Outline

1. Motivation

2. Systems
   - Ricochet
   - Maelstrom

3. Conclusion
Commodity Datacenters

- Blade-servers, Fast Interconnects
- Different Apps:
  - Google -> Search
  - Amazon -> Etailing
  - Computational Finance, Aerospace, Military C&C, e-Science...
  - ... YouTube?
The Datacenter Paradigm

Motivation
Systems
Conclusion

Extreme Scale-out
- More Nodes, More Capacity
- Services distributed / replicated / partitioned over multiple nodes
Building Datacenter Apps

- High-level Abstractions:
  - Publish/Subscribe
  - Event Notification
  - Replication (Data/Functionality)
  - Caching
- BEA Weblogic, JBoss, IBM Websphere, Tibco, RTI DDS, Tangosol, Gemfire...
- What’s under the hood?
  Multicast!
Properties of a Multicast Primitive

Rapid Delivery
- ... when failures occur (reliable)
- ... at extreme scales (scalable)

Questions:
- What technology do current systems use?
- Is it truly ‘reliable’ and ‘scalable’?
- Can we do better?
A Brief History of Multicast

- Two Divergent Directions:
  - Overlay Multicast *instead of* IP Multicast (e.g, BitTorrent)
  - Reliable Multicast *over* IP Multicast (e.g, TIBCO)
- Datacenters have IP Multicast support...
Many different reliable, scalable protocols
Designed for streaming video/TV, file distribution
Reliable:
  - Packet Loss at WAN routers
Scalable:
  - Single group with massive numbers of receivers
Not suited for datacenter multicast!
  - Different failure mode
  - Different scalability dimensions
Motivation
Systems
Conclusion

(Reliable) Multicast in the Datacenter

Packet Loss occurs at end-hosts: independent and bursty

(receiver r₁: loss bursts in A and B)  (receiver r₁: data bursts in A and B)
(receiver r₂: loss bursts in A)  (receiver r₂: data bursts in A)

Mahesh Balakrishnan  CS 514: Transport Protocols for Datacenters
Financial Datacenter Example:

- Each equity is mapped to a multicast group.
- Each Node is interested in a different set of equities...
- ... each Node joins a different set of groups.

Lots of overlapping groups $\implies$ Low per-group data rate.
Designing a Time-Critical Multicast Primitive

- **Wanted:** A *reliable*, *scalable* multicast protocol.
  - Reliable:
    - can tolerate end-host loss bursts
  - Scalable:
    - the size of the group
    - the number of senders to a group
    - the number of groups per node
Wanted: A reliable, scalable multicast protocol.

Reliable:
- can tolerate end-host loss bursts

Scalable:
- the size of the group
- the number of senders to a group
- the number of groups per node
Wanted: A *reliable*, *scalable* multicast protocol.

**Reliable:**
- can tolerate end-host loss bursts

**Scalable:**
- the size of the group
- the number of senders to a group
- the number of groups per node
Outline

1. Motivation

2. Systems
   - Ricochet
   - Maelstrom

3. Conclusion
Design Space for Reliable Multicast

How does latency scale?

Two Phases: *Discovery* and *Recovery* of Lost Packets

- ACK/timeout: RMTP/RMTP-II
- Gossip-based: Bimodal Multicast, Ipbcast
- NAK/sender-based sequencing: SRM
- Forward Error Correction

Fundamental Insight: \( \text{latency} \alpha \frac{1}{\text{datarate}} \)
NAK/Sender-Based Sequencing: SRM

Scalable Reliable Multicast - Developed 1998

- Loss discovered on next packet from same sender in same group
- $\text{latency} \propto \frac{1}{\text{datarate}}$
  
  data rate: at a single sender, in a single group
Pros:
- Tunable, Proactive Overhead
- *Time-Critical*: Eliminates need for retransmission

Cons:
- FEC packets are generated over a stream of data
  - Have to wait for \( r \) data packets before generating FEC
  - \( \text{latency} \propto \frac{1}{\text{datarate}} \)

\( \text{data rate: at a single sender, in a single group} \)
Randomness: Each Receiver picks another Receiver randomly to send XOR to

Tunability: Percentage of XOR packets to data is determined by rate-of-fire \((r, c)\)

\(\text{latency} \propto \frac{1}{\text{datarate}}\)

data rate: across all senders, in a single group
Randomness: Each Receiver picks another Receiver randomly to send XOR to

Tunability: Percentage of XOR packets to data is determined by rate-of-fire \((r, c)\)

\(\text{latency} \propto \frac{1}{\text{data rate}}\)

data rate: across all senders, in a single group
Randomness: Each Receiver picks another Receiver randomly to send XOR to

Tunability: Percentage of XOR packets to data is determined by rate-of-fire \((r, c)\)

\[\text{latency} \propto \frac{1}{\text{datarate}}\]

data rate: across all senders, in a single group
Randomness: Each Receiver picks another Receiver randomly to send XOR to

Tunability: Percentage of XOR packets to data is determined by rate-of-fire \((r, c)\)

\[\text{latency} \propto \frac{1}{\text{datarate}}\]  
data rate: across all senders, in a single group
Randomness: Each Receiver picks another Receiver randomly to send XOR to

Tunability: Percentage of XOR packets to data is determined by rate-of-fire \((r, c)\)

Latency \(\alpha \frac{1}{\text{datarate}}\)

data rate: across all senders, in a single group
Randomness: Each Receiver picks another Receiver randomly to send XOR to

Tunability: Percentage of XOR packets to data is determined by rate-of-fire \((r, c)\)

\[ \text{latency} \propto \frac{1}{\text{datarate}} \]

Data rate: across all senders, in a single group
Nodes $n_1$ and $n_2$ are both in groups A and B.
Nodes \( n_1 \) and \( n_2 \) are both in groups A and B.

**INCOMING DATA PACKETS**

- \( n_1 \):
  - A1
  - A2
  - A3
  - B1
  - B2
  - A4
  - A5
  - B3
  - B4
  - A6
  - B5

- \( n_2 \):
  - A1
  - A2
  - A3
  - B1
  - B2
  - A4
  - A5
  - B3
  - B4
  - A6
  - B5

**Repair Packet I:**
- (A1, A2, A3, A4, A5)

**Repair Packet II:**
- (B1, B2, B3, B4, B5)

**Repair Packet I:**
- (A1, A2, A3, B1, B2)

**Repair Packet II:**
- (A4, A5, B3, B4, A6)
Lateral Error Correction

Combine error traffic for multiple groups within intersections, while conserving:

- Coherent, tunable per-group overhead: Ratio of data packets to repair packets in the system is $r : c$
- Fairness: Each node receives on average the same ratio of repair packets to data packets

$\text{latency} \propto \frac{1}{\text{datarate}}$

Data rate: across all senders, in intersections of groups
Lateral Error Correction

Combine error traffic for multiple groups within intersections, while conserving:

- Coherent, tunable per-group overhead: Ratio of data packets to repair packets in the system is $r : c$
- Fairness: Each node receives on average the same ratio of repair packets to data packets
- $\text{latency} \propto \frac{1}{\text{data rate}}$
  
  data rate: across all senders, in intersections of groups
Lateral Error Correction: Mechanism

Divide overlapping groups into *regions*

- $n_1$ belongs to groups $A$, $B$, $C$:
- It divides them into regions $abc, ab, ac, bc, a, b, c$
- $n_1$ selects proportionally sized chunks of $c_A$ from the regions of $A$
- Total number of targets selected, across regions, is equal to the $c$ value of a group
Repair Bin Structure

- **Repair Bins:**
- **Input:** Data Packets in \textit{union} of Groups
- **Output:** Repair Packets to \textit{region}
- **Expectation:** Avg # of targets chosen from \textit{region}

<table>
<thead>
<tr>
<th>Venn Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>ab</td>
</tr>
<tr>
<td>abc</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>bc</td>
</tr>
<tr>
<td>ac</td>
</tr>
<tr>
<td>abc</td>
</tr>
</tbody>
</table>

At node $n_1$

- $|a|=5$
- $|ab|=2$
- $|abc|=10$
- $|b|=1$
- $|bc|=7$
- $|ac|=3$
- $|C|=5$
- $|C|=25$

<table>
<thead>
<tr>
<th>Venn Diagram Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
</tr>
<tr>
<td>$</td>
</tr>
<tr>
<td>$</td>
</tr>
</tbody>
</table>

Repair Bin Structure

- $ac$
- $a$
- $abc$
- $ab$
- $bc$
- $b$
- $c$

Regions

- $ac$
- $a$
- $abc$
- $ab$
- $bc$
- $b$
- $c$

- $0.36$
- $0.39$
- $1.25$
- $0.5$
- $1.2$
- $0.4$
- $0.84$
- $0.56$
- $0.2$
- $0.6$

- $0.1$
Experimental Evaluation

- Cornell Cluster: 64 1.3 Ghz nodes
- Java Implementation running on Linux 2.6.12
- Three Loss Models: {Uniform, Burst, Markov}
- Grouping Parameters: $g \ast s = d \ast n$
  - $g$: Number of Groups in System
  - $s$: Average Size of Group
  - $d$: Groups joined by each Node
  - $n$: Number of Nodes in System
- Each node joins $d$ randomly selected groups from $g$ groups
Distribution of Recovery Latency
16 Nodes, 128 groups per node, 10 nodes per group, Uniform 1% Loss

96.8% LEC + 3.2% NAK

92% LEC + 8% NAK

84% LEC + 16% NAK

(a) 10% Loss Rate  (b) 15% Loss Rate  (c) 20% Loss Rate
Scalability in Groups
64 nodes, * groups per node, 10 nodes per group, Loss Model: Uniform 1%

Ricochet scales to hundreds of groups. Comparison: at 128 groups, SRM latency was 8 seconds. **400 times slower!**
Ricochet is lightweight $\Rightarrow$ Time-Critical Apps can run over it
Impact of Loss Rate on LEC
64 nodes, 128 groups per node, 10 nodes per group, Loss Model: *

Effect of Loss Rate on Recovery %
- Bursty b=10
- Uniform
- Markov m=10

Effect of Loss Rate on Latency
- Bursty b=10
- Uniform
- Markov m=10

Works well at typical datacenter loss rates: 1-5%

Mahesh Balakrishnan
CS 514: Transport Protocols for Datacenters
Resilience to Burstiness
64 nodes, 128 groups per node, 10 nodes per group, Loss Model: Bursty 1%

... can handle short bursts (5-10 packets) well. Good enough?

Resilience to Bursty Losses: Recovery Percentage
Resilience to Bursty Losses: Recovery Latency

Mahesh Balakrishnan
CS 514: Transport Protocols for Datacenters
Staggering
64 nodes, 128 groups per node, 10 nodes per group, Loss Model: Bursty 1%

Staggering of $i$: Encode every $i$th packet
Stagger 6, burst of 100 packets $\implies$ 90% recovered at 50 ms!
Ricochet: Overview

- Time-Critical Datacenters:
  - large numbers of low-rate groups
  - bursty end-host loss patterns
- Ricochet is the first protocol to scale in the *number of groups* in the system
Outline

1. Motivation

2. Systems
   - Ricochet
   - Maelstrom

3. Conclusion
Open Problem: LambdaNets

- The Lambda Internet: A collection of geographically dispersed datacenters...
- ... connected by optical ‘lambda’ links
- Applications need to run over LambdaNets:
  - Financial services operating in different markets
  - MNCs with operations in different countries
  - High-volume e-tailers
Why is this hard?

- Speed of Light!
- Existing systems are not designed for very high communication latencies:
  - Try executing a Java RMI call on a server sitting in India...
  - Or mirroring your Oracle database to Kentucky...
- Need for fundamental redesign of software stack
TCP/IP uses RTT-based timeouts and retransmissions... ... hundreds of milliseconds to recover lost packets!

FEC: Perfect technology for long-distance transfer... ... but useless if loss is bursty.

Maelstrom: Decorrelated FEC — Constructs repair packets from across multiple outgoing channels from one datacenter to another
The applications you build will run on Datacenters
Current technology works... barely.
Next-generation applications will push the limits of scalability:
- What if all TV is IP-based (YouTube on steroids)?
- What if all your data/functionality is remote? (AJAX-based Apps...)
- What if *everything* is remote? (Web-based Operating Systems...)