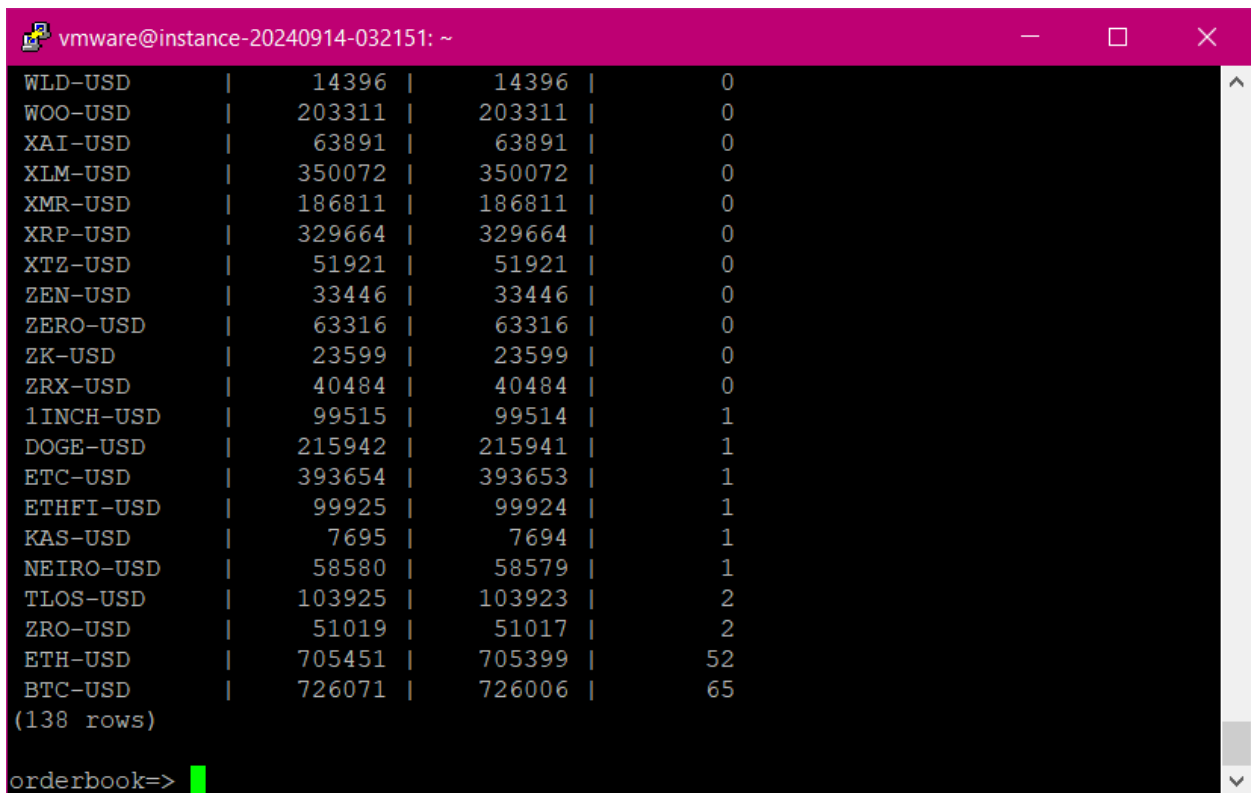
Part 5) Useful DBA Queries

1. Log into the database with the following command:

```
psql -h localhost -d orderbook -U vmware
\pset pager off
```

2. Run the following query to see the status of the orderbook:

```
select a.market1, a.messageid, b.messageid, (a.messageid - b.messageid) from v4server a,
v4client b where a.market1=b.market1 order by (a.messageid - b.messageid), market1;
```



WLD-USD	14396	14396	0
WOO-USD	203311	203311	0
XAI-USD	63891	63891	0
XLM-USD	350072	350072	0
XMR-USD	186811	186811	0
XRP-USD	329664	329664	0
XTZ-USD	51921	51921	0
ZEN-USD	33446	33446	0
ZERO-USD	63316	63316	0
ZK-USD	23599	23599	0
ZRX-USD	40484	40484	0
1INCH-USD	99515	99514	1
DOGE-USD	215942	215941	1
ETC-USD	393654	393653	1
ETHFI-USD	99925	99924	1
KAS-USD	7695	7694	1
NEIRO-USD	58580	58579	1
TLOS-USD	103925	103923	2
ZRO-USD	51019	51017	2
ETH-USD	705451	705399	52
BTC-USD	726071	726006	65
(138 rows)			

orderbook=>

There are 4 columns, consisting of (from left to right): 1) Market, 2) Server's message_id, 3) Client's message_id, and 4) Lag (delta between #2 and #3). If Lag is very large number (>1,000), it means you need a faster server. The lag is the number of messages the client (writing to the PostgreSQL database) is lagging the data fed by the server (reading from indexer websocket).