High Frequency Trading and NoSQL

Peter Lawrey
CEO, Principal Consultant
Higher Frequency Trading
Agenda

Who are we?
Brief introduction to OpenHFT.
What does a typical trading system look like
What requirements do these systems have
OpenHFT performance.
Who are we

Higher Frequency Trading is a small consulting and software development house specialising in

- Low latency, high throughput software
- 8 developers in Europe and USA.
- Sponsor HFT related open source projects
- Core Java engineering
Who am I?

Peter Lawrey
- CEO and Principal Consultant
- 3rd on Stackoverflow for Java, most Java Performance answers.
- Founder of the Performance Java User's Group
- An Australian, based in the U.K.
What is our OSS

Key OpenHFT projects

- Chronicle, low latency logging, event store and IPC. (record / log everything)
- HugeCollections, cross process embedded persisted data stores. (only need the latest)

Millions of operations per second.
Micro-second latency.
With other NoSQL databases

Uses with NoSQL

- Off heap cache of data from a DB.
- Low latency queue for persisting to a DB
- Map which replicates only the latest values. Supports very high update rates by only replicating the latest value.
What is HFT?

- No standard definition.
- Trading faster than a human can see.
- Being fast can make the difference between making and losing money.
- For different systems this means typical latencies of between
  - 10 micro-seconds and
  - 10 milli-second.
(Latencies external to the provider)
Time scales every developer should know.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>In human terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Cache hit</td>
<td>1 ns</td>
<td>A blink of an eye (~20 ms)</td>
</tr>
<tr>
<td>L2 Cache hit</td>
<td>3 ns</td>
<td>Noticeable flicker</td>
</tr>
<tr>
<td>L3 Cache hit</td>
<td>10 – 20 ns</td>
<td>Time to say “A”</td>
</tr>
<tr>
<td>Main memory</td>
<td>70 – 100 ns</td>
<td>Time to say a ten word sentence</td>
</tr>
<tr>
<td>Signal down a 200m fibre cable</td>
<td>1 μsec</td>
<td>One slide (speaking quickly)</td>
</tr>
<tr>
<td>SSD access</td>
<td>5 – 25 μsec</td>
<td>Time to reheat a meal (3 mins)</td>
</tr>
<tr>
<td>HDD access</td>
<td>8 msec</td>
<td>Time to flight around the world. (1.8 days)</td>
</tr>
<tr>
<td>Network packet from Germany to the USA</td>
<td>45 msec</td>
<td>Waiting for a 7 working day delivery</td>
</tr>
</tbody>
</table>
Simple Trading System

Market Data → Configuration Service

Market Data Parser → Strategies

Time Series + logger → Custom Graph

CEP logger → Order Manager

Trading Service → SQL + logger
Event driven processing

Trading system use event driven processing to minimise latency in a system.

- Any data needed should already be loaded in memory, not go off to a slow SQL database.
- Each input event triggers a response, unless there is a need to limit the output.
Critical Path

A trading system is designed around the critical path. This has to be as short in terms of latency as possible.

- Critical path has a tight latency budget which excludes many traditional databases.
- Even the number of network hops can be minimised.
- Non critical path can use tradition databases
Critical Path databases

- Time Series databases
  - Kdb, kona
  - InfluxDB
  - OpenTSDB

Designed for millions of writes per second.

Column based database => 100 Million operations per second e.g. sum a column.
## Critical Path Databases

The table below shows the mean response times for various tasks on a 1 TB and 6 TB system, along with the speedup.

<table>
<thead>
<tr>
<th>Task Description</th>
<th>1 TB System (KDB140206)</th>
<th>6 TB System (KDB140116)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>10T.MKTSNAP/s.TIME</td>
<td>2,569</td>
<td>57</td>
<td>44.8</td>
</tr>
<tr>
<td>10T.STATS-AGG/s.TIME</td>
<td>19,213</td>
<td>2,871</td>
<td>6.7</td>
</tr>
<tr>
<td>50T.STATS-UI/s.TIME</td>
<td>17,707</td>
<td>3,291</td>
<td>5.4</td>
</tr>
<tr>
<td>1T.YRHBID/s.TIME</td>
<td>1,213</td>
<td>287</td>
<td>4.2</td>
</tr>
<tr>
<td>100T.VWAB-12D-NO/s.TIME</td>
<td>17,150</td>
<td>4,348</td>
<td>3.9</td>
</tr>
<tr>
<td>100T.STATS-UI/s.TIME</td>
<td>15,145</td>
<td>6,293</td>
<td>2.4</td>
</tr>
<tr>
<td>1T.THEOPL/s.TIME</td>
<td>81</td>
<td>46</td>
<td>1.7</td>
</tr>
<tr>
<td>10T.VOLCURV/s.TIME</td>
<td>14,910</td>
<td>10,253</td>
<td>1.5</td>
</tr>
<tr>
<td>1T.YRHBID-2/s.TIME</td>
<td>339</td>
<td>280</td>
<td>1.2</td>
</tr>
<tr>
<td>1T.STATS-UI/s.TIME</td>
<td>222</td>
<td>227</td>
<td>1.0</td>
</tr>
<tr>
<td>1T.VWAB-D/s.TIME</td>
<td>21</td>
<td>22</td>
<td>0.9</td>
</tr>
<tr>
<td>1T.WKHIBID/s.TIME</td>
<td>24</td>
<td>27</td>
<td>0.9</td>
</tr>
<tr>
<td>1T.MOHIBID/s.TIME</td>
<td>43</td>
<td>50</td>
<td>0.9</td>
</tr>
<tr>
<td>10T.STATS-UI/s.TIME</td>
<td>716</td>
<td>940</td>
<td>0.8</td>
</tr>
<tr>
<td>1T.QTRHIBID/s.TIME</td>
<td>51</td>
<td>77</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Source: www.STACresearch.com

February 18, 2014
Critical Path data store

HFT strategies are;
- described using graphs.
- handle events in real time \(\sim 10 - 100 \ \mu\text{sec}.\)
- cache state rather than query a database.
- all custom written libraries AFAIK.
Critical Path data store

Logging is performed by appending to memory mapped files.

OpenHFT's Java Chronicle makes this easier to do in Java in a GC-free, off heap, lock less way.

Such low level coding is relatively easy in C or C++.
Non-critical Datastore

Configuration management
- ZooKeeper, etcd
- Plain files with Version control
- LDAP
- Any distributed key-value store. e.g. MongoDB
Big Data

Back testing a HFT system is critical and a number of solutions are available

- Hadoop
- Matlab
- Time series
- R
Operational Infrastructure

Control and management infrastructure

- JMS, JMX
- Tibco RV, LBM
- Terracotta
- MongoDB
Reliable persistence

Trades and Orders are high value data and less voluminous than Market data or strategy results.

- Typically SQL Database.
- Sometimes multiple databases for different applications.
Why use more exotic database?

- Mostly for high throughput.
  - Million per second in one node.
- Often for low latency.
  - Latencies well below a milli-second.
Why wouldn't you use exotic DB

- Not easy to learn, high knowledge investment.
- Often harder to use.
  - Less management tools.
  - Not designed to work with web applications.
- More sensitive to the details of the hardware and what else is running on the same machine.
Low latency at high throughput

Java Chronicle is designed as a low latency logger and IPC. At one million small messages per second:

- Almost zero garbage
- Latency between processes around 1 micro-second
- Concurrent readers and writers

Supports bursts of 10 million messages/sec.
Chronicle and replication

Replication is point to point (TCP)

Server A records an event
- replicates to Server B

Server B reads local copy
- B processes the event

Server B stores the result.
- replicates to Server A

Server A replies.

Round trip
25 micro-seconds
99% of the time

GC-free
Lock less
Off heap
Unbounded
HugeCollections

HugeCollections provides key-value storage.

- Persisted (by the OS)
- Embedded in multiple processes
- Concurrent reads and writes
- Off heap accessible without serialization.
HugeCollections and throughput

SharedHashMap tested on a machine with 128 GB, 16 cores, 32 threads.
String keys, 64-bit long values.

- 10 million key-values updated at 37 M/s
- 500 million key-values updated at 23 M/s
- On tmpfs, 2.5 billion key-values at 26 M/s
HugeCollections and latency

For a Map of small key-values (both 64-bit longs)
With an update rate of 1 M/s, one thread.

<table>
<thead>
<tr>
<th>Percentile</th>
<th>100K entries</th>
<th>1 M entries</th>
<th>10 M entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>50% (typical)</td>
<td>0.1 μsec</td>
<td>0.2 μsec</td>
<td>0.2 μsec</td>
</tr>
<tr>
<td>90% (worst 1 in 10)</td>
<td>0.4 μsec</td>
<td>0.5 μsec</td>
<td>0.5 μsec</td>
</tr>
<tr>
<td>99% (worst 1 in 100)</td>
<td>4.4 μsec</td>
<td>5.5 μsec</td>
<td>7 μsec</td>
</tr>
<tr>
<td>99.9%</td>
<td>9 μsec</td>
<td>10 μsec</td>
<td>10 μsec</td>
</tr>
<tr>
<td>99.99%</td>
<td>10 μsec</td>
<td>12 μsec</td>
<td>13 μsec</td>
</tr>
<tr>
<td>worst</td>
<td>24 μsec</td>
<td>29 μsec</td>
<td>26 μsec</td>
</tr>
</tbody>
</table>
A peak time an application writes 49 “mb/s” to a disk which supports 50 “mb/s” and is replicated over a 100 “mb/s” network.

What units were probably intended and where would you expect buffering if any?
Bonus topic: Units

A peak times an application writes 49 MiB/s to a disk which supports 50 MB/s and is replicated over a 100 Mb/s network.

MiB = $1024^2$ bytes

MB = $1000^2$ bytes

Mb = 125,000 bytes

The 49 MiB/s is the highest rate and 100 Mb/s is the lowest.
## Bonus topic: Units

<table>
<thead>
<tr>
<th>Unit</th>
<th>Bandwidth</th>
<th>Used for</th>
</tr>
</thead>
<tbody>
<tr>
<td>mb - miili-bit</td>
<td>mb/s – milli-bits per second</td>
<td>?</td>
</tr>
<tr>
<td>mB - milli-byte</td>
<td>mB/s – milli-bytes per second</td>
<td>?</td>
</tr>
<tr>
<td>kb – kilo-bit (1000)</td>
<td>kb/s – kilo-bits (baud) per second</td>
<td>Dial up bandwidth</td>
</tr>
<tr>
<td>kB – kilo-byte (1000)</td>
<td>kB/s – kilo-bytes per second</td>
<td>?</td>
</tr>
<tr>
<td>Mb – mega-bit (1000^2)</td>
<td>Mb/s – mega-bits (baud) per second</td>
<td>Cat 5 ethernet</td>
</tr>
<tr>
<td>MB - mega-byte (1000^2)</td>
<td>MB/s – mega bytes per second</td>
<td>Disk bandwidth</td>
</tr>
<tr>
<td>Mib – mibi-bit (1024^2)</td>
<td>Mib – Mibi-bits per second</td>
<td>?</td>
</tr>
<tr>
<td>MiB – mibi-byte (1024^2)</td>
<td>MiB – Mibi-bytes per second</td>
<td>Memory bandwidth</td>
</tr>
<tr>
<td>Gb – giga-bit (1000^3)</td>
<td>Gb/s – giga-bit (baud) per second</td>
<td>High speed networks</td>
</tr>
<tr>
<td>GB – giga-byte (1000^3)</td>
<td>GB/s – giga-byte per second</td>
<td>-</td>
</tr>
<tr>
<td>Gib – gibi-bit (1024^3)</td>
<td>Gib/s – gibi-bit per second</td>
<td>-</td>
</tr>
<tr>
<td>GiB – gibi-byte (1024^3)</td>
<td>GiB/s – gibi-byte per second.</td>
<td>Memory Bandwidth</td>
</tr>
</tbody>
</table>
Q & A

https://github.com/OpenHFT/OpenHFT

@PeterLawrey

peter.lawrey@higherfrequencytrading.com