Exo: Atomic Broadcast for the Rack-Scale Computer

Matthew P. Grosvenor  Marwan Fayed Andrew W. Moore
What is a Rack-Scale Computer

Today

• 50 - 200 machines

• Commodity hardware

• Commodity network

Tomorrow

• 500-2000 “nodes”

• Disaggregated hardware

• Custom (photonic?) interconnect(s)
The problem with disaggregation
Other Types of Coordination

Single Machines

• Coordination using simple MESI/MOSI protocols

• Specialised hardware over a reliable, low latency interconnect.

✓ High performance

✗ Not failure tolerant

✗ Doesn’t scale well
Other Types of Coordination

Single Machines

- Coordination using simple MESI/MOSI protocols
- Specialised hardware over a reliable, low latency interconnect.

✓ High performance

✗ Not failure tolerant

✗ Doesn’t scale well

Cluster Systems

- Coordination using software protocols
- General purpose, high latency, error prone network

✗ Low performance

✓ Fault Tolerant

✗ Doesn’t scale well
Other Types of Coordination

**Single Machines**
- Coordination using simple MESI/MOSI protocols
- Specialised hardware over a reliable, low latency interconnect.

- **✓** High performance
- **✗** Not failure tolerant
- **✗** Doesn’t scale well

**Cluster Systems**
- Coordination using software protocols
- General purpose, high latency, error prone networks

- **✗** Low performance
- **✓** Fault Tolerant
- **✗** Doesn’t scale well

**Something in the middle ??**
Other Types of Coordination

Single Machines

- Coordination using simple MESI/MOSI protocols
- Specialised hardware over a reliable, low latency interconnect

✓ High performance
✓ Fault Tolerant
x Not failure tolerant
x Doesn’t scale well

Cluster Systems

- Coordination using software protocols
- General purpose, high latency, error prone network

x Low performance
✓ Doesn’t scale well

That scales?

Something in the middle ???
✓ Silicon Photonic Interfaces on chip

✗ High radix optical switches?

✓ Burst mode transceivers

✓ Passive all optical interconnect (PON)

✓ Network broadcast as a primitive for building Atomic Broadcast
Media Access and Agreement

- How do we mediate access to the network?
- What agreement protocol do we use?

- Use token rings

  - Two birds with one stone
    - Well established MAC protocol
    - Well established agreement protocol. Very high (optimal) throughput performance.
The Problem with Token Passing

- Token passing is latency bound
- Each host must wait at least n-1 hops for agreement = n x per-hop latency (t)
- Per hop latency must be minimised

Total latency = t + t + t + t = 4t

Throughput (per host) ≈ 1/4t
What’s the latency?

Assuming 3m of fibre @ 0.75C

7.5ns  0ns  0ns  7.5ns  = 15ns per hop

Unicast  Broadcast
What's the latency?

Assuming 3m of fibre @ 0.75C

7.5ns  0ns  0ns  7.5ns
Unicast  Broadcast

= 15ns per hop

@10G 64B frames
= 14.4M msg/sec
Building a Coordination Network (Today)

Use Exalink Fusion Switch to emulate all optical network

- Broadcast - 5ns
- Aggregation - 95ns
Building a Coordination Network (Today)

7.5ns  95ns  5ns  7.5ns

Unicast  Broadcast

A
B
C
D

Fusion Aggregate

Fusion Broadcast

= 115ns per hop

= 8.6M msg/sec
• Use ethernet switch and matrix switch to emulate Exablaze Fusion

• Arista 7124FX
  • Aggregation/switching 350ns

• Exablaze ExaLink 50
  • Broadcast <5ns
Building a Coordination Network (in the lab)

7.5ns  350ns  5ns  7.5ns

= 370ns per hop
= 2.7M msg/sec

Artist 7124FX        Exablaxe ExaLink50
But wait, there’s more… (latency)

Exo Server

NIC

Unicast

Broadcast
But wait, there’s more… (latency)

Network Latency

370ns
But wait, there’s more (latency)

Network Latency  PCIe Latency

370ns
But wait, there’s more… (latency)

Network Latency: 370ns

PCle Latency: >800ns

Unicast

Broadcast
But wait, there’s more… (latency)

- total per-hop-latency ≈ 1us
- 200 x 1us ≈ 200us
  ≈ 5000 msgs per host per second.
1. Make it go away
   - e.g. use a Fusion
   - e.g. use all optical

2. Hide it
   - Introduce pipelining
Dealing with latency

1. Make it go away
   - e.g. use a Fusion
   - e.g. use all optical

2. Hide it
   - Introduce pipelining
Hiding Latency

- Introduce pipelining to hide latency
- Host to NIC and NIC to network transmissions run in parallel
Hiding Latency

- In network latency ≈ 400ns
- Per-hop latency ≈ 1us (pipelined for 1 message per 400ns)
- 200 x 400ns ≈ 80us
  ≈ 12,500 messages per second / per host.
Let’s Talk About Failure

- Failure is very unlikely.

- No packet loss due to congestion

- Bit errors on the wire $1/10^{12}$ … $1/10^{14}$

- Partitioning - Single chip? Can it half fail? Byzantine?

  - $1/10^{20}$ ??

- Partitioning - requires n-way x m - correlated failure

  - $(1/10^{12})^n \times m$

- Therefore, optimise for the common case
The Exo Protocol

- Protocol is implemented correctly on all hosts
- Hosts may stop, crash, restart, loose packets or become partitioned
- There is a fixed upper number of hosts (n) in the network. Host may leave and come back, but more hosts may not be added.
Assume a network with fast and cheap broadcast messaging
And some maximum number of hosts $n$ (5) labeled A-E
And some maximum number of hosts $n$ (5) labeled A-E, in a fixed, predetermined order.
Each host keeps the following state:
- append only log
- sequence number
- $n/2$ message history
Host B broadcasts a message M with the sequence number and message history included as well as any data (optional).
Message from B arrives at all hosts, causes a: -log append
Message from B arrives at all hosts, causes:
- log append
- sequence number update

\[ M = 0, X, X \]
Message from B arrives at all hosts, causes a:
- log append
- sequence number update
- message history update
Host C now has the network
Host C broadcasts a message

\[ M=1,B,X \]
Message from C arrives at hosts, causing update

\[ M=1, B, X \]
D now has the token
Host \( D \) broadcasts a message

\[ M = 2, C, B \]
Message from D arrives at hosts, causing *update and agreement on sequence number 0*
Message from D arrives at hosts, causing *update and agreement on sequence number 0*
Zooming in...

```
B  C  D
0  1  2
x  B  C
x  x  B
```

```
seq : 3
D   C
```
NB: Using a 64bit sequence number gives >400 years operation at 1 message per nano-second.
seq : 3

D  C

Agreement

Append

B  C  D
<table>
<thead>
<tr>
<th>seq : 3</th>
<th></th>
<th>D</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A majority of hosts agree on the message order

seq: 3

NB: Assuming at most 256 nodes, message history is 128B and can
How to accelerate the protocol?

<table>
<thead>
<tr>
<th>Host</th>
<th>NIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq : 3</td>
<td>seq : 3</td>
</tr>
<tr>
<td>0</td>
<td>D</td>
</tr>
<tr>
<td>x</td>
<td>C</td>
</tr>
<tr>
<td>x</td>
<td>D</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>x</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
</tr>
<tr>
<td>x</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>B</td>
</tr>
</tbody>
</table>

x

x

x

D

C

D

C
Implementation

• Protocol implemented over CamIO
  • Runs on TCP, Broadcast UDP, Raw Frames and OpenOnload (ExaSock)
• Offload engine (mostly) implemented in FPGA
  • Currently working for empty messages.
<table>
<thead>
<tr>
<th>Protocol Type</th>
<th>Latency</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux TCP</td>
<td>36µs/msg</td>
<td>28K msg/sec</td>
</tr>
<tr>
<td>Linux UDP</td>
<td>29µs/msg</td>
<td>(34K msg/sec)</td>
</tr>
<tr>
<td>Linux UDP (broadcast)</td>
<td>16µs/msg</td>
<td>(62K msg/sec)</td>
</tr>
<tr>
<td>Linux TCP (loopback)</td>
<td>9µs/msg</td>
<td>(110K msg/sec)</td>
</tr>
<tr>
<td>Linux UDP (loopback)</td>
<td>4µs/msg</td>
<td>(231K msg/sec)</td>
</tr>
<tr>
<td>Libexanic (raw frames)</td>
<td>3µs/msg</td>
<td>(359K msg/sec)</td>
</tr>
<tr>
<td>Exo HW offload</td>
<td>0.4µs/msg</td>
<td>(2500K msg/sec)</td>
</tr>
<tr>
<td></td>
<td>Time per Message</td>
<td>Throughput</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Linux TCP</td>
<td>36µs/msg</td>
<td>28K msg/sec</td>
</tr>
<tr>
<td>Linux UDP</td>
<td>29µs/msg</td>
<td>(34K msg/sec)</td>
</tr>
<tr>
<td>Linux UDP (broadcast)</td>
<td>16µs/msg</td>
<td>(62K msg/sec)</td>
</tr>
<tr>
<td>Linux TCP (loopback)</td>
<td>9µs/msg</td>
<td>(110K msg/sec)</td>
</tr>
<tr>
<td>Linux UDP (loopback)</td>
<td>4us/msg</td>
<td>(231K msg/sec)</td>
</tr>
<tr>
<td>Libexanic (raw frames)</td>
<td>3µs/msg</td>
<td>(359K msg/sec)</td>
</tr>
<tr>
<td>Exo HW offload</td>
<td>0.4µs/msg</td>
<td>(2500K msg/sec)</td>
</tr>
</tbody>
</table>

\( \approx 100x \text{ Faster} \)
Conclusions

• Exo is a work in progress

• We use a specialised network and hardware offload to build fast scalable coordination for rack-scale architectures

• Initial results suggest at least a 100x speed improvement over existing protocols/systems
This work was jointly supported by the EPSRC INTERNET Project EP/H040536/1 and the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-11-C-0249. The views, opinions, and/or findings contained in this article/presentation are those of the author/presenter and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense.
Backups
E now has the token

Anatomy of the Exo Protocol

E

A

B

C

D

E

D

C

B

seq : 3

D

C

Append

Agreement

0

x

x

1

B

x

2

C

B

55
E now has the token
Consider the following pathological case.
Consider the following pathological case

Network partition

Handling Errors?

seq : 3

D  C

Agreement

seq : 3

D  C

Append

0 1 2

B  C  D

x  B  x

E  B  A

x  C  B

A  C  D

Append

Acknowledgments

Consider the following pathological case.
Handling Errors?

Host $E$ broadcasts a message $M=3,D,C,B$.
Message from $E$ arrives at $C,D$ & $E$ causing an update and agreement on sequence 1.
Handling Errors?

Host A broadcasts a message

M = 3, D, C, B

Append

Agreement

seq : 3

D C

seq : 4

E D
Message from A arrives at A & B, causing an **update** and **agreement**
Handling Errors?

B now has the token

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>x</td>
<td>B</td>
<td>C</td>
<td>3</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>D</td>
<td>C</td>
</tr>
</tbody>
</table>

Append: seq: 4

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
<th>D</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>x</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

Agreement

seq: 4

E D
Handling Errors?

Host B broadcasts a message

\[ M=4, A, D, C \]

<table>
<thead>
<tr>
<th>seq : 4</th>
<th>E</th>
<th>D</th>
</tr>
</thead>
</table>

\[ 0 \quad x \quad B \quad C \quad D \quad x \]

\[ 1 \quad B \quad C \quad D \quad A \]

Agreement
Message from B arrives at A & B, causing an update and agreement.

Sequence:

- seq: 4, E, D
- seq: 5, B, A

Table:

<table>
<thead>
<tr>
<th>seq</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>2</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>D</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

Message from B arrives at A & B, causing an update and agreement.
After some time, A times out and assumes it has the token.
Message from A is sent. Since there is no quorum, the message is rejected.
Message from A is sent. Since there is no quorum, the message is rejected.

Exo will only accept a new message if the source of the message is not in the history list. Nodes in a minor partition are halted until the partition restores.
In the meantime, Host C broadcasts a message $M=4,E,D,C$. However, Host A, which has received $M=4,E,D,C$ from Host C, cannot agree with Host B and Host D, who have received $M=0,B,C$ and $M=1,B,C$, respectively. Therefore, Host A does not join the agreement and creates a new message $M=5,E,A,D$. Host B and Host D, who have received $M=5,E,A,D$, now agree with each other, and the agreement is achieved.
Handling Errors?

Message from B arrives at A & B, causing an **update** and **agreement**
D now has the token
Miraculously, the partition is healed!!!
Miraculously, the partition is healed!!!
Handling Errors?

Host D broadcasts a message

$M=5, C, E$

seq : 5

B C D E C

seq : 5

C E
Message from D arrives at A, B, C, D & E causing ???

seq : 5

B C D E A
0 x 1 B x 2 C x 3 D C 4 A D

seq : 5

C E
Message from $D$ arrives at $A$, $B,C,D$ & $E$ causing ???

```
<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>1</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>2</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>3</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>E</td>
<td></td>
</tr>
</tbody>
</table>
```

Sequence: 5

```
<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>1</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>2</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>3</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>
```

Sequence: 5
Handling Errors?

Message from $D$ arrives at $A$, $B,C,D$ & $E$ causing ???

seq : 6  

seq : 5  

Append

Agreement

Append

Agreement
Handling Errors?

Message from D arrives at A, B, C, D & E causing ???

seq : 6

Append

seq : 5

Append
Message from $D$ arrives at $A$, $B,C,D$ & $E$ causing
-update and agreement to $C$, $D$ & $E$
-error detection for $A$ & $B$

seq : 6

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>1</td>
<td>B</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>B</td>
<td>x</td>
<td>C</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>C</td>
<td>E</td>
<td>D</td>
<td>5</td>
</tr>
</tbody>
</table>

seq : 5

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
<th>D</th>
<th>A</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>1</td>
<td>B</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>B</td>
<td>x</td>
<td>C</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>C</td>
<td>E</td>
<td>D</td>
<td>5</td>
</tr>
</tbody>
</table>

Append
Handling Errors?

Hosts A & B must now use the out of band network to query C, D & E about their log state for sequence 3-5. Once a majority of response are gathered is reached, host A & B can continue to move the agreement pointer.

seq : 6

seq : 5
Hosts A & B must now use the out of band network to query C, D & E about their log state for sequence 3-5. Once a majority of responses are gathered is reached, host A & B can continue to move the agreement pointer.

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>x</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
</tbody>
</table>

Sequence: 6

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>A</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>x</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>A</td>
</tr>
</tbody>
</table>

Sequence: 5

NB: New messages can be added to the log at the append pointer, but the agreement pointer cannot move until consensus has been reached.