The Use of Java in Online Event Building and Recording at Jefferson Lab

Why JAVA?
- Highly portable
- Short development time
- Easily comprehensible
- Easily maintainable
- Fairly fast I/O
- Event building requires little compute intensive code
- Excellent IDEs and profiling tools available at no cost

Don’t use queues
- Either full or empty. When empty, producers and consumers contend. When full, all consumers contend. All operations contend over setting size.
- Entries must be allocated, inserted, removed, and garbage collected
- Head, tail, and size often on same cache-line
- Backing by linked-list is slow and garbage generating.
- Under heavy load, queues may fill causing application to slow so entries live longer and are copied to older generation space. Eventual collection from old generation is expensive and may lead to “stop-the-world” pauses to unfragment memory.

<table>
<thead>
<tr>
<th>Array Blocking Queue (ns)</th>
<th>Disruptor (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min Latency</td>
<td>145</td>
</tr>
<tr>
<td>Mean Latency</td>
<td>33,000</td>
</tr>
<tr>
<td>99% less than</td>
<td>2,100,000</td>
</tr>
<tr>
<td>99.99% less than</td>
<td>4,200,000</td>
</tr>
<tr>
<td>Max Latency</td>
<td>5,100,000</td>
</tr>
</tbody>
</table>

Don’t use locks

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single thread</td>
<td>300</td>
</tr>
<tr>
<td>Single thread with volatile write</td>
<td>4,700</td>
</tr>
<tr>
<td>Single thread with CAS</td>
<td>5,700</td>
</tr>
<tr>
<td>Two threads with CAS</td>
<td>30,000</td>
</tr>
<tr>
<td>Single thread with lock</td>
<td>10,000</td>
</tr>
<tr>
<td>Two threads with lock</td>
<td>224,000</td>
</tr>
</tbody>
</table>

DAQ layout

Good programming techniques
- Avoid locks
- Minimize creation of objects
- Reuse objects
- Avoid queues like the plague
- Replace queues with Disruptor’s ring buffers.

Use Disruptor’s ultra-fast ring buffer
- Ring entries allocated once as array and are containers which are never added or removed. Arrays work well with caching.
- For single producer, access has NO locks, CAS or contention. Uses only memory barriers.
- Written to eliminate false sharing (cache friendly)
- Operates in a “batch” mode, in which consumer asks for next entry, but gets access to all available entries without any more concurrency operations.

Ring buffer

EB Performance
- Simulating a DC: 12 ROCs at 100 MB/s (1.2 GB/s) to 1 EB is easily handled with 4 cores.
- Simulating an SEB: 5 ROCs at 350 MB/s each (1.75 GB/s) to 1 EB is handled with 6 cores.
- Old Queues: > 40% of cpu time
- New Ring buffers: 0.6% of cpu time

Consumer waiting strategies
- 1) spin, 2) spin then yield, 3) spin, yield, then sleep, 4) timeout, 5) block and spin on waking, 6) spin, block, then spin on waking (spin-block)
- Each ring created with 1 strategy
- Different strategies can greatly affect performance
- For simulation with 11 ROCs at 32 MB/s each to 1 DC, spin-yield used 15.8 cores, spin-block used 2.7 cores and performed better!

References
http://www-exchange.github.io/disruptor