Disruptor
Using High Performance, Low Latency Technology in the CERN Control System

ICALEPCS 2015
The problem at hand
The problem at hand

- CESAR is used to control the devices in CERN experimental areas
The problem at hand

- CESAR is used to control the devices in CERN experimental areas
- These devices produce 2500 event streams
The problem at hand

• CESAR is used to control the devices in CERN experimental areas

• These devices produce 2500 event streams

• The business logic on the CESAR server combines the data coming from these streams to calculate device states
The problem at hand

- CESAR is used to control the devices in CERN experimental areas
- These devices produce 2500 event streams
- The business logic on the CESAR server combines the data coming from these streams to calculate device states
- This concurrent processing must be properly synchronized
What happens when all flows converge?
1- Some background about the Disruptor
1- Some background about the Disruptor

- Created by LMAX, a trading company, to build a high performance Forex exchange
1- Some background about the Disruptor

• Created by LMAX, a trading company, to build a high performance Forex exchange

• Is the result of different trials and errors
1- Some background about the Disruptor

- Created by LMAX, a trading company, to build a high performance Forex exchange
- Is the result of different trials and errors
- Challenges the idea that “CPUs are not getting any faster”
1- Some background about the Disruptor

- Created by LMAX, a trading company, to build a high performance Forex exchange
- Is the result of different trials and errors
- Challenges the idea that “CPUs are not getting any faster”
- Designed to take advantage of the architecture of modern CPUs, following the concept of “mechanical sympathy”
Feed the cores – avoid cache misses

<table>
<thead>
<tr>
<th>Socket</th>
<th>Core 1</th>
<th>Core 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L1 Cache</td>
<td>L1 Cache</td>
</tr>
<tr>
<td></td>
<td>L2 Cache</td>
<td>L2 Cache</td>
</tr>
<tr>
<td>L3 Cache</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Socket Interconnect

RAM

Socket Interconnect
Feed the cores – avoid cache misses

- Core 1
  - L1 Cache: 64 KB, 1 ns
  - L2 Cache
  - L3 Cache

- Core 2
  - L1 Cache
  - L2 Cache
  - L3 Cache: 1 to 20 MB

Socket Interconnect

RAM
Feed the cores – avoid cache misses

- Core 1
  - L1 Cache
  - L2 Cache
- Core 2
  - L1 Cache
  - L2 Cache
  - L3 Cache

Socket Interconnect

- RAM

- Socket
  - L1 Cache: 64 KB (1 ns)
  - L2 Cache: 256 KB (3 ns)
  - L3 Cache: 1 to 20 MB
Feed the cores – avoid cache misses

<table>
<thead>
<tr>
<th>Core 1</th>
<th>Core 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Cache</td>
<td>L1 Cache</td>
</tr>
<tr>
<td></td>
<td>64 KB</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>L2 Cache</td>
</tr>
<tr>
<td></td>
<td>256 KB</td>
</tr>
<tr>
<td>L3 Cache</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 to 20 MB</td>
</tr>
</tbody>
</table>

Socket Interconnect

RAM
Feed the cores – avoid cache misses

- Core 1:
  - L1 Cache: 64 KB
  - L2 Cache
  - L3 Cache

- Core 2:
  - L1 Cache: 256 KB
  - L2 Cache

Socket:
- Socket Interconnect

RAM:
- 1 to 20 MB

Timing:
- Socket Interconnect: 65 ns
- L1 cache: 1 ns
- L2 cache: 3 ns
- L3 cache: 12 ns
Avoiding false sharing

1 cache line = 64 bytes
(on modern x86)

Core 1

L1 / L2

X Y

Core 2

L1 / L2

X Y

1 to 3 ns

L3

12 ns

Socket Interconnect

40 ns
Avoiding false sharing

1 cache line = 64 bytes
(on modern x86)

Core 1

L1 / L2

X Y

L3

X Y

1 to 3 ns

12 ns

Socket Interconnect

40 ns
Avoiding false sharing

1 cache line = 64 bytes
(on modern x86)

Socket Interconnect

L3

12 ns

L2

Core 2

X

Y

Core 1

L1

X'

Y

1 to 3 ns

40 ns
Avoiding false sharing

1 cache line = 64 bytes
(on modern x86)

1 to 3 ns

12 ns

40 ns

Socket Interconnect

Core 1

Core 2

L1 / L2

L3

X' Y

X Y

X' Y
Avoiding false sharing

1 cache line = 64 bytes
(on modern x86)

L1 / L2

Core 1

L3

Core 2

Socket Interconnect

1 to 3 ns

12 ns

40 ns
Avoiding false sharing

1 cache line = 64 bytes (on modern x86)

Core 1

L1 / L2

Core 2

L3

Socket Interconnect

X' Y

X Y

X' Y

1 to 3 ns

12 ns

40 ns
Avoiding false sharing

1 cache line = 64 bytes (on modern x86)

L1 / L2

Core 1

L3

Core 2

Socket Interconnect

X' Y

X' Y

X Y

1 to 3 ns

12 ns

40 ns
Avoiding false sharing

1 cache line = 64 bytes
(on modern x86)

The solution?
Avoiding false sharing

1 cache line = 64 bytes (on modern x86)

Core 1

L1 / L2

Core 2

P P X P P P

PP YPPP

L3

Socket Interconnect

1 to 3 ns

12 ns

40 ns

The solution? Padding
• What is it?

→ Can be viewed as a very efficient FIFO bounded queue

→ A data structure to pass data between threads, designed to avoid contention
The mighty ring buffer
The mighty ring buffer
The mighty ring buffer

- Represented internally as an array → caches get prefetched
The mighty ring buffer

• Represented internally as an array → caches gets prefetched
• The sequence number is a padded long → no false sharing
The mighty ring buffer

- Represented internally as an array → caches get pre-fetched
- The sequence number is a padded long → no false sharing
- The memory visibility relies on the volatile sequence number → no locks
The mighty ring buffer

- Represented internally as an array → caches get prefetched
- The sequence number is a padded long → no false sharing
- The memory visibility relies on the volatile sequence number → no locks
- Slots are preallocated → no garbage collection
Main differences compared to a queue
Main differences compared to a queue

→ Latency and jitter reduced to a minimum
Main differences compared to a queue

→ Latency and jitter reduced to a minimum

→ No garbage collection
Main differences compared to a queue

- Latency and jitter reduced to a minimum
- No garbage collection
- Can have multiple consumers organized in a graph of dependency
Main differences compared to a queue

→ Latency and jitter reduced to a minimum

→ No garbage collection

→ Can have multiple consumers organized in a graph of dependency

→ Consumers can use batching to catch up with producers
Benefits for the architecture
Benefits for the architecture

→ Performance
   No locks, no garbage collection, CPU friendly
Benefits for the architecture

→ Performance
   No locks, no garbage collection, CPU friendly

→ Determinism
   The order in which events were processed is known
   Messages can be replayed to rebuild the server state
Benefits for the architecture

→ **Performance**
   No locks, no garbage collection, CPU friendly

→ **Determinism**
   The order in which events were processed is known
   Messages can be replayed to rebuild the server state

→ **Simplification of the code base**
   Since the business logic runs on a single thread, there is no need to worry about concurrency
3 – The Disruptor in CESAR
Each event received from hardware is stored on the ring buffer
3 – The Disruptor in CESAR

- Each event received from hardware is stored on the ring buffer
- For each stream of data, the last value is kept
3 – The Disruptor in CESAR

- Each event received from hardware is stored on the ring buffer

- For each stream of data, the last value is kept

- We make use of batching
3 – The Disruptor in CESAR

- Each event received from hardware is stored on the ring buffer
- For each stream of data, the last value is kept
- We make use of batching
- At the end of a batch, the business logic is triggered and executed on a single thread
3 – The Disruptor in CESAR

→ Each event received from hardware is stored on the ring buffer

→ For each stream of data, the last value is kept

→ We make use of batching

→ At the end of a batch, the business logic is triggered and executed on a single thread

→ Publish the new states over the network, making sure that we do not block the Disruptor thread if the message broker is down
Conclusions

• The Disruptor, a tool from the world of finance, fits really well in an Accelerator control system

• It simplified the CERN CESAR code base while handling the flow of data more efficiently

• It is easily integrated in an existing design to replace a queue or a full pipeline of queues

• The main challenge faced was to switch the developers’ mind-set to think in asynchronous terms
Useful Links

• The Disruptor main page with an introduction and code samples:
  http://lmax-exchange.github.io/disruptor

• Presentation of the Disruptor at Qcon
  http://www.infoq.com/presentations/LMAX

• An article from Martin Fowler:
  http://martinfowler.com/articles/lmax.html

• A useful presentation on Latency by Gil Tene who shows that most of what we measure during performance test is wrong:
  http://www.infoq.com/presentations/latency-pitfalls

• New Async logger in Log4J 2
  http://logging.apache.org/log4j/2.x/manual/async.html