数字货币交易系统架构

Python实现

黄毅

目录 CONTENTS

1 交易系统功能介绍

交易系统功能-基本概念

```
class Side(Enum):
  ‘下单方向’
  BUY = 0
  SELL = 1

class Order(NamedTuple):
  ‘订单’
  id: int
  side: Side
  amount: Decimal
  price: Decimal
  user: int
```

```
class Trade(NamedTuple):
  taker: Order  # 流动性提取方
  maker: Order  # 流动性提供方

  def id(self):
    return (self.taker.id,
            self.maker.id)

  def price(self):
    return self.maker.price

  def amount(self):
    return min(self.taker.amount,
               self.maker.amount)

  def side(self):
    return self.taker.side
```

交易系统功能-伪代码

```
@app.post('/limit_order')
def limit_order(order: Order):
    # 冻结资金
    lock_user_fund(order)
    # 撮合
    trades = book.match(order)
    # 更新资金余额
    update_user_fund(trades)
    # 记录成交历史
    save_trade(trades)
    # 更新订单状态
    save_order(order, trades)
    # 更新K线
    update_kline(trades)
    # 推送用户信息变更
    push_user_messages(trades)
```

交易系统功能-撮合

```
def match_buy(book, taker: Order):
  for price, orders in sorted(book.asks):
    for maker in sorted(orders, key=time):
      if maker.price <= taker.price:
        # 产生一笔成交
        amount = min(taker.amount, maker.amount)
        price = maker.price

        taker.amount -= amount
        maker.amount -= amount

        if maker.amount == 0:
          # maker完全成交，移除maker订单

        if taker.amount == 0:
          # taker完全成交, 结束
```

交易系统功能-K线

```
select time_bucket(time, '1 minute') as time
        first(price order by time) as open
        last(price order by time) as close
        max(price) as high
        min(price) as low
        sum(amount) as amount
from trades
group by 1
```

2 交易系统架构设计

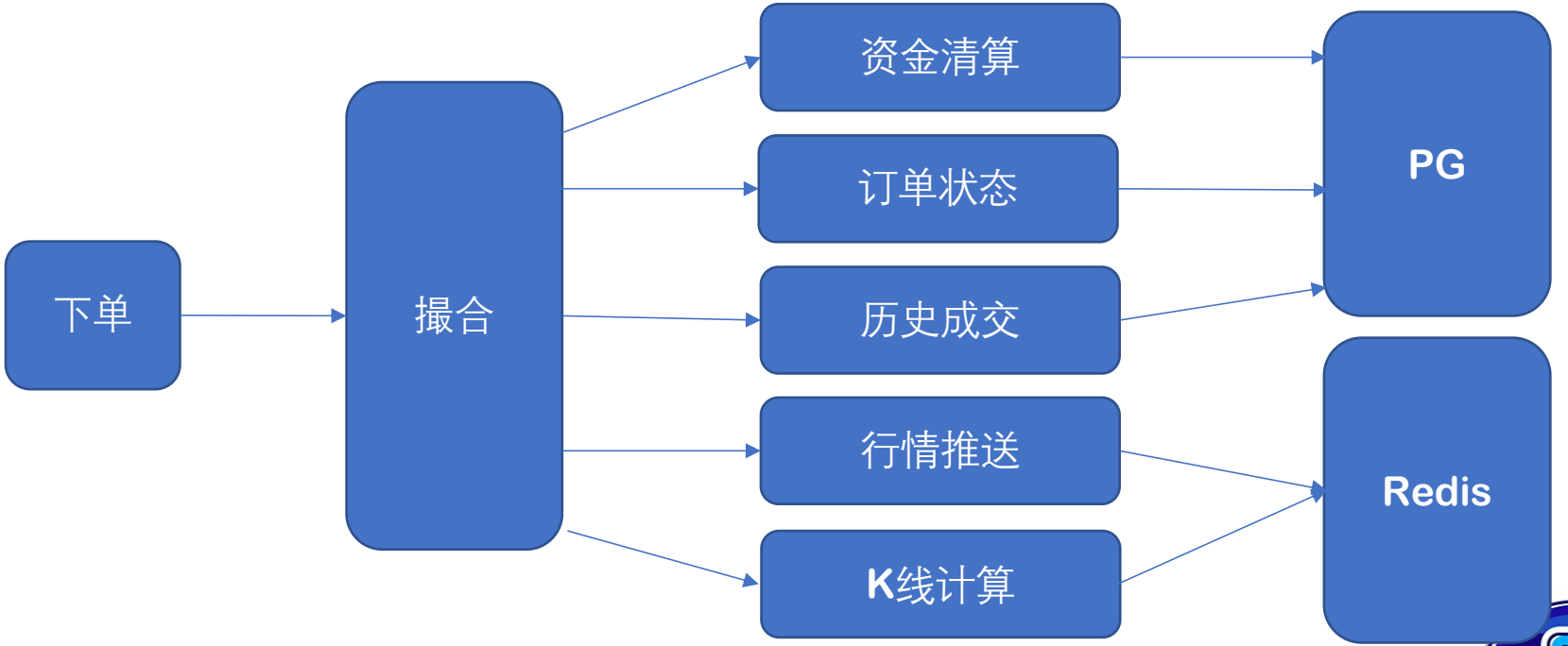
交易系统架构设计-设计目标

- 吞吐量**5w TPS,** 延迟**10ms**
- 能睡个好觉**(**数据持久化，数据最终一致性，容错**)**
- 设计简单，运维方便

交易系统架构设计-关键点

- 内存撮合服务的状态持久化和高可用
- 消息处理顺序的保证
- 异步服务之间消息传送的可靠性
- **K**线：实时的分组聚合操作
- 高效利用关系数据库

交易系统架构设计-架构图



交易系统架构设计-撮合服务

- **C++**实现的**Redis Module**
- 复用**Redis**本身的**aof**持久化和**replication**
- 撮合结果直接输出到本地的**redis stream**（无需主动发送消息给其他服务）
- 下游任务订阅**redis stream**

```
$ redis-cli
> ob.limit b{btc_usdt} * 10000 buy 1 10000
88181047656742912
```

- 实现为**Redis Module**的流式聚合组件(**https://github.com/cryptorelay/redis-aggregation**)

```
$ redis-cli
> agg.new agg{btc_usdt} time price amount
OK
> agg.view agg{btc_usdt} kline_1m{btc_usdt} interval 60
  first price max price min price last price sum amount
OK
> agg.insert agg{btc_usdt} 1571408598000 9000 1
> agg.save agg{btc_usdt}
> hgetall kline_1m{btc_usdt}
1571408580000
“[9000,9000,9000,9000,1]”
```

交易系统架构设计-redis stream

- **Kafka-like**
- 一个订阅者的情况，消息有序
- **Consumer group**负载均衡（不保证消息顺序）
- 可以批量处理消息
- 在同一个事务中存储处理结果和更新消费**offset**

```
$ redis-cli
> xread block 60 count 100000 streams b{btc_usdt}o offset
…
```

交易系统架构设计-异步任务

```
def read_events(redis, key, offset=0):
  while True:
    events = redis.execute('xread', 'block', 60, 'count', 100000,
                           'streams', key, offset)
    if not events:
      # timeout
      continue

    _, events = events[0]
    yield events
    offset = events[-1][0]
```

交易系统架构设计-异步任务

```
def task(pg, redis):
  offset = pg.fetch('select offset from task_offset')

  for events in read_events(redis, 'u{btc_usdt}o', offset):
    result = process(events)
    offset = events[-1][0]
    with pg:  # 事务
      save_results(pg, results)
      pg.execute('update task_offset set offset=%s',
                 [offset])
```

交易系统架构设计-关系数据库批量操作

```
insert into trades values
  (1, …)
  (2, …)
  …

psycopg2.extras.execute_values(
  cur,
  ‘insert into trades values %s’,
  data
)

stream = io.StringIO()
# 往stream写csv格式的数据
cur.copy_from(stream, ‘trades’)
```

交易系统架构设计-关系数据库批量操作

```
with tmp(id, amount) as (
  values (1, …),
         (2, …),
         (3, …))
update orders set amount=tmp.amount
  from tmp
 where orders.id=tmp.id;
```

```
delete from orders where id = ANY[%s]
```

交易系统架构设计-部署

- 有状态服务，单独部署集群
  - 数据库
  - **Redis**
- 无状态服务，**k8s/docker swarm**
  - **Restful api**
  - **Websocket api**
  - 异步任务

3 钱包实现

钱包的功能

- 地址生成和私钥管理
- 构建并签名交易
- 监听充值交易的发生
- 跟踪交易的确认数

钱包的实现-使用官方钱包节点的rpc接口

- 难以维护和部署
- 私钥存储不安全
- 冷热钱包分离复杂

钱包的实现-分离地址生成和签名算法

https://github.com/cryptorelay/py-wallet-core

- 可以自行负责私钥存储
- 可以自己实现冷钱包
- 钱包节点只负责区块链数据解析的工作和广播交易
- 甚至可以直接使用第三方节点**API**

THANK YOU

https://cryptorelay.io