Performance Results

mikeb01 edited this page on Sep 26 2012 · 6 revisions

Pages 13

Clone this wiki locally

https://github.com/LMAX-Exc

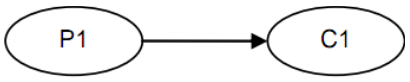
Clone in Desktop

The results of latency and throughput testing against a queue implementation

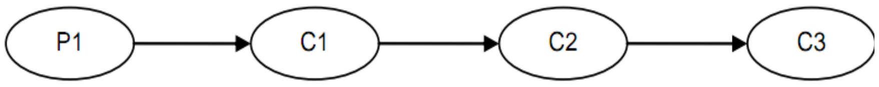
Performance Results (Disruptor 1.x)

We tested the performance of the disruptor in a number of graph configurations and measured the latency and throughput against a queue-based implementation.

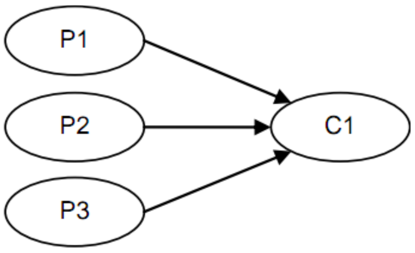
Configurations



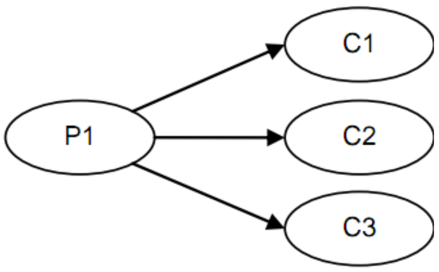
Unicast: 1P – 1C



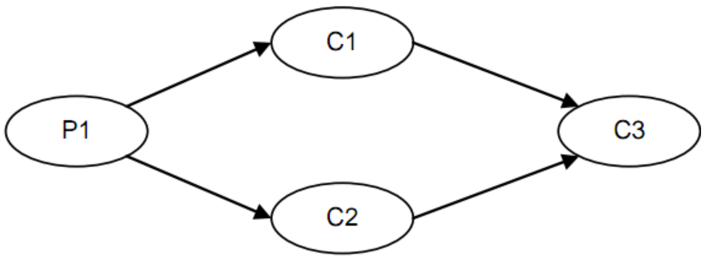
Three Step Pipeline: 1P – 3C



Sequencer: 3P – 1C



Multicast: 1P – 3C



Diamond: 1P – 3C

Throughput Performance Testing

As a reference we choose Doug Lea's excellent java.util.concurrent.ArrayBlockingQueue which has the highest performance of any bounded queue based on our testing. The tests are conducted in a blocking programming style to match that of the Disruptor. The tests cases detailed below are available in the Disruptor open source project.

Note: running the tests requires a system capable of executing at **least 4 threads** in parallel.

Nehalem 2.8Ghz – Windows 7 SP1 64-bit

Comparative throughput (in ops per sec)

	Array Blocking Queue	Disruptor
Unicast: 1P – 1C	5,339,256	25,998,336
Pipeline: 1P – 3C	2,128,918	16,806,157
Sequencer: 3P – 1C	5,539,531	13,403,268
Multicast: 1P – 3C	1,077,384	9,377,871
Diamond: 1P – 3C	2,113,941	16,143,613

Sandy Bridge 2.2Ghz – Linux 2.6.38 64-bit

Comparative throughput (in ops per sec)

	Array Blocking Queue	Disruptor
Unicast: 1P – 1C	4,057,453	22,381,378
Pipeline: 1P – 3C	2,006,903	15,857,913
Sequencer: 3P – 1C	2,056,118	14,540,519
Multicast: 1P – 3C	260,733	10,860,121
Diamond: 1P – 3C	2,082,725	15,295,197

Latency Performance Testing

To measure latency we take the three stage pipeline and generate events at less than saturation. This is achieved by waiting 1 microsecond after injecting an event before injecting the next and repeating 50 million times. To time at this level of precision it is necessary to use time stamp counters from the CPU. We choose CPUs with an invariant TSC because older processors suffer from changing frequency due to power saving and sleep states. Intel Nehalem and later processors use an invariant TSC which can be accessed by the latest Oracle JVMs running on Ubuntu 11.04. No CPU binding has been employed for this test. For comparison we use the `ArrayBlockingQueue` once again.

We could have used ConcurrentLinkedQueue which is likely to give better results but we want to use a bounded queue implementation to ensure producers do not outpace consumers by creating back pressure.

The results below are for 2.2Ghz Core i7-2720QM running Java 1.6.0_25 64-bit on Ubuntu 11.04.

	Array Blocking Queue (ns)	Disruptor (ns)
Mean Latency	32,757	52
99% observations less than	2,097,152	128
99.99% observations less than	4,194,304	8,192

Mean latency per hop for the Disruptor comes out at 52 nanoseconds compared to 32,757 nanoseconds for `ArrayBlockingQueue`. Profiling shows the use of locks and signalling via a condition variable are the main cause of latency for the `ArrayBlockingQueue`.

Note: It is worth playing with JVM configuration in conjunction with different settings for Hyper Threading, C States, SMI Interrupts and binding the JVM threads to specific core groups away from system interrupts. These changes can make a significant difference to latency results.