



Simulátor akciové burzy

Dokumentace systému

Obsah

Shrnutí implementace.....	2
Kód.....	2
Struktura programu.....	3
Spuštění.....	3
Efektivita.....	5
Automatické testy.....	5
Zátěžový test.....	6
Funkčnost rozšiřující zadání.....	6
Neceločíselné ceny.....	6
Ošetření chyb.....	6
„Future work“.....	6

Shrnutí implementace

Systém je implementován v jazyce Python pro verzi 3.5. Pro síťovou komunikaci využívá knihovnu *asyncio*, pro datovou strukturu haldy knihovnu *heapq*; obě jsou součástí Pythonu. Umí využít knihovnu *ujson* ke zvýšení výkonu, pokud je dostupná.

Cílem implementace je maximální efektivita, zejména při velkém počtu otevřených pokynů, ale bez změny protokolu. Toho je dosaženo hlavně datovou strukturou *Limit Order Booku*. Kromě požadované funkcionality a testů jsou navíc implementovány mj. přesné neceločíselné ceny. Za účelem vyhodnocování efektivity obsahuje soubor testů i zátěžový test. Program můžete nainstalovat pomocí přiloženého *setup.py* skriptu.

Kód

Kód serveru je formátován podle standardu *PEP8* (s délkou řádků 121 znaků) a obsahuje dokumentaci všech tříd a veřejných metod.

Příklad kódu – metoda *Exchange.cancel_order*:

```
async def cancel_order(self, clientid: int, orderid: str):
    """
    Removes order
    :param clientid: int id of the client
    :param orderid: order id unique for the client
    :return: None
    """
    order = self.book.remove_order(clientid, orderid)
    if self.datastream_callback:
        await self.datastream_callback("cancel", order.side, order.time,
                                       order.price, order.qty)
```

Struktura programu

Kód simulátoru je rozdělený do balíčků *tests* a *exchange* a v *exchange* do modulů *main*, *server*, *exchange*, *book* a *order*.

Modul *server* a jeho třídy abstrahují komunikační protokol od logiky burzy, která je díky tomu na protokolu nezávislá. Privátní a veřejný kanál je rozdělen do tříd *OrderServer* a *DatastreamServer*, jež dědí společnou funkcionalitu od předka, abstraktní třídy *GenericServer*.

Modul *exchange* zapouzdřuje celou logiku burzy: práci s pokyny a poskytování informací o jejich vykonávání. Volající může třídě *Exchange* pomocí metody *set_callbacks(f1, f2)* předat dvě funkce, kterým budou předávány informace pro privátní a veřejný kanál.

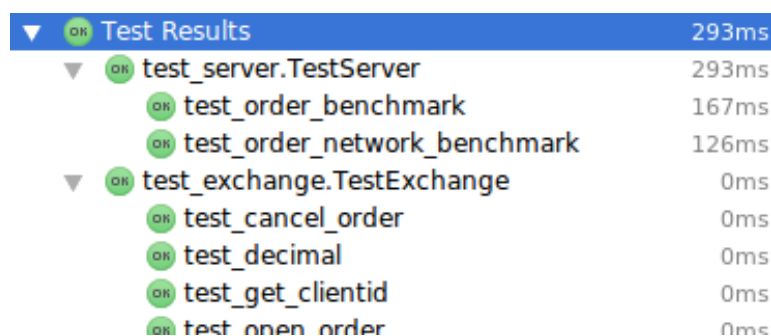
Book je datová struktura reprezentující *Limit order book*. Zajišťuje práci s pokyny a jejich matchování a umí efektivně vyřizovat dotazy na počet pokynů s danou cenou, což je třeba pro veřejný kanál.

Order je datový typ reprezentující burzovní pokyn. Definuje pořadí pokynů primárně podle ceny a sekundárně podle času pro oba typy pokynů, „BUY“ a „SELL“. Tato uspořádání pak používají haldy v *Booku*.

Spuštění

Server je možné spustit souborem *exchange-simulator.py*. Standardně bude naslouchat na portech 7001 pro privátní a 7002 pro veřejný kanál. Změnit porty je možné přes argumenty příkazové řádky, viz argument „--help“. Testy je možné spustit například příkazem „python -m unittest discover“. Celý balík *exchange* můžete nainstalovat příkazem „python setup.py install“.

Program kromě interpretru Python 3.5 nevyžaduje žádné další knihovny, umí ale v případě dostupnosti využít knihovnu *ujson* ke zvýšení výkonu.



Test Results	293ms
test_server.TestServer	293ms
test_order_benchmark	167ms
test_order_network_benchmark	126ms
test_exchange.TestExchange	0ms
test_cancel_order	0ms
test_decimal	0ms
test_get_clientid	0ms
test_open_order	0ms

Efektivita

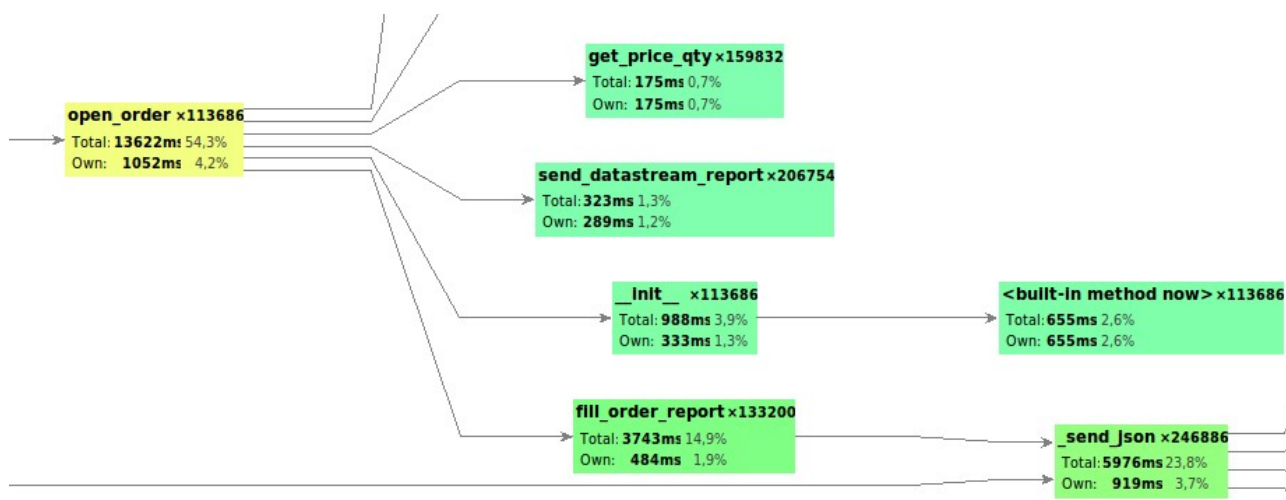
Simulátor burzy byl vyvinut s důrazem na efektivitu, a to zejména při vysokém počtu vyřizovaných pokynů. *Book* proto obsahuje dvě haldy pokynů a jednu (hashovací) tabulku počtu pokynů podle ceny.

Průměrný asymptotický čas operací je pro n pokynů v *Booku*:

- vytvoření pokynu: $O(1)$
- match pokynů: $O(\log_2(n))$
- dotaz na množství komodity v tabulce booku podle ceny: $O(1)$
- zrušení pokynu: $O(n)$

Podle výsledků zátěžových testů je systém schopný obsloužit řádově 10 000 pokynů za sekundu a to i pokud *Book* obsahuje desítky tisíc pokynů. Propustnost serveru se s otevíranými pokyny snižuje jen pomalu.

Na základě výsledků profilování se za účelem zrychlení používá místo modulu *json* knihovna *ujson*, pokud je na systému k dispozici.



Automatické testy

Kód obsahuje automatické testy pomocí modulu *unittest*. Testování pokrývá veřejné metody třídy *Exchange* a síťovou logiku tříd modulu *server*. Nepokrývají ale veškerou funkcionalitu a obsluhu chyb. Zbývající třídy *Book* a *Order* jsou testovány nepřímo. Moduly testů jsou umístěny v balíčku *tests* a začínají prefixem „test_“.

Zátěžový test

Testy obsahují i dva zátěžové testy. První je určen pro měření výkonu síťové části a vytváří pokyny tak, aby v *booku* zůstal vždy maximálně jeden. Druhý test generuje ceny a množství pokynů z normálního (Gaussovského) rozdělení, takže se v *booku* postupně hromadí nezobchodované pokyny. Tento test tedy hodnotí výkonnost logiky burzy a datových struktur.

Funkčnost rozšiřující zadání

Funkci *orderid*, uvedeného v zadání, jsem interpretoval jako číslo unikátní pro daného klienta. Jelikož si *orderid* volí klienti, bylo by jinak možné, aby si příkazy rušili navzájem.

Neceločíselné ceny

Jelikož je standardní desetinná aritmetika nepřesná, používá simulace modul *decimal*, který nabízí datový typ s efektivními a zároveň přesnými číselnými operacemi, garantuje přesnost až do pevně daného počtu desetinných míst. V json protokolu systém manipuluje s cenami jako s řetězcem, což narušuje kompatibilitu s klienty ve starter-kitu.

Ošetření chyb

Pokud nastane při komunikaci s klientem k výjimce, server vypíše chybové hlášení včetně tracebacku na standardní chybový výstup. Klientovi pošle report s chybovou hláškou:

```
{
  "message": "error",
  "report": "Processing your request failed"
  "reason": "KeyError: 'Order with specified id does not exit'",
}
```

„Future work“

Systém by bylo vhodné ještě, mimo nápadů v zadání, v několika směrech rozšířit. Komunikace by měla probíhat uvnitř šifrovaného spojení. Pro nasazení je vhodné pro server vytvořit „initscript“ v závislosti na distribuci a instalační balíček nebo třeba *Docker* image. Testy nepokrývají veškerou funkcionalitu a bylo by vhodné je rozšířit.