In-Memory Techniques
Low-Latency Trading

Kevin A. Goldstein R.
Neeve Research
Kevin Goldstein

- Live in NYC
- +18 years on Wall St.
- Extensive low latency development for market makers, LL trading shops, Banks…
- Extensive performance tuning for distributed trading applications
  - Head G2 architecture Hedgefund Management System
  - Head of dev USA at FlowTraders
- Sr. Solutions Architect at Neeve Research
- Frequent speaker at Industry Events
  - (10/25 NYC IMC Meetup)
  - (11/14 NYC IMC Meetup HTAP)
Agenda

• Introduce trading systems
• Top concerns for trading systems
• IMC applied to trading systems
• Q & A
Trading System at a Glance:

- Pre-Trade Risk
- Positions
- Pricing engine
- Trade reconciliator
- Drop copier
- LOPER
- OATS
- P&L Calc
- Best Ex.
- Pricing Engine
- Algo Engine
- Order Router
- Market Data
- Arbitrage engine
- Hedger
Basic Order Manager

Client-IN → Market-OUT

Client-OUT → Market-IN
Top Three Requirements for Trading Systems

- **Performance**
  - Low 5-20 microseconds

- **Consistency**
  - Perform the same with 10K mps as with 100K mps
  - 1mic std deviation for I2E

- **Reliability**
  - Message Reliability
  - Survive process and machine failure
IMC Applied to Data Management for Performance

BEFORE:

- **Data**
  - Capture, Refine
  - Store
  - Query
  - Process

- **Compute**
- **Transactions**

- **Does not Scale**
- **Does not Perform**
- **Complex to Author**

**Choke point**
- Complex queries
- Multiple queries
- Very large volumes of data required
- Complex components
## IMC Applied to Data Management for Performance

<table>
<thead>
<tr>
<th>Memory</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Cache</td>
<td>~1ns</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>~3ns</td>
</tr>
<tr>
<td>L3 Cache</td>
<td>~12ns</td>
</tr>
<tr>
<td>Remote NUMA Node</td>
<td>~40ns</td>
</tr>
<tr>
<td>Main Memory</td>
<td>~100ns</td>
</tr>
<tr>
<td>Random SSD Read 4K</td>
<td>150μs</td>
</tr>
<tr>
<td>Data Center Read</td>
<td>500μs*</td>
</tr>
<tr>
<td>Mechanical Disk Seek</td>
<td>10ms</td>
</tr>
</tbody>
</table>

**MEMORY ORIENTED COMPUTING!**

All State in Memory All The Time!

Non Starters For Performance We’re Talking About!

Sources: [https://gist.github.com/jboner/2841832](https://gist.github.com/jboner/2841832)  
IMC Applied to Data Management for Performance

- Ownership: Responsible for updating any consumers
- Publication
- Consumption

How do you consume the data in the most efficient manner possibly?

Data gravity
IMC Applied to Data Management for Performance

AFTER

Data

Capture, Refine

Data at rest

Data (In-Motion)

Capture Data from Source

Refine (aggregate, transform) and route data in motion

Store full data in-memory.

Function (Stateful)

Function co-located with its private state.

Transactions
Ref Data (A) Owner Publishes

Data Ownership – when things change, I’m responsible for updating registered clients
   → No pull for reference data

• Reduce the amount of noise to deal with
• Opens the door for efficient HA
• Much smaller memory footprint
• Faster access times & smaller machines

Mkt data


Algo [N-Z] OR [N-Z']
IM Applied for Reliability, Performance and Consistency

High volume groups

- OR-AMZN
- OR-GOOG
- OR-AAPL
- OR-QQQ,NVDA
- OR-AAPL
- OR-GOOG
- OR-[N-Z]

Low volume groups

- OR-[A-M]
- OR-[N-Z]

Benefits:
- Symbology flexibility
- Hardware risk
- Scaling flexibility

Remote NUMA Node: ~40ns
Main Memory: ~100ns
IM Applied for Performance

AVOID GC

- Pooling is the way to go
- Leverage Off-heap memory
- Actively manage live objects
- Warmups are key
Warmups & POOLING

Warmups are a must:

- Not warmed up
- Warmed up
<table>
<thead>
<tr>
<th>Message Driven</th>
<th>Zero Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stateful</td>
<td>Fully Fault Tolerant</td>
</tr>
<tr>
<td>Multi-Agent</td>
<td>Horizontally Scalable</td>
</tr>
<tr>
<td>Zero Garbage</td>
<td>Ultra Performance</td>
</tr>
</tbody>
</table>
Questions?

kevin@neeveresearch.com  @neeveresearch  @kevgol0  www.neeveresearch.com