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

![Diagram showing CESAR system](attachment://CESAR_diagram.png)
What happens when all flows converge?
THE MAGIC ROUNDABOUT

Ring road
Cirencester
A4289

Town centre

Marlborough
Burford
Oxford

(M4)

A4312

H

A & E
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

Core 1
- L1 Cache (64 KB)
- L2 Cache

Core 2
- L1 Cache (256 KB)
- L2 Cache

L3 Cache (1 to 20 MB)

Socket Interconnect

RAM
Feed the cores – avoid cache misses

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

- Core 2
  - L1 Cache
  - L2 Cache

L1 Cache: 64 KB, 1 ns
L2 Cache: 256 KB
L3 Cache: 1 to 20 MB

Socket Interconnect

RAM
Feed the cores – avoid cache misses

Socket Interconnect

Core 1

L1 Cache

L2 Cache

L3 Cache

64 KB

256 KB

1 ns

3 ns

Core 2

L1 Cache

L2 Cache

1 to 20 MB

Socket

RAM
Feed the cores – avoid cache misses

Socket

Core 1
L1 Cache
L2 Cache
L3 Cache

Core 2
L1 Cache
L2 Cache
L3 Cache

Socket Interconnect

RAM

64 KB
256 KB
1 to 20 MB

1 ns
3 ns
12 ns
Feed the cores – avoid cache misses

- **Core 1**
  - L1 Cache
  - L2 Cache

- **Core 2**
  - L1 Cache
  - L2 Cache

- **L3 Cache**
  - 1 to 20 MB
  - 12 ns

- **Socket Interconnect**
  - RAM
  - 65 ns

- **Socket**
  - 64 KB
  - 1 ns
  - 256 KB
  - 3 ns
  - 1 to 20 MB
  - 12 ns
Avoiding false sharing

1 cache line = 64 bytes (on modern x86)

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

1 cache line = 64 bytes

1 to 3 ns

12 ns

40 ns
Avoiding false sharing

1 cache line = 64 bytes
(on modern x86)

Core 1

X'
Y

Core 2

X
Y

L1 / L2

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

- Core 2
  - X
  - Y

Socket Interconnect

- 1 to 3 ns
- 12 ns
- 40 ns
Avoiding false sharing

1 cache line = 64 bytes
(on modern x86)

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

X' Y

X' Y

X Y

1 to 3 ns

12 ns

Socket Interconnect

40 ns
Avoiding false sharing

1 cache line = 64 bytes (on modern x86)

Core 1

L1 / L2

X' Y

Core 2

L1 / L2

X Y

L3

X' Y

Socket Interconnect

1 to 3 ns

12 ns

40 ns
Avoiding false sharing

Core 1

L1 / L2

Core 2

X

Y

1 cache line = 64 bytes
(on modern x86)

1 to 3 ns

12 ns

Socket Interconnect

40 ns

The solution?
Avoiding false sharing

1 cache line = 64 bytes
(on modern x86)

The solution? Padding
2 - Disruptor architecture

- 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

Producer

Consumer
The mighty ring buffer

• Represented internally as an array → caches gets prefetched
The mighty ring buffer

- Represented internally as an array → caches get prefetched
- The sequence number is a padded long → no false sharing
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
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
→ 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
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