LMAX Exchange Getting Up To 50% Improvement in Latency From Azul's Zing JVM

by **Charles Humble** on May 07, 2013 - **view original article here**

Developers at the **LMAX Exchange**, an execution venue for trading FX established in the City of London in October 2010, have begun testing **Azul's Zing** JVM as a way of improving their already impressive response times and throughput rates.

LMAX Exchange has become well known, even outside of financial services, at least in part by dint of being willing to talk publicly about technology choices. The firm open-sourced a key component of its software stack, the **Disruptor framework**, in March 2011 and has **spoken at QCon** and **other conferences** about its work.

At the end of last year, LMAX Exchange completed the latest re-build of its production data centre, moving from Hewlett-Packard servers with Intel's HexaCore Dunnington processor to Dell 520 and 720 with dual socket, 8 core, Intel Sandy Bridge CPUs. The Dells have 64 and 128 GB of memory respectively. "The servers are commodity, with not much specific tuning," Michael Barker, Head of Software at LMAX Exchange, told InfoQ.

> *For our performance critical journal based storage we use 15k SAS raid with battery backed write cache. The tests we did showed that a raid array was as fast/faster than SSDs for our workload, which is testament to the quality of the journalling code.*
> *We use flash in some other areas.*

At the same time as updating the servers the firm switched from CentOS 5 to CentOS 6 as the operating system. "One of the reasons to stay with CentOS (as opposed to a distribution like Fedora Core) is that it is a supported platform for Azul Zing," Barker told us. Currently though, LMAX Exchange is using Oracle's HotSpot JVM in production.

The servers are running in an **Equinix** data centre west of London, which is where a lot of other financial service firms have their servers and applications, so the latency hop between the systems is comparatively small. The firm also has a DR data centre in London, and a third site at a hosting centre in north London which holds the test and development servers along with a Jenkins based Continuous Integration system using the **OpenStack** private cloud. "We also make extensive use

of Cloud services from Amazon and Rackspace," Baker told us, "for those systems that don't hold customer data and are not latency sensitive."

The firm handles an average load of 1,000-2,000 orders per second with peaks of over 5,000. On rare occasions it has gone to over 20,000 orders/second. "We test using our peak load as the sustained baseline, and regularly run tests to find the point at which the system breaks (so called 'Red Line' tests)."

LMAX Exchange **publish data** on execution performance; order latency is 1.5ms on average, whilst the trade latency is less than 3ms end to end, including real time pre-trade risk control.

Whilst these figures are impressive, the variations, caused primarily by stop-the-world pauses in the CMS collector that is part of HotSpot, are becoming a significant problem. LMAX Exchange tried upgrading to the CMS version in JDK 7, but encountered around a 20% increase in the length of GC pauses for the same work load. The reasons for this weren't entirely clear, but Barker suggested it was probably down to a need to re-tune the collector. That Zing's collector (C4) typically requires little or no tuning was a major selling point for LMAX Exchange.

> *I think that we really needed to do retuning of our GC setting and investigating whether JDK 7 specific options like -XX:+UseCondCardMark and -XX:+UseNUMA should be applied. One of the other big reasons to go with Azul is the reduced need to tune the collector. The general recommendation is that you should re-tune from scratch on each new version of the JDK, which sounds fine in theory, but can be impractical. Collector tuning in Oracle JDK is essentially walking through a large search space for a result that meets your needs. Experience, knowledge and guess-work can crop significant chunks off that search space, but even then an extensive tuning exercise can take weeks. For example, our full end-to-end performance test takes 1 hour (10 minutes build & deploy, 10 minutes warm-up, 40 minutes testing), so I could reasonably run 8 runs a day. If you consider the number of different collectors (CMS, Parallel, Serial,...) and all of their associated options (new and old sizes, survivor spaces, survivor ratios,...) how many runs do I need to do to get effective coverage of that search space: 20, 30, more? With Zing the defaults work significantly better than a finely tuned Oracle JDK. We still have some investigation over whether we can get a bit more out of the Zing VM through tuning (e.g. fewer collector threads as our allocation rate is relatively low). However, tuning Zing is just that, i.e. looking to eke out the very best from the system; compared to the Oracle JDK where tuning from the defaults can be the difference between usable and unusable. The effort involving in tuning does come with an opportunity cost. I*

> *would much rather have the developers that would typically be involved with GC tuning (they are probably the ones that have the best working knowledge of software performance) be focusing on improving the performance of other areas of the system.*

Another option would be to look at two newer collectors - IBM's Balanced GC and Oracle's G1. Though the two were developed independently they appear remarkably similar, and aim to provide consistent low pauses over time. They use incremental compaction, a technique which assumes that some regions of memory are more popular than others, allowing the collector to compact a single region at a time and only scan the regions pointing into it when remapping all potential references. This hypothesis may be true for a majority of applications, but it isn't universally true. Moreover, whilst both collectors have a mostly concurrent marker, and a mostly incremental compaction for the old generation, they also have a fall-back to a stop-the-world collector. Finally, in common with every other commercial collector apart from Azul's C4, they use a non-concurrent, stop-the-world collector for the young generation. By contrast, Azul's C4 uses the same fully concurrent, compacting algorithm across both the young and old generation with no fall-back. A **previous InfoQ** article explored the algorithm in some detail.

Given this, LMAX Exchange didn't try either G1 or Balanced GC. Barker told us, "Both still suffer from the same fundamental problem that CMS does, in that whilst they can delay potential pauses they don't eradicate them completely." Therefore

> *...of the GC implementations available on the market, the only other options that look [feasible] are IBM's **Metronome** collector and JRockit's Real Time collector. Metronome is quite interesting and I would have liked to spend some more time looking into it, however I ran into a couple of issues and I wasn't able to test it within the time window I had. In IBM's defence - we were pushing quite hard into some corners of the JDK that aren't used by most applications and my report got in front the right people and was fixed quickly. With JRockit Real Time, unfortunately Oracle are not continuing on with its development and it won't be moving beyond version JDK 1.6, which felt like a dead-end as a technology choice.*
>
> *Azul Zing was the one that worked and it showed positive results, that and the team over at Azul were really keen to work closely with us to improve the product for our use case, which sold me on the product.*

The results LMAX Exchange are seeing are remarkable: a 10-20% improvement in the mean latency, increasing to around a 50% improvement at the 99th percentile. Moreover

> *At the max/99.99th percentile with HotSpot the number would jump all over the place so it is hard to produce a relative comparison, except to say that the Zing values are much more stable. A bad run with HotSpot could easily be an order of magnitude worse.*

In terms of throughput

> *Zing gives us the ability to lift what we call our "red-line" - the throughput value at which the latency starts to drop off a cliff. This effect often manifests as a second order effect of GC pauses. If we get a stall that is sufficiently long, we will start to drop packets. The process of packet redelivery can create back-pressure throughout the rest of the system sending client latencies through the roof. Having a more efficient collector with very short predictable pauses should allow us to increase our "red-line".*

Barker points out that there are plenty of other problems which can contribute to variance in performance - bad code, OS jitter, lock contention, and lack of bandwidth between LMAX Exchange and its customers being some examples. "However, GC was the at the top of the list and as with most performance tuning exercises you start with the biggest offender first."

Azul has seen good take-up amongst financial services companies for Zing. Indeed Azul CTO Gil Tene told InfoQ that whilst LMAX Exchange is well known for their technological excellence and leading edge performance, it isn't the most aggressive low latency customer Azul has in the industry - "We have several that are even larger and more demanding in latencies (e.g. the latency bounds in Equities tend to be more demanding than they are in FX)."

Part of the reason Zing is so attractive to these companies is that it remains the only collector that eliminates stop-the-world pauses from the young generation as well as the old generation. Whilst young generation pauses are shorter, where an application is particularly performance sensitive they still matter. As a result, Tene told us, "All we have to do is point to newgen pauses in other JVMs and say: 'those too will be gone'."

> *...Furthermore, the fact that it can handle multi-GB-per-sec allocations without worsening latencies or pauses, makes it very appealing for developers who have been trying hard not to allocate things because "it hurts". With Zing, you can use plenty of memory to go fast, instead of trying to avoid using it so that things won't hurt and jitter.*

For production use Zing is priced on an annual subscription/server. Unsurprisingly the vendor is reluctant to reveal pricing information, though it is in line with a supported Oracle or IBM JVM.

Zing is closed source, but is made available for free to open-source developers, so you can try it out whilst developing code.

About the author

Charles Humble is the CTO for **PRPi Consulting**, recently acquired by PwC. He has overall responsibility for the development of all the custom software used within the company. He has worked in enterprise software for around 20 years as a developer, architect and development manager. He co-founded **Conissaunce**, a UK based enterprise computing consultancy focused on the retail and !nancial services industries, and remains a director of the !rm. He spends as much time as he can with his young family, and writes music with **twofish**.

About the article

Reprinted with permission from C4 Media Inc and InfoQ.com.
C4Media Inc. (InfoQ.com),
2275 Lake Shore Boulevard West,
Suite #509,
Toronto, Ontario, Canada,
M8V 3Y3

This article was !rst published on InfoQ.com here:
http://www.infoq.com/news/2013/05/lmax-zing