Wood Coding Challenge - Stock Exchange

This is a code for the Wood coding challenge.

Task

Implement stock exchange server that can trade a stock between multiple users. Server have two parts: private and public. Private part expects orders from clients that are later matched and traded. Public part informs connected clients about situation at the stock exchange.

You can have two types of orders: Bid (buy) and Ask (sell). You can trade them if bid price is higher or equal than ask price. When some orders are traded all participants will get message about that and it is also reflected on public channel.

For more information please look at Wood coding chalenge website (in czech).

Installation

You will need python 3.5 to run this. To be able to use this repository you need to:

1. Clone it.
2. Install requirements `$ pip install -r requirements.txt`

3. To use multiple server environment install also Redis and specify its address in `wood.wood.settings_local`.
4. If you want to monitor application install Graylog (I tested it with docker image).

Usage

To start server call:

```
$ python -m wood -s
```

To start two artificial clients that will make some traffic call:

```
$ python -m wood -c -n 2
```

To discover all settings call:

```
 python -m wood -h
usage: __main__.py [-h] [-s] [-m] [--private PRIVATE] [--pu
blic PUBLIC]
                   [--persist] [-c] [-n NUMBER_OF_CLIENTS]
[-p PORT]

optional arguments:
  -h, --help            show this help message and exit

server:
  -s, --server          Run server.
  -m, --multiple-servers
                        Run multiple server environment (ne
eds redis).
  --private PRIVATE     Private port.
  --public PUBLIC       Public port.
  --persist             Whether to persist limit order book
. Only for multiple
                        server environment.

client:
  -c, --client          Run random client.
  -n NUMBER_OF_CLIENTS, --number-of-clients NUMBER_OF_CLIEN
TS
                        Number of connected clients.
  -p PORT, --port PORT  Private port of the server.
```

To use in multiple environment (see chapter Implementation.Mutliple servers) you can use prepared haproxy configuration `haproxy.cfg` that will load balance between three servers. You need to run three servers (or more if you edit haproxy configuration) on ports 17001+17002, 17003+17004, 17005+17006:

```
$ python -m wood -s -m --private 17001 --public 17002
```

To check that everything is all right run:

```
$ py.test
```

To run tests also in multiple server environment run (this command will erase redis data):

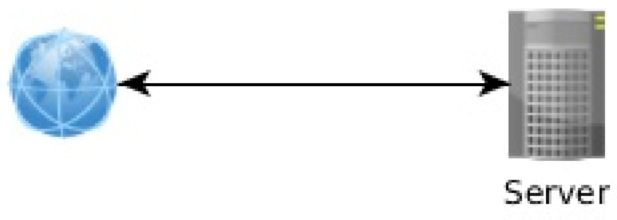
```
$ py.test --redis
```

Implementation

Server was written in Python 3.5 with massive using of `asyncio`.

Single server

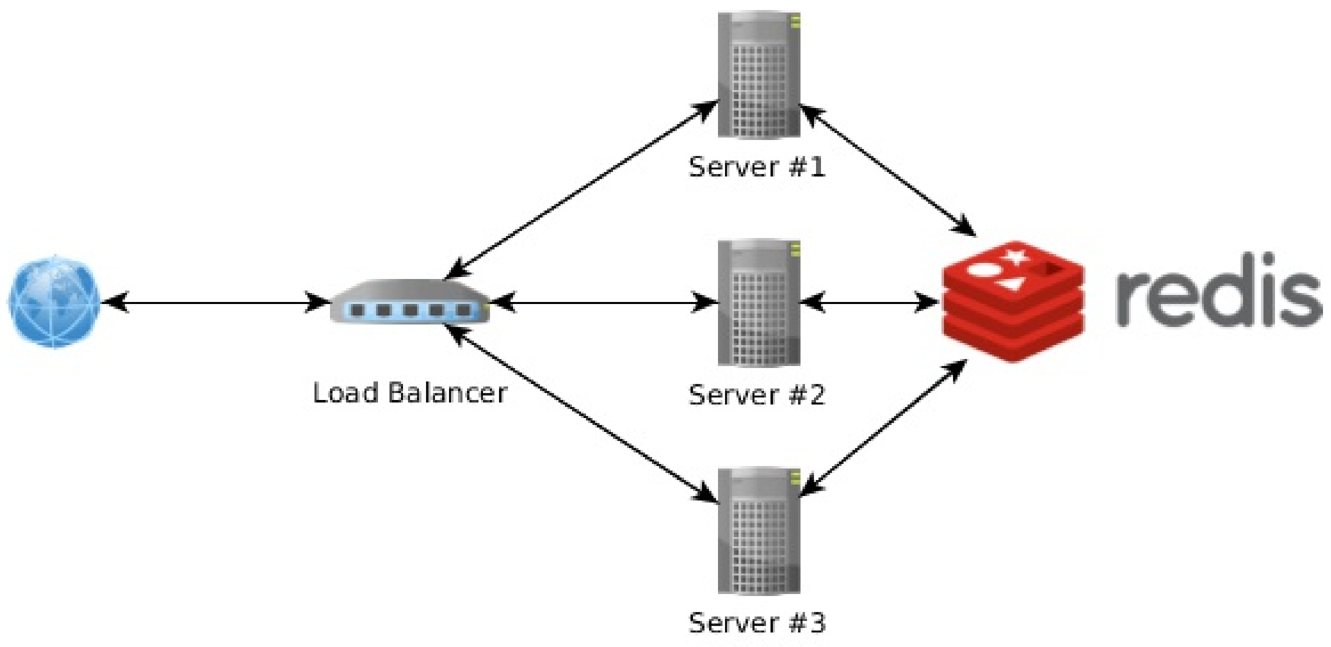
Single server architecture is easier variant of stock server that is not scalable to more processes and machines. It is single threaded application with most simple network diagram:



It uses in-memory priority queue and publisher/subscriber messaging.

Multiple servers

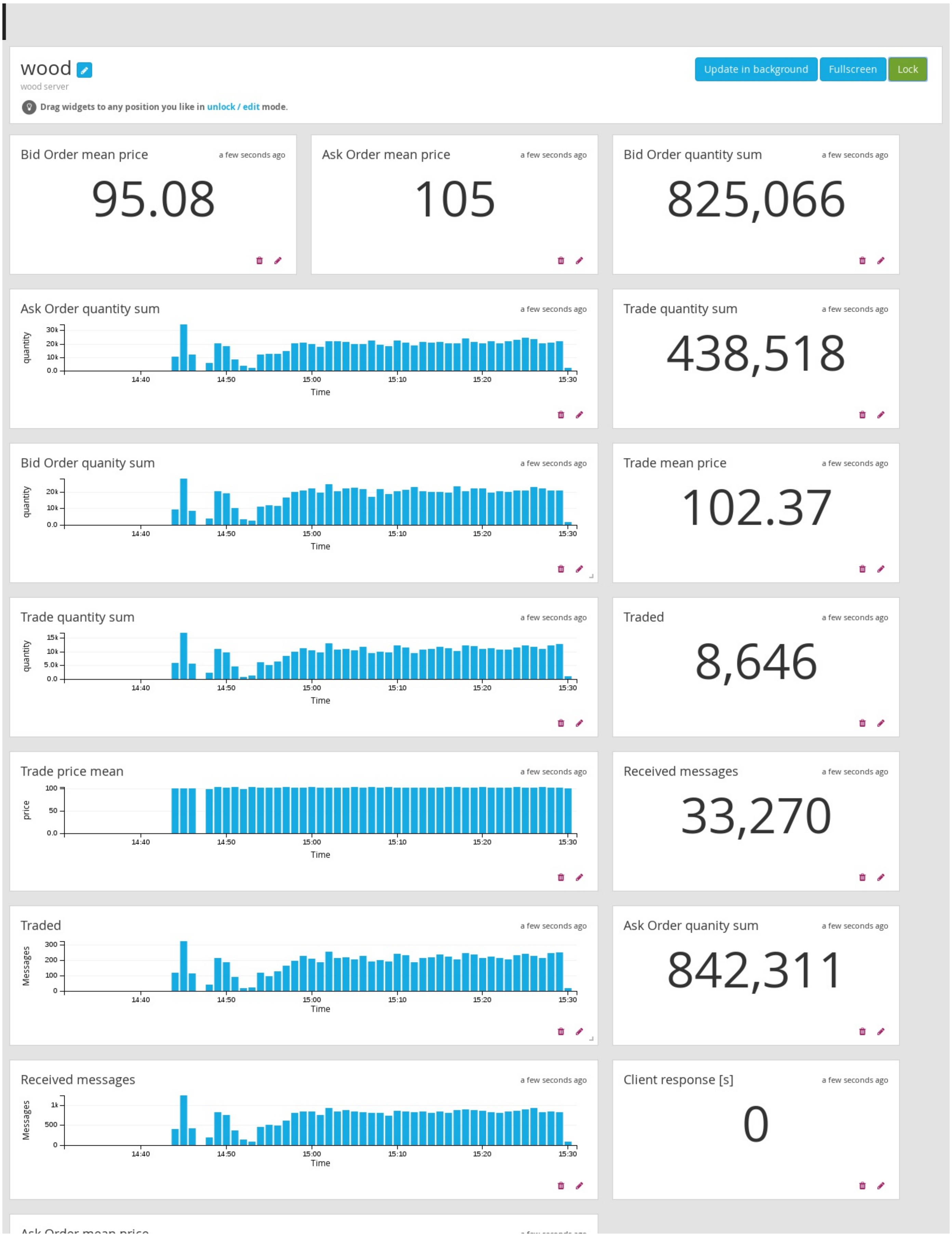
Multiple servers architecture aims to be scalable extension of single server architecture. It is possible to run it in different processes on the same machine or run it on more machines in cluster. To load balance between all the servers you can use HAProxy. Sample configuration is prepared in `haproxy.cfg`. Here you can see network diagram of multiple servers architecture:



To achieve this I needed to store Limit order book in a database. Also I needed to create a communication channel between multiple servers because when trade is performed on one server, another needs to know about it to inform client. I chose Redis because it is fast, in memory and allows to create priority queue and publish subscribe messaging. Multiple servers allows easily horizontally scale to more machines without much effort.

Monitoring

Monitoring is essential part of every long running application. To monitor wood server I chose Graylog which is an excellent way to handle logs. Logs are stored in ElasticSearch and therefore you can search in them very fast. Another greate feature of Graylog is its dashboards where you have complete overview of the state of your application. Here you can see a dashboard I made for wood server with all information about trades you need to know:


wood

wood server

Drag widgets to any position you like in unlock / edit mode.

Update in background | Fullscreen | Lock

Bid Order mean price — a few seconds ago — 95.08

Ask Order mean price — a few seconds ago — 105

Bid Order quantity sum — a few seconds ago — 825,066

Ask Order quantity sum — a few seconds ago

Trade quantity sum — a few seconds ago — 438,518

Bid Order quanity sum — a few seconds ago

Trade mean price — a few seconds ago — 102.37

Trade quantity sum — a few seconds ago

Traded — a few seconds ago — 8,646

Trade price mean — a few seconds ago

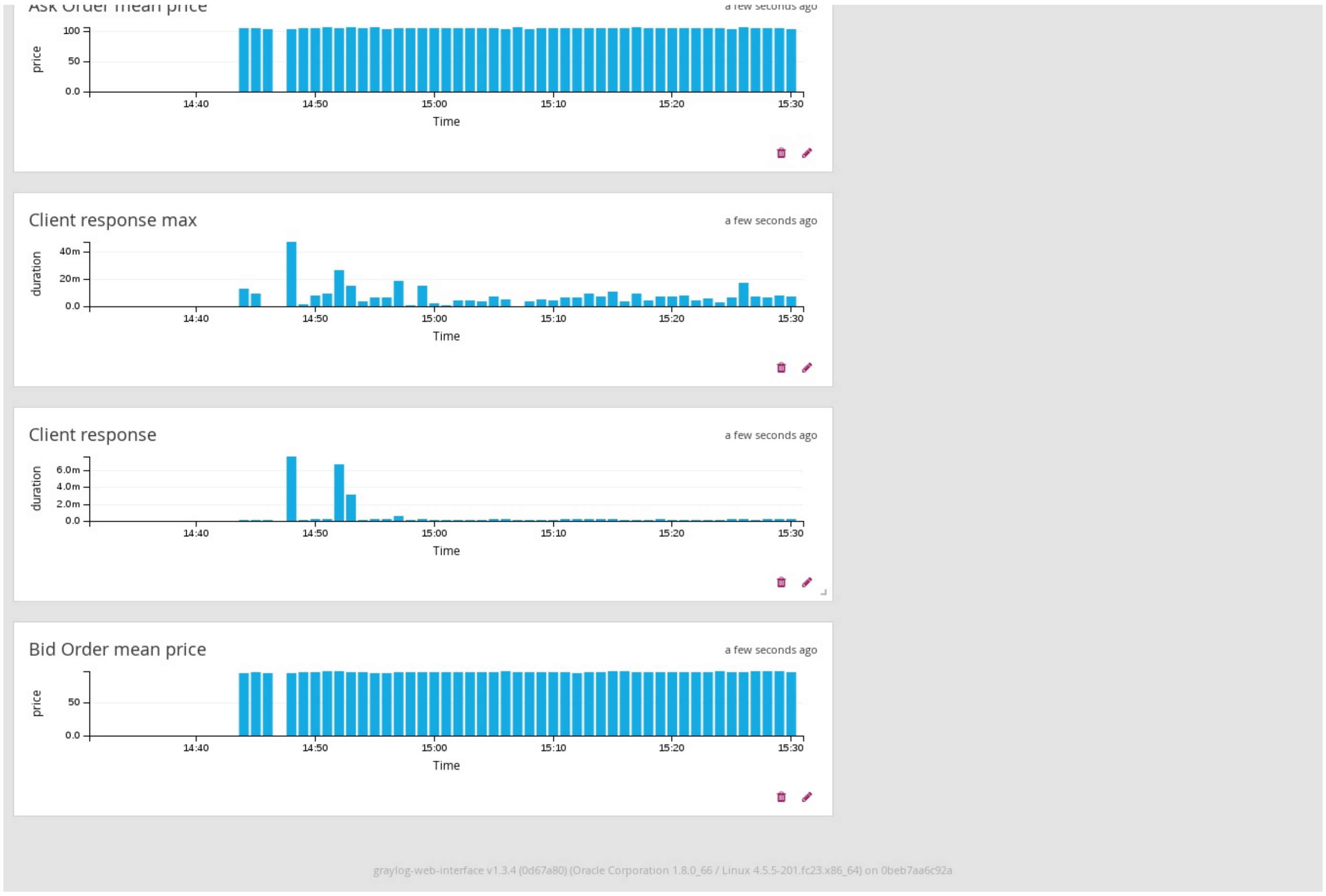
Received messages — a few seconds ago — 33,270

Traded — a few seconds ago

Ask Order quanity sum — a few seconds ago — 842,311

Received messages — a few seconds ago

Client response [s] — a few seconds ago — 0



Key features

- Used modern python library `asyncio`
- Ability to horizontally scale using `redis`.
- Monitoring application with `graylog` dashboard.
- Persistency of limit order book in multiple server environment.