Using Trio – Juniper Networks’ Programmable CHIPset – for Emerging In-Network Applications

Mingran Yang
Alex Baban, Valery Kugel, Jeff Libby, Scott Mackie,
Swamy Sadashivaiah Renu Kananda, Chang-Hong Wu, Manya Ghobadi
Data-intensive applications are the foundation of online services

- Moore’s law is slowing down
- We need efficient distributed systems for data-intensive applications
The rise of in-network computing

- Caching: NetCache [SOSP ’17]
- Database query processing: Cheetah [SIGMOD ’20]
- Machine learning training: SwitchML [NSDI ’21], ATP [NSDI ’21], PANAMA [MLSys ’21]

Reduce latency
Improve performance
Background on data parallel distributed machine learning training

Server 1
- DNN model
- Dataset

Server 2
- DNN model
- Dataset

Server 3
- DNN model
- Dataset

Server 4
- DNN model
- Dataset

Switch

Allreduce
In-network aggregation for distributed ML training

- Allreduce puts significant pressure on network fabric
- SwitchML [NSDI ’21] and ATP [NSDI ’21] proposed in-network aggregation
Stragglers impose a practical deployment challenge with in-network aggregation

- There has not been much attention to straggler mitigation for in-network computing applications
Enabling efficient in-network straggler mitigation in Tofino switches is challenging

- The switch needs to perform efficient timer-based operations to mitigate stragglers
  - Challenge: requires frequent interactions with the control plane
- Today’s Tofino-based solutions have no in-network straggler mitigation implemented
Trio-ML
Our proposed system for efficient in-network straggler mitigation

Achieve 1.8x faster training time for machine learning jobs by leveraging Juniper Networks’ programmable chipset.
Talk outline

• Overview of Juniper Networks’ chipset
• Trio-ML: our proposed in-network straggler mitigation for distributed ML training
• Evaluations
Talk outline

• Overview of Juniper Networks’ chipset
• Trio-ML: our proposed in-network straggler mitigation for distributed ML training
• Evaluations
Tofino vs. Trio (Juniper Networks’ programmable chipset)

Tofino programmable switches

- Pipeline₁
- Pipeline₂
- Pipeline₃
- Pipeline₄

Traffic Manager

Egress

- Pipeline₁
- Pipeline₂
- Pipeline₃
- Pipeline₄

Trio-based programmable switch

Packet forwarding engine 1
Packet forwarding engine 2
Packet forwarding engine 3
Packet forwarding engine N

Interconnection Fabric

Thread-based architecture
Trio’s thread-based architecture

Trio thread

Packet forwarding engine 1
Packet forwarding engine 3
Interconnection Fabric
Packet forwarding engine 2
Packet forwarding engine N

Trio-based programmable switch
Trio’s thread-based architecture

Trio-based programmable switch

Trio gracefully handles non-homogeneous packet processing rates
Talk outline

• Overview of Juniper Networks’ chipset
• Trio-ML: our proposed in-network straggler mitigation for distributed ML training
• Evaluations
Challenges for efficient in-network straggler mitigation

- In-network straggler detection:
  - Enable the switch to efficiently detect straggler events
  - Requires timer-based operations based on user-defined straggler timeout

- In-network straggler recovery:
  - Enable the switch to gracefully serve the job without waiting for stragglers
  - Requires a light-weight mechanism in the switch to proceed the computation

Our system, Trio-ML, addresses both challenges using Trio’s threads
Challenges for efficient in-network straggler mitigation

- In-network straggler detection:
  - Enable the switch to efficiently detect straggler events
  - Requires timer-based operations based on user-defined straggler timeout

- In-network straggler recovery:
  - Enable the switch to gracefully serve the job without waiting for stragglers
  - Requires a light-weight mechanism in the switch to proceed the computation

Our system, Trio-ML, addresses both challenges using Trio’s threads
Trio-ML gradient record

- Trio-ML creates a new gradient record when it receives a new packet from server

Trio-based switch memory

Server 1  
Server 2  
Server 3  
Server 4
Naïve approach for in-network straggler detection

- Goal: detect non-responsive servers within $\Delta$ milliseconds timeout

Trio-based switch memory

Gradient record
- Record ID: 1
- Job ID: Job_1
- List of servers: [S1, S2, S3, S4]
- Received gradients: [S1, S2, S3, --]

Naïve approach:
The switch launches a thread and starts a timer as soon as new packets arrive.
The naïve approach is not scalable for large ML models

**Trio-based switch memory**

- Gradient record 1 (missing gradients from server 4)
- Gradient record 2 (missing gradients from server 4)
- Gradient record 3 (missing gradients from server 4)
- ... (missing gradients from server 4)

The switch would need to create one thread per packet, and is not scalable for large ML models.

**Trio-based switch**

Servers send lots of gradient packets per training iteration

- **Server 1**
- **Server 2**
- **Server 3**
- **Server 4 (Straggler)**
Trio-ML’s approach: decouple timeout threads from packet arrivals

Every $\Delta$ timeout interval, Trio-ML launches threads to scan gradient records.
Trio-ML’s approach: decouple timeout threads from packet arrivals

Trio-based switch memory

<table>
<thead>
<tr>
<th>Thread</th>
<th>Gradient record 1 (missing gradients from server 4)</th>
<th>Gradient record 2 (missing gradients from server 4)</th>
<th>Gradient record M (missing gradients from server 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Time out! Straggler detected</td>
<td>Time out! Straggler detected</td>
<td>Time out! Straggler detected</td>
</tr>
<tr>
<td>2</td>
<td>Time out! Straggler detected</td>
<td>Time out! Straggler detected</td>
<td>Time out! Straggler detected</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>N</td>
<td>Time out! Straggler detected</td>
<td>Time out! Straggler detected</td>
<td>Time out! Straggler detected</td>
</tr>
</tbody>
</table>

After another $\Delta$ timeout interval, Trio-ML launches threads to scan gradient records again
Trio-ML’s approach: decouple timeout threads from packet arrivals

Trio-based switch memory

<table>
<thead>
<tr>
<th>Thread</th>
<th>Gradient record 1 (missing gradients from server 4)</th>
<th>Gradient record 2 (missing gradients from server 4)</th>
<th>Gradient record M (missing gradients from server 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Time out! Straggler detected</td>
<td>Time out! Straggler detected</td>
<td>Time out! Straggler detected</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>Gradient record (N-1)M (missing gradients from server 4)</td>
<td>Gradient record (N-1)M+1 (missing gradients from server 4)</td>
<td>Gradient record NM (missing gradients from server 4)</td>
</tr>
<tr>
<td></td>
<td>Time out! Straggler detected</td>
<td>Time out! Straggler detected</td>
<td>Time out! Straggler detected</td>
</tr>
</tbody>
</table>

Time out! Straggler detected

Trio-ML guarantees straggler detection within $[\Delta, 2\Delta]$ timeout window

After another $\Delta$ timeout interval, Trio-ML launches threads to scan gradient records again
Challenges for efficient in-network straggler mitigation

• In-network straggler detection:
  • Enable the switch to efficiently detect straggler events
  • Requires timer-based operations based on user-defined straggler timeout

• In-network straggler recovery:
  • Enable the switch to gracefully serve the job without waiting for stragglers
  • Requires a light-weight mechanism in the switch to proceed the computation

Our system, Trio-ML, addresses both challenges using Trio’s threads
Trio-ML sends partial aggregation results to recover from stragglers

- Partial aggregation can achieve comparable convergence performance (Yu et al., [ICML '19])
- Non-straggler servers continue to make progress without waiting
- Trio-ML keeps updating the straggler to ensure ML model is consistent
Talk outline

• Overview of Juniper Networks’ chipset
• Trio-ML: our proposed in-network straggler mitigation for distributed ML training
• Evaluations
Testbed setup

- Real-world testbed
- Comparisons
  - Our solution: Trio-ML
  - Tofino-based solution: SwitchML [NSDI ’21]
  - Ideal setup: PyTorch NCCL RDMA without adding stragglers
- Straggler generation pattern (Harlap et al. [SoCC ’16])
  - Three possible delay points per iteration
  - One server slows down at each delay point with given straggler probability \( p \)
  - Delay time at each delay point: uniformly randomly chosen between 0.5x and 2x of typical iteration time
Impact of straggling probability on training iteration time

Straggling Probability (%) vs. Training Iteration Time (ms)

VGG11 on ImageNet dataset
Trio-ML improves the time-to-accuracy by 1.60x

VGG11 on ImageNet dataset

Similar gains for ResNet50 and DenseNet161
Conclusion

• Stragglers impose a deployment challenge for in-network computing solutions.
• Trio-ML: a novel in-network straggler mitigation system using Juniper Networks’ thread-based programmable chipset.
• Trio-ML outperforms today’s in-network aggregation solutions by up to 1.8x.

Please email mingrany@mit.edu for more information!