Low level Java programming With examples from OpenHFT

Peter Lawrey

CEO and Principal Consultant

Higher Frequency Trading.

Presentation to Joker 2014 St Petersburg, October 2014.

About Us...

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

- Low latency, high throughput software
- 6 developers + 3 staff in Europe and USA.
- Sponsor HFT related open source projects
- Core Java engineering

HFT

About Me…

- CEO and Principal Consultant
- Third on Stackoverflow for Java, most Java Performance answers.
- Vanilla Java blog with 3 million views
- Founder of the Performance Java User's Group
- An Australian, based in the U.K.

HFT

What Is The Problem We Solve?

"I want to be able to read and write my data to a persisted, distributed system, with the speed of in memory data structures"

OpenHFT

Agenda

What are our Open Source products used for?

Where to start with low latency?

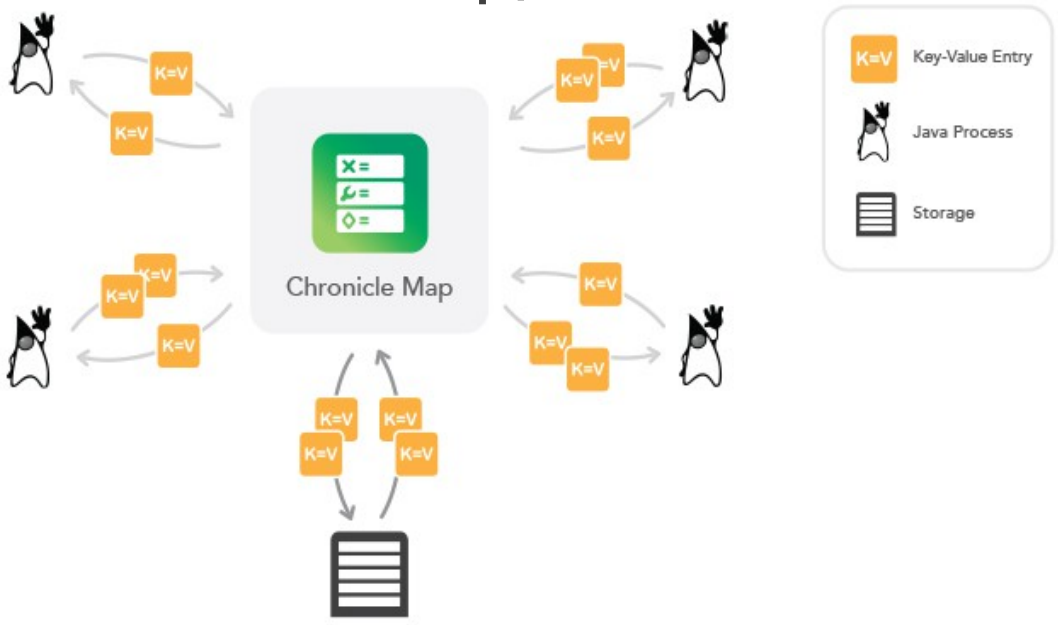
When you know you have a problem, what can you do?

Chronicle scaling Vertically and Horizontally

- Shares data structure between processes
- Replication between machines
- Build on a low level library Java Lang.
- Millions of operations per second.
- Micro-second latency. No TCP locally.
- Synchronous logging to the OS.
- Apache 2.0 available on GitHub
- Persisted via the OS.

OpenHFT

What is Chronicle Map/Set?

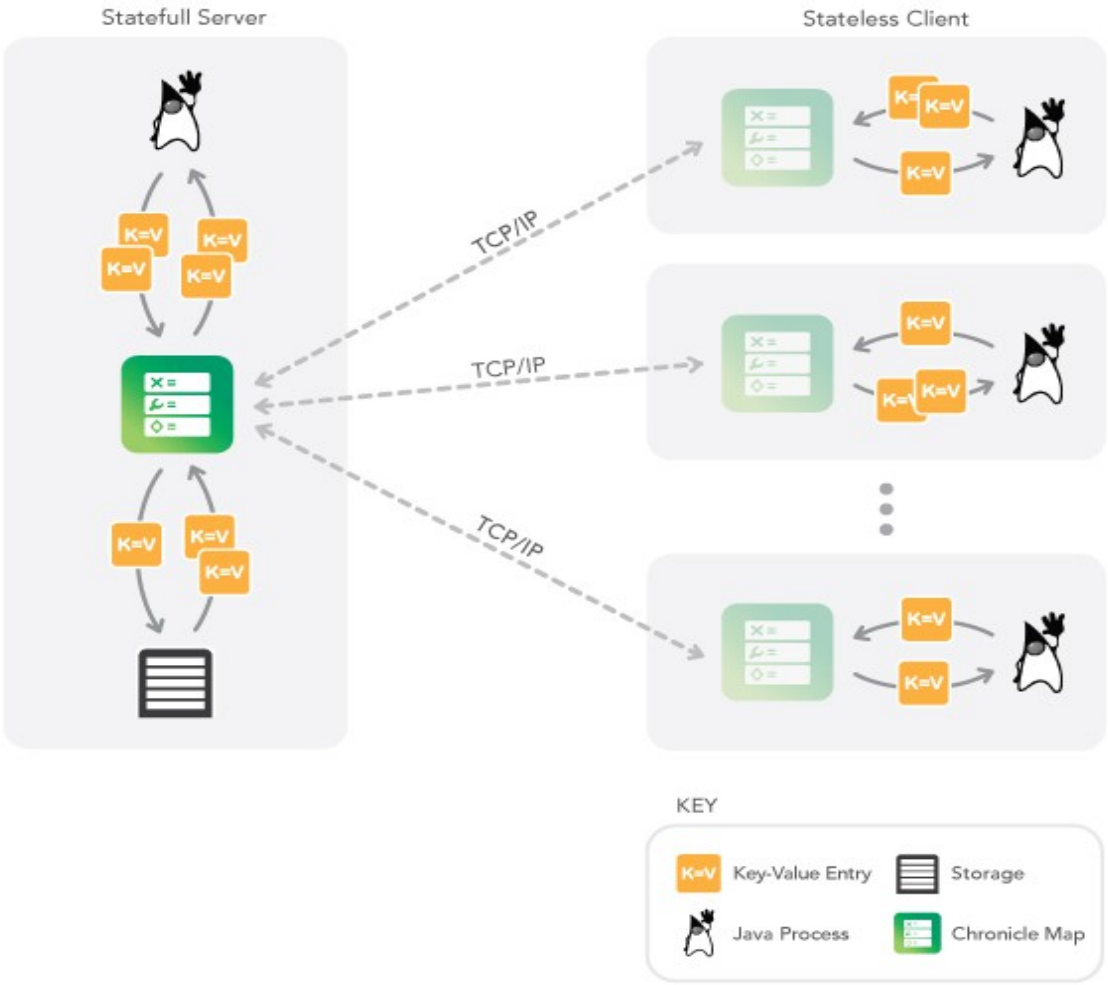


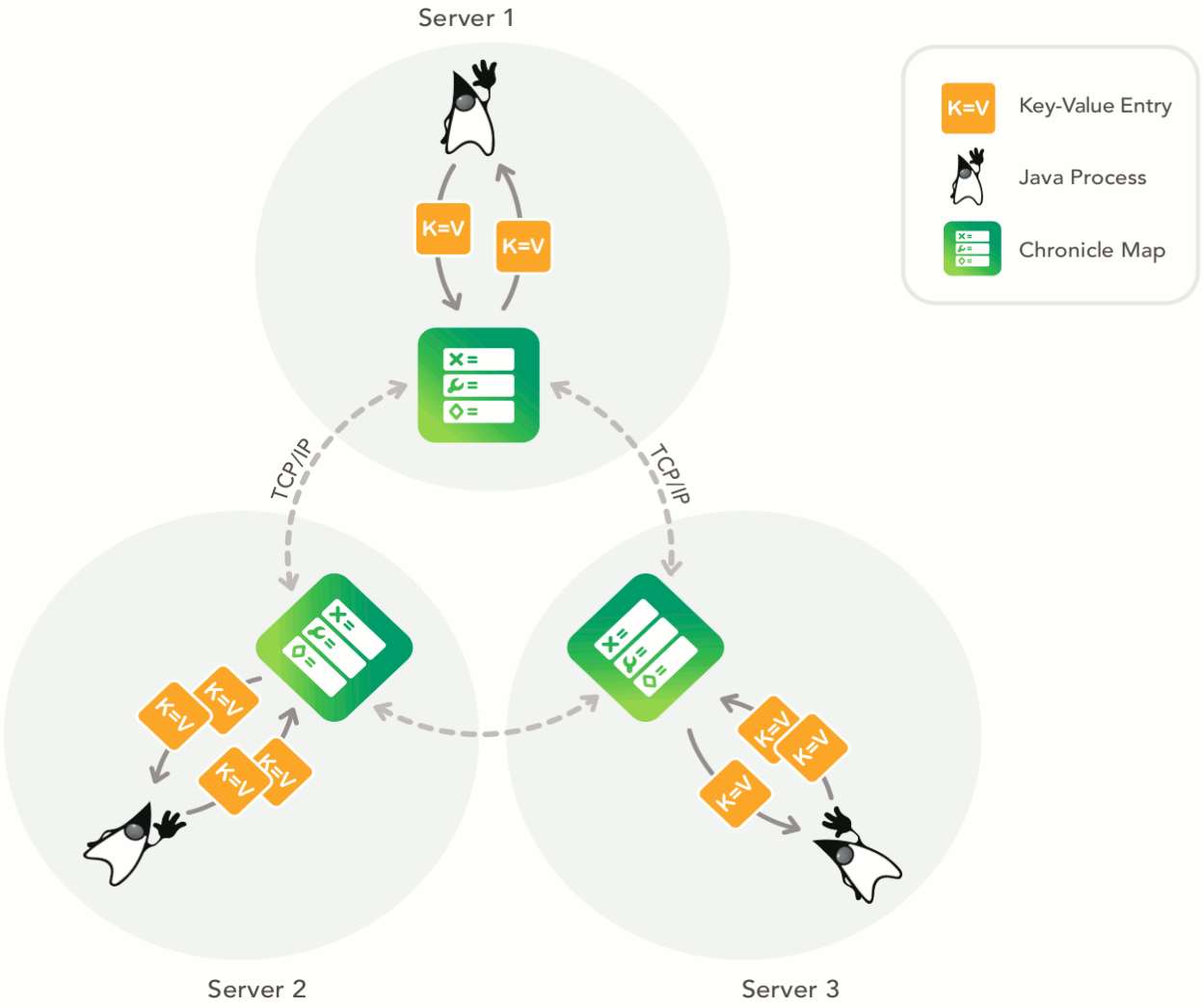
- Low latency persisted key-value store.
- Latency between processes around 200 ns.
- In specialized cases, latencies < 25 ns.
- Throughputs up to 30 million/second.

What is Chronicle Map/Set?

- ConcurrentMap or Set interface
- Designed for reactive programming
- Replication via TCP and UDP.
- Apache 2.0 open source library.
- Pure Java, supported on Windows, Linux, Mac OSX.

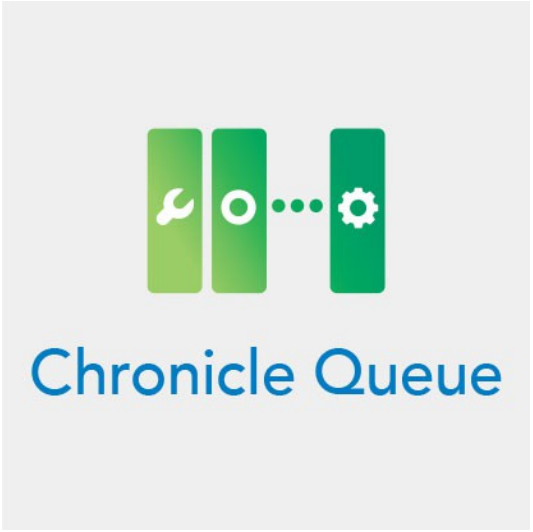
OpenHFT


Statefull Server
Stateless Client
K=V
TCP/IP
TCP/IP
TCP/IP
KEY
Key-Value Entry
Storage
Java Process
Chronicle Map


Server 1
K=V
K=V
TCP/IP
TCP/IP
Server 2
Server 3
K=V
Key-Value Entry
Java Process
Chronicle Map

What is Chronicle Queue?

- Low latency journaling and logging.
- Low latency cross JVM communication.
- Designed for reactive programming
- Throughputs up to 40 million/second.

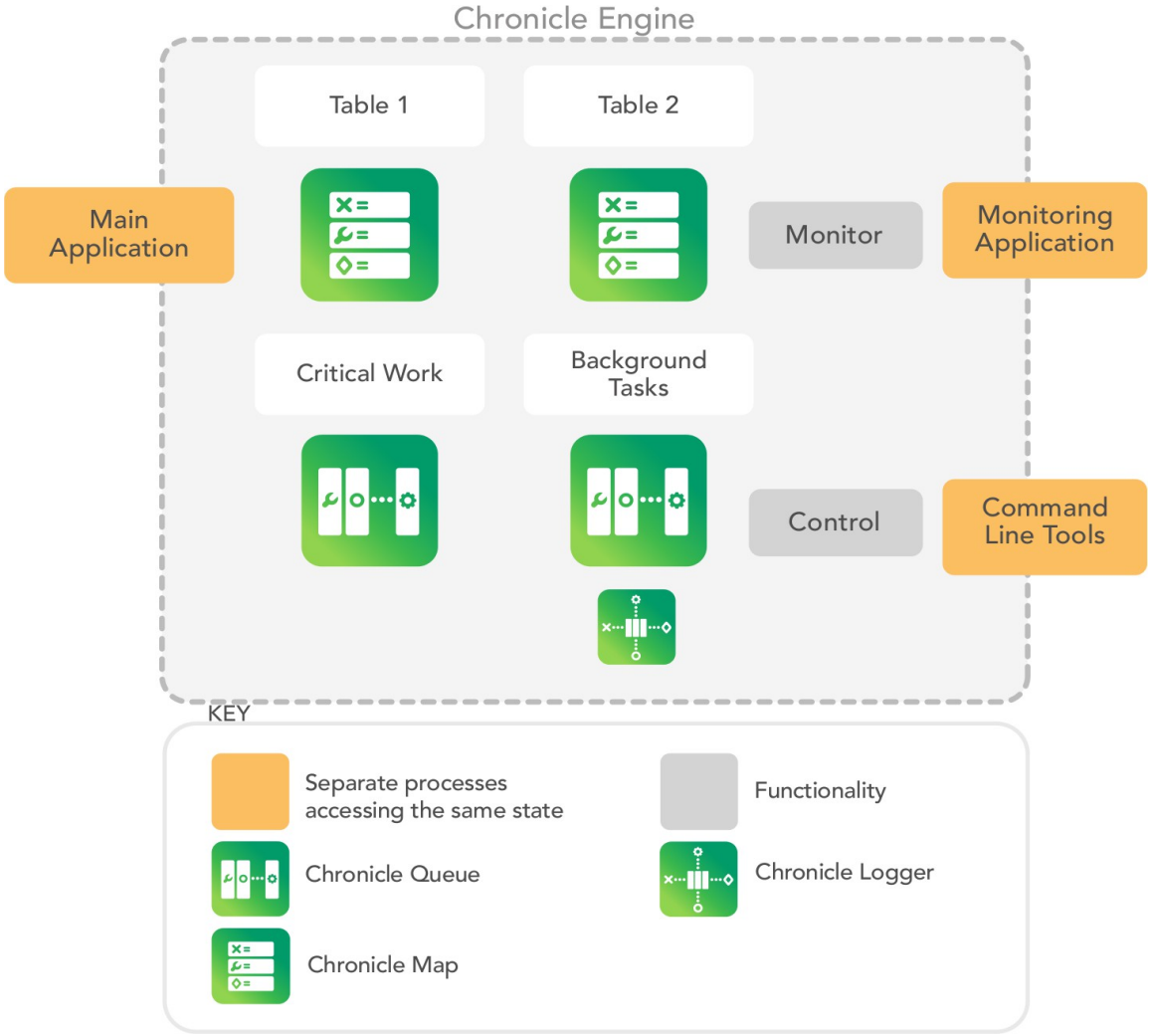


What is Chronicle Queue?

- Latencies between processes of 200 nano-seconds.
- Sustain rates of 400 MB/s, peaks much higher.
- Replication via TCP.
- Apache 2.0 open source library.
- Pure Java, supported on Windows, Linux, Mac OSX.

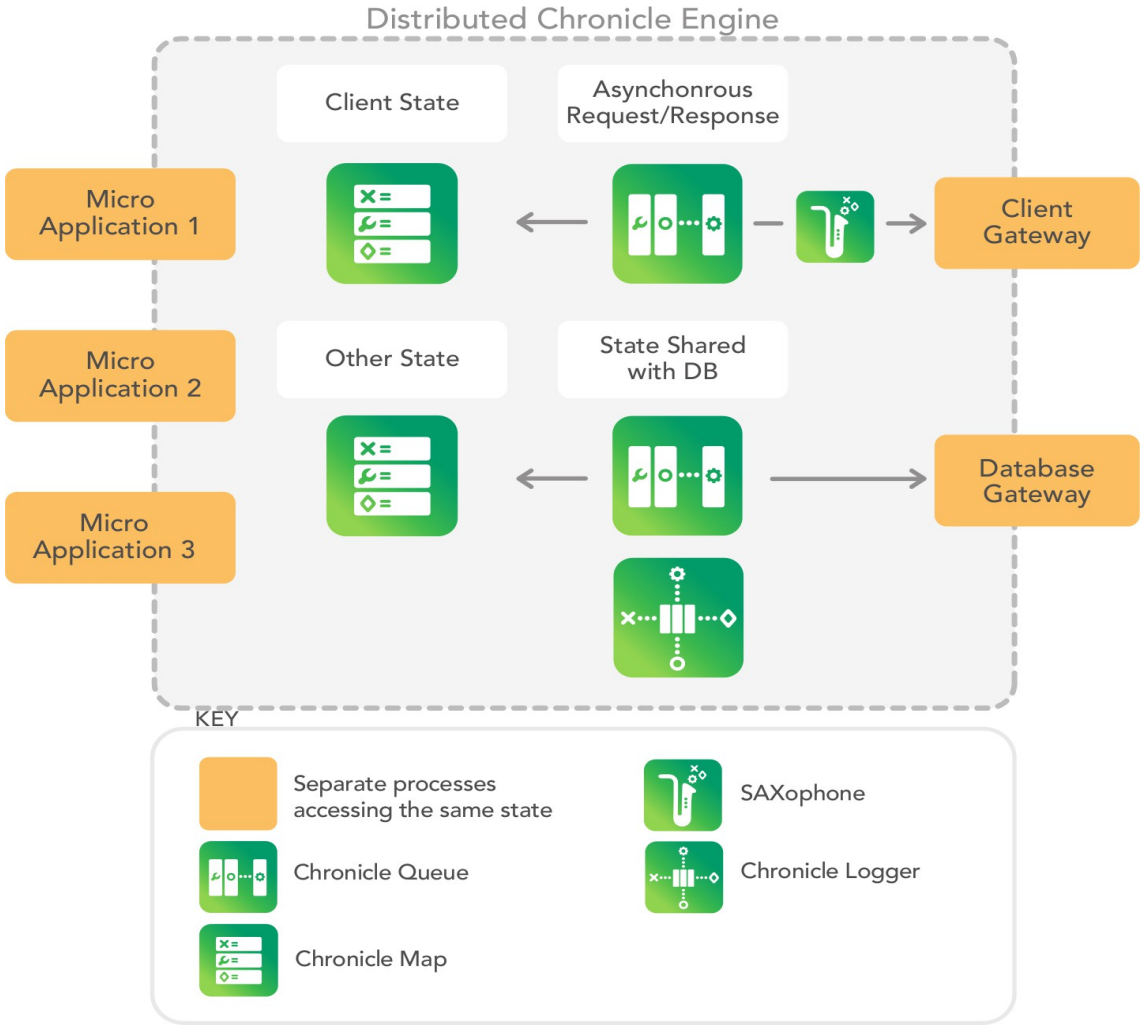
OpenHFT

Chronicle monitoring a legacy application



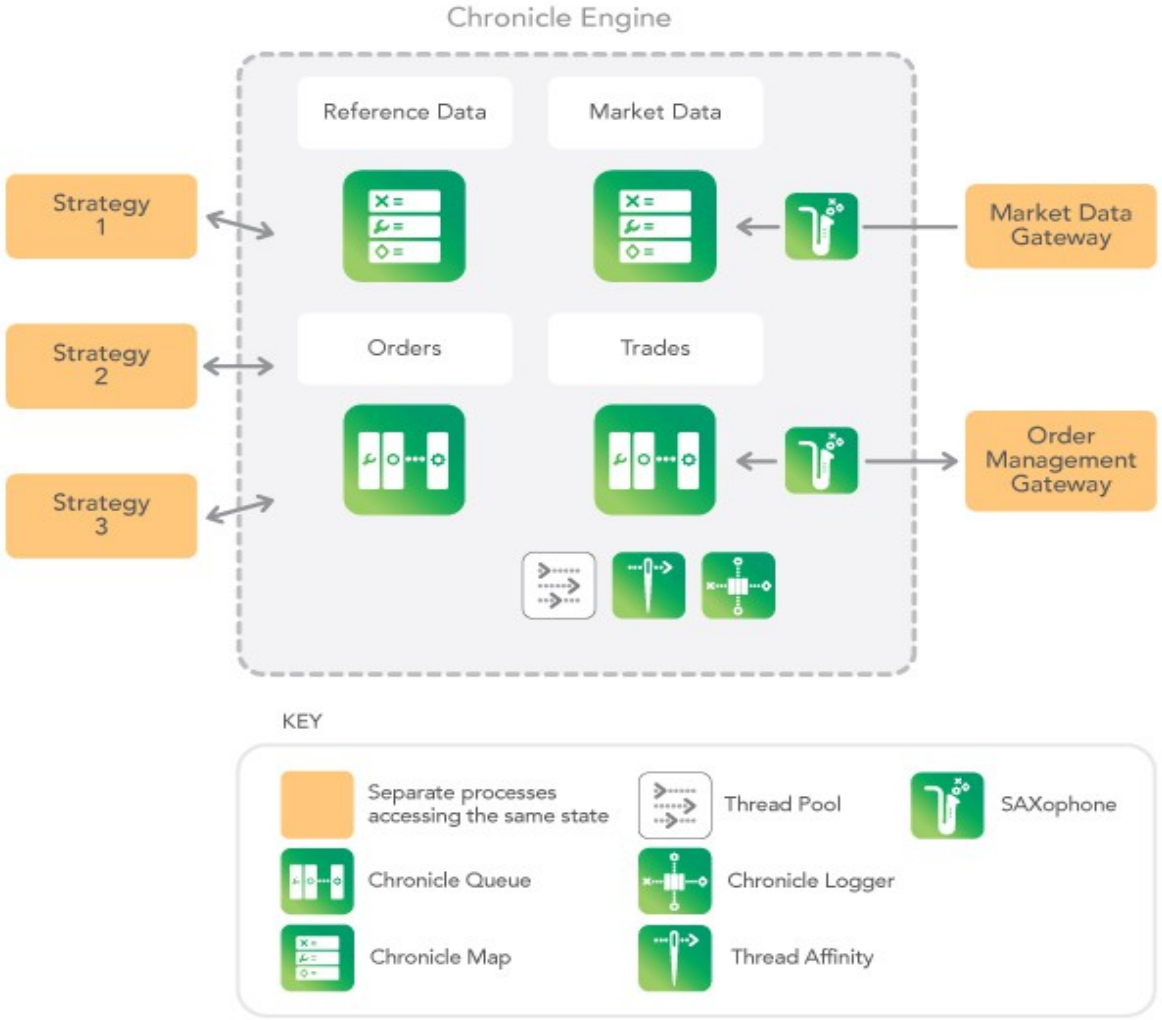
OpenHFT

Chronicle journalling multiple applications



OpenHFT

Chronicle for low latency trading



OpenHFT

Short demo using OSResizesMain

```
peter@chronos: ~
top - 17:04:58 up 19:25,  2 users,  load average: 0.94, 0.96, 0.69
Tasks: 220 total,   1 running, 219 sleeping,   0 stopped,   0 zombie
Cpu(s): 15.8%us, 19.3%sy,  0.2%ni, 64.5%id,  0.2%wa,  0.0%hi,  0.1%si,  0.0%st
Mem:   7690672k total,  5767192k used,  1923480k free,    84588k buffers
Swap: 15788028k total,   197112k used, 15590916k free,  1456056k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
17599 peter     20   0  125t 192m 108m S   99  2.6   0:06.40 java
17066 peter     20   0 5696m 1.3g  33m S   27 17.2   4:43.45 java
```

Note: The “VIRT” virtual memory size is 125t for 125 TB, actual usage 97M
System memory: 7.7 GB, Extents of map: 137439.0 GB, disk used: 97MB,
addressRange: 233a777f000-7f33a8000000
$ ls -lh /tmp/oversized*
-rw-rw-r-- 1 peter peter **126T** Oct 20 17:03 /tmp/over-sized...
$ du -h /tmp/oversized*
97M /tmp/over-sized....

OpenHFT

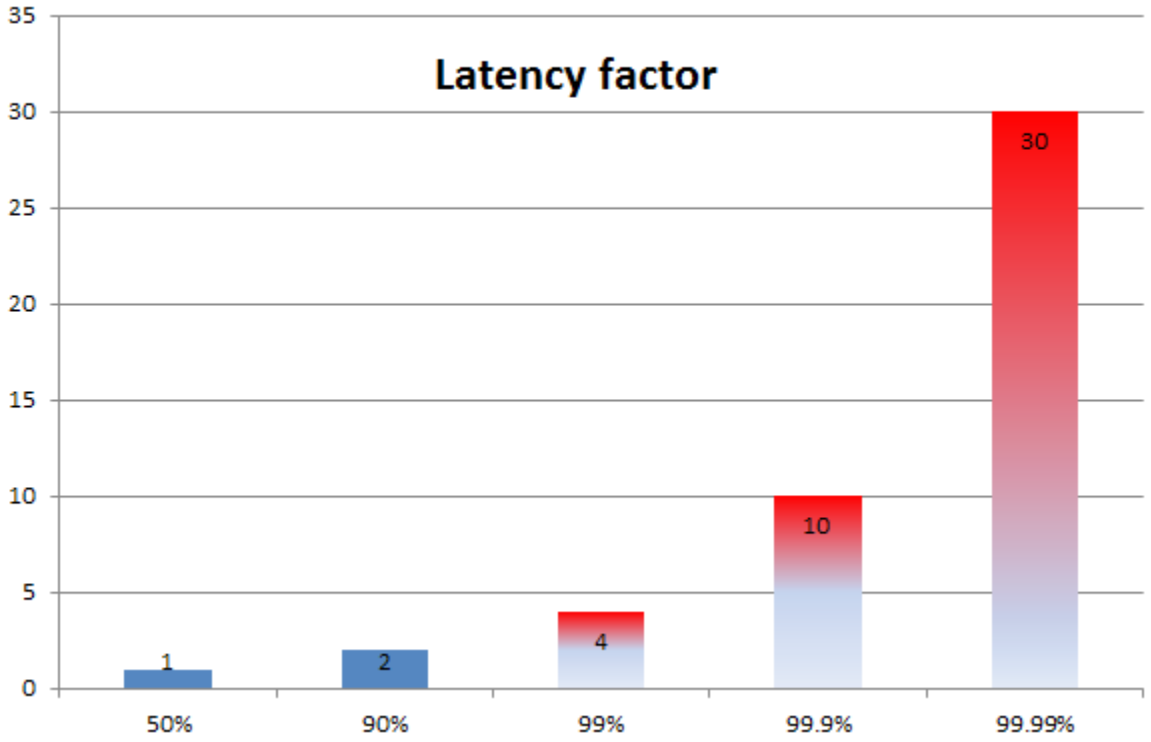
Where to start with low latency?

You need to measure first.

- What is the end to end use case you need to improve?
- Is it throughput or latency you need to improve?
- Throughput or average latency hides poor latencies.
- Avoid co-ordinated omission. See Gil Tene's talks.

OpenHFT

Looking at the latency percentiles



Tools to help you measure

- A commercial profiler. e.g. YourKit.
- Instrument timings in production.
- Record and replay production loads.
- Avoid co-ordinated omission. See Gil Tene's talks.
- If you can't change the code, Censum can help you tune your GC pause times.
- Azul's Zing “solves” GC pause times, but has many other tools to reduce jitter.

What to look for when profiling

- Reducing the allocation rate is often a quick win.
- Memory profile to reduce garbage produced.
- When CPU profiling, leave the memory profiler on.
- If the profiler is no long helpful, application instrumentation can take it to the next level.

OpenHFT

When you know you have a problem, what can you do about it?

Is garbage unavoidable?

- You can always reduce it further and further, but at some point it's not worth it.
- For a web service, 500 MB/s might be ok.
- For a trading system, 500 KB/s might be ok.
- If you produce 250 KB/s it will take you 24 hours to fill a 24 GB Eden space.

Common things to tune.

A common source of garbage is Iterators.

```
for (String s : arrayList) { }
```

Creates an Iterator, however

```
for (int i = 0, len = arrayList.size(); i < len; i++) {
    String s = arrayList.get(i);

}
```

Doesn't create an Iterator.

Common things to tune.

BigDecimal can be a massive source of garbage.

```
BigDecimal a = b.add(c)
                .divide(BigDecimal.TWO, 2, ROUND_HALF_UP);
```

The same as double produces no garbage.

```
double a = round(b + c, 2);
```

You have to have a library to support rounding. Without it you will get rounding and representation errors.

Be aware of your memory speeds.

	concurrency	Clock cycles	Seconds
L1 caches	multi-core	4	1 seconds
L2 cache	multi-core	10	3 seconds
L3 cache	socket wide	40-75	15 seconds
Main memory	System wide	200	50 seconds.
SSD access	System wide	50,000	14 hours
Local Network	Network	180,000	2 days
HDD	System wide	30,000,000	1 year.

To maximise performance you want to spend as much time in L3, or ideally L1/L2 caches as possible.

Memory access is faster with less garbage

Reducing garbage minimises filling your caches with garbage.

If you are producing 300 MB/s of garbage your L1 cache will be filled with garbage is about 100 micro-seconds, your L2 cache will be filled in under 1 milli-second.

The L3 cache and main memory shared and the more you use this, the less scalability you will get from your multi-cores.

Faster memory access

- Reduce garbage
- Reduce pointer chasing
- Use primitives instead of objects.
- Avoid false sharing for highly contended mutated values

Lock free coding

- AtomicLong.addAndGet(n)
- Unsafe.compareAndSwapLong
- Unsafe.getAndAddLong

OpenHFT

Using off heap memory

- Reduce GC work load.
- More control over layout
- Data can be persisted
- Data can be embedded into multiple processes.
- Can exploit virtual memory allocation instead of main memory allocation.

Low latency network connections

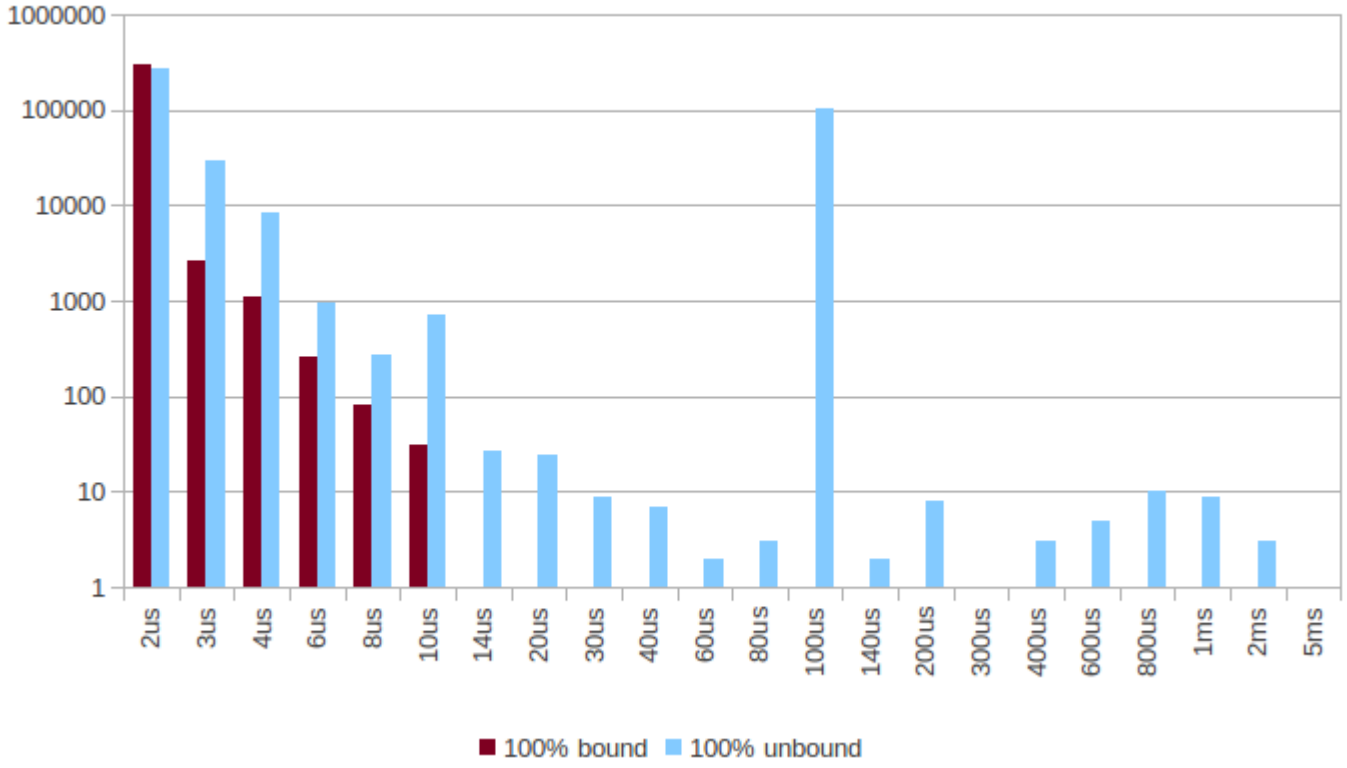
- Kernel by pass network cards e.g. Solarflare
- Single hop latencies around 5 micro-seconds
- Make scaling to more machines practical when tens of micro-seconds matter.

Reducing micro-jitter

- Unless you isolate a CPU, it will get interrupted by the scheduler a lot.
- You can see delays of 1 – 5 ms every hour on an otherwise idle machine.
- On a virtualised machine, you can see delays of 50 ms.
- Java Thread Affinity library can declaratively layout your critical threads.

Reducing micro-jitter

Number of interrupts per hour by length.



OpenHFT

Q & A

http://openhft.net/
Performance Java User's Group.
@PeterLawrey
peter.lawrey@higherfrequencytrading.com