Using ROOT to analyse High-Frequency Finance Data

20th International Workshop on Advanced Computing and Analysis Techniques in Physics Research

01-12-2021, Philippe Debie, on behalf of Project HighLO

Collaboration between
1. Wageningen University & Research (WUR)
2. CERN
3. Commodity Risk Management Expertise Centre (CORMEC)

Research goal
1. Describe and detect manipulation of financial markets
2. Help regulators and lawmakers
Limit orders (LO)
- Bid LO = buy for maximum price
- Ask LO = sell for minimum price

Limit order book (LOB)
- Summary of all demand and supply
- Price of any ask LO is greater than price of any bid LO

<table>
<thead>
<tr>
<th>Level</th>
<th>Price</th>
<th>Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>$55.00</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>$54.60</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>$53.50</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>$53.00</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>$50.00</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level</th>
<th>Price</th>
<th>Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$49.00</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>$48.00</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>$47.50</td>
<td>38</td>
</tr>
<tr>
<td>4</td>
<td>$46.00</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>$40.00</td>
<td>1</td>
</tr>
</tbody>
</table>

Spread
Data

- Commodity futures from the Chicago Mercantile Exchange (CME)
- 300TB of messages
  - For each order, for each transaction, etc.
  - Nanosecond timestamp
  - Irregularly spaced
Background info

Message example (FIX protocol)

Level 3 has price $11695 and volume 6
Corn December 2019 contract, with ID 749821
Change a level in the limit order book
Update message at 2019-07-08 15:25:52.248796358 UTC

→ 15TB in ROOT
Finance research

- Data is noisy, irregular in shape, and large in size
- Current storage tools are basic (e.g., csv files)

The power of ROOT

- TFile and TTree are perfect for LOB data
- Transform timeseries into events → Apply HEP statistical methods
Overview

- Counterpart of RDataframe for timeseries
- Templated (LimitOrderBook, TimeNS, Message)
- Keeps track of a state (LimitOrderBook)
- Based on lambda functions
Create a TimeFrame object

```java
TimeFrame timeFrame;

timeFrame.add(chainSoybean);
timeFrame.add(chainCorn);
```
Keep track of the internal state

timeFrame.setStateInitializer([&](int id)
{
    return LimitOrderBook(metaData.at(id).Name, id);
});

timeFrame.setStateUpdater([](int id, TimeNS time, LimitOrderBook& lob, const Message& message)
{
    lob.update(time, message);
});
Data iteration using a TimeFrame

Simple iteration

timeFrame.setForEachRow([&](int id, TimeNS time, const Message& message, const LimitOrderBook& lob) {
    std::cout << lob.getName() << " has " << lob.getTradeVolume() << " transactions so far\n";
});

Making snapshots

timeFrame.setForEachSnapshot(T_Sec * 10, [](TimeNS time, const map<int, LimitOrderBook>& lobs) {
    std::cout << lobs.size() << " internal states tracked at " << nsToTimestamp(time) << '\n';
});
Start iteration

```javascript
timeFrame.run();
```

What happens?

1. Synchronize the 2 chains
2. Build the state for each message
3. Call the lambda functions

→ Constant memory usage
Results – Visualization of limit order book
Trigger-filter-action system
Trigger-filter-action system

timeFrame.setTrigger([&](int id, TimeNS time, const Message& message) {
    return isLOBUpdate(message);
});
timeFrame.setFilter([&](int id, TimeNS time, const Message& message) {
    return true;
});
timeFrame.setAction(-T_Minute, T_Minute, [&](int id, TimeNS time, const Message& triggerMessage,
    const std::list<std::pair<TimeNS, Message>>& msgs) {
    std::cout << msgs.size() << " msgs in the 2 minutes before and after the trigger at " << nsToTimestamp(time) << "'\n';
});
Results – Impact plot
Conclusion

TimeFrame

- Use ROOT with timeseries
- Introduce HEP tools into Finance
References


- https://github.com/HighLO/TimeFrame
Questions