Accelerating Distributed Systems with In-Network Computing

Wenfei Wu

June, 2023
In-Network Computing (INC)

- Offload application functions to network devices
- Benefit from recent programmable network devices
  - Intel/Barefoot Tofino, Cisco Sillicon One, Broadcom Trident
- Benefit applications
  - High throughput
  - Sub-RTT latency
  - High computation speed
In-Network Computing (INC)

- Offload application functions to network devices
- Benefit from recent programmable network devices
- Benefit applications
- A promising approach to accelerate and scale distributed systems

Moore’s Law is slowing down.

Computation requirement is exponentially increasing.

Distributed systems (e.g., distributed training) are hard to scale well.
INC changes network architecture (1/2)

Traditional Networks

- Has a layered architecture
- advantage: evolve independently, support the Internet ecosystem.
- disadvantage: not efficient
- diverse applications have their own communication libraries.
  - e.g., MPI, NCCL.

Presented by Wenfei Wu
INC changes network architecture (2/2)

- Trend of large clusters (for AI, Big Data)
  - Belong to a single party (no ecosystem)
  - Each layer can be programmed
  - Significant gain to promote efficiency

Roadmap

- Separate Application-network codesign
- Build universal libraries for developers
- Build new devices for INC.
State-of-the-art INC research

Application-network codesign (use cases)

machine learning, data analytics, caches, consensus protocols.

Programming frameworks

Targetting network operators; few are for application developers.

Runtime management

Switch memory management.
Our practice

Distributed Systems
- Machine Learning
- ATP/NetReduce
- Data Analytics
- ASK
- Others

Runtime Management
- INAloc
- Controller
- Controller

Communication Library
- NetRPC, ClickINC

Device
- Programmable Network Devices

Presented by Wenfei Wu
Contents

- INC Background
- Our practice
  - Accelerating Machine Learning [NetReduce, ASPLOS23; ATP, NSDI21]
  - Accelerating Data Analytics [ASK, ASPLOS23]
- Conclusion
Machine Learning Algorithm

- Algorithm: iterate until convergence
  - Compute a gradient
  - Update the model
- Distributed Training (data parallel)
  - Each worker computes a gradient
  - Aggregate all workers' gradients, and send result back (AllReduce)
  - Work updates model

\[
W(t+1) := W(t) - \alpha \nabla J(W(t), D(t))
\]

\[
W(t+1) := W(t) - \alpha \sum_{p=1}^{P} \nabla J(W(t), D_p(t))
\]
Parameter Server, PS

• Assume: $N$ workers, gradient size is $M$
• Data Volume
  ○ worker: $M$
  ○ PS: $N \times M$

Problem: bottleneck @ PS Link

• Slow down the training
  ○ e.g., VGG16 by 4 times
Goals

- Primary Goal: Accelerate the ML system
- Secondary Goals
  - Compatibility
  - Multi-tenancy
  - cross-rack
  - ...

<table>
<thead>
<tr>
<th></th>
<th>Solution 1</th>
<th>Solution 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Job Acc.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Transport-Layer Friendly</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Multi-job Efficiency</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Cross-rack</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Different scenarios lead to different requirements, and different designs.
Relationship between research works

Multi-Job Acc  Multi-Job Mgmt
ATP                INAloc

Single-Job Acc.
SwitchML

Single-Job Acc.
NetReduce

Time
Solution 1: Replace PS by Switch [NetReduce, ASPLOS23]

- Primary Goal: Accelerate Machine Learning
- Secondary Goal: Be RDMA-Compatible
How does the PS work?

1. Each worker sends gradient to PS.
2. PS sums up gradients
3. PS multicasts result to workers.

Problem: Can the switch do this?

- Switch cannot cache large pieces of data
- Switch does not have transport layer functions, thus, cannot guarantee data integrity.
PS Workflow

PS

Switch

Worker 1
Gradient

Worker 2
Gradient
Design: Architecture and Data Structure

- Worker: chuck the gradient as a packet sequence
  - Maintain a sliding window to send packets, assume window size $W$
- Switch: organize the memory as an array, with size $N$

Worker1’s Gradient

Worker2’s Gradient

Switch

Agtr Array
Workflow

- Worker always sends packets in the window
- Switch addresses a packet to an aggregator
  - Modulo Addressing: \( \text{agtr}.idx \leftarrow PSN\%N \)
  - Implying \( W \leq N \) (necessary not sufficient)
- When switch completes an aggregation (judged by bitmap), multicasts ACK to workers
  - ACK piggybacks the result
  - Release the aggregator at \((i + W)\%N\) (explain later)
- ACK advances the sliding window, and server sends new packets.
Workflow (ideal case)
Workflow (ideal case)

Switch

Agtr Array

Size is 2

Gradient 5 4 3 2 1 0

Window Size is 2

Gradient 5 4 3 2 1 0

worker

worker
Workflow (ideal case)

Switch

Agtr Array

Size is 2

Gradient worker 5 4 3 2 1 0

Gradient worker 5 4 3 2 1 0

Window Size is 2
Problem: what if packet is lost?

We need correct computation, but network is not reliable.

- Worker: if it does not receive ACK (timeout), resends gradient packets
- Switch: could receive duplicate gradient packets, but should avoid duplicate computation
  - Aggregator bitmap records whether each work participates computation.
  - If a worker has participated, it skips the duplicate computation.
Challenge: Recycle switch memory

- If aggregator array size is smaller than a gradient, how to recycle the array?
  - i.e., when should the switch release an aggregator?

- It is incorrect if the switch releases an aggregator when multicasting the ACK
  - Lossless workers: move window forward, send new packets
  - Lossy workers: do not move window, send old packets
  - In switch: a deadlock happens
Problem: ACK Loss

Switch

Agtr Array

Size is 2

Gradient 5 4 3 2 1 0

window worker

Gradient 5 4 3 2 1 0

Window Size is 2
Problem: ACK Loss

Switch

Agtr Array

Size is 2

Window Size is 2

Gradient worker

5 4 3 2 1 0

Gradient worker

5 4 3 2 1 0

Switch

Gradient worker

5 4 3 2 1 0

Gradient worker

5 4 3 2 1 0
Challenge: Recycle switch memory

- Solution: a packet releases an aggregator "one-window" away
  - the $i$-th packet releases the $(i + W)$-th aggregator
- Implied $N \geq 2W$ (necessary and sufficient)
  - When packet $i$ appears, all packets in $(i - W, i + W)$ could be in aggregator
  - $(i + W) \% N$ (to release) should not overlap this interval
  - It requires at least $2W$ aggregators.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Agtr Array Size</td>
</tr>
<tr>
<td>$W$</td>
<td>(Max) Window Size</td>
</tr>
</tbody>
</table>

PS: The "twice-of-window" design is like "shadow copy" technique in SwitchML.
Problem: ACK Loss

- Size is 4
- Switch
  - Agtr Array
    - Gradient: 5 4 3 2 1 0
    - Window Size is 2
  - Worker
    - Gradient: 5 4 3 2 1 0
    - Window Size is 2
Problem: ACK Loss

Switch

Agtr Array

Size is 4

Gradient worker 5 4 3 2 1 0

Gradient worker 5 4 3 2 1 0

Window Size is 2
Secondary Goal: Transport Transparency

Problem: Need to replace Transport Layer (TCP/RDMA)

- Does not benefit from recent transport: (RDMA over Converged Ethernet, RoCE)
- Redundant development of transport layer.
- Need to isolate INC traffic in runtime

Table 1: Performance of RoCE, parallel DMA, and DPDK.

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>Throughput (Gbps)</th>
<th>RTT (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RoCE</td>
<td>25.6%</td>
<td>84.2</td>
<td>5.7</td>
</tr>
<tr>
<td>Parallel DMA</td>
<td>100%</td>
<td>58.67</td>
<td>70.2</td>
</tr>
<tr>
<td>DPDK</td>
<td>200%</td>
<td>90.5</td>
<td>20.5</td>
</tr>
</tbody>
</table>

Table 2: Lines of code of functions in ATP network stack.

<table>
<thead>
<tr>
<th>Function</th>
<th>Lines of Code</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packetization &amp; IO</td>
<td>1090</td>
<td>32.12%</td>
</tr>
<tr>
<td>Flow Control</td>
<td>50</td>
<td>1.47%</td>
</tr>
<tr>
<td>Reliability</td>
<td>181</td>
<td>5.33%</td>
</tr>
<tr>
<td>Congestion Control</td>
<td>64</td>
<td>1.89%</td>
</tr>
<tr>
<td>Floating Point Support</td>
<td>220</td>
<td>6.48%</td>
</tr>
<tr>
<td>Fallback</td>
<td>100</td>
<td>2.95%</td>
</tr>
<tr>
<td>Others</td>
<td>1689</td>
<td>49.76%</td>
</tr>
<tr>
<td>Total</td>
<td>3394</td>
<td>100%</td>
</tr>
</tbody>
</table>

Figure 2: Bandwidth contention of VGG16 in ATP and background DCTCP, cited from [35].

Presented by Wenfei Wu
Intuition to achieve the transparency

- Splice network flows in switch, making end host not perceive the change.
Architecture and Workflow

- Host: keep RoCE
  - Add flow control to send gradients
  - Establish connections along a ring
- Switch
  - Allocate an aggregator array ($N$ units)

Workflow

- Sender: send messages with flow control
- Switch: address packets to aggregators
  - Aggregate packets
  - Recovery packet header
- Receiver: receive result packets
Establish RDMA Connections
Packet Addressing

Packet Addressing

Packet

Aggregator Array

checksum

addressing
Aggregation Process

Packet buffer

Packet indexed by Rank

bitmap

header header header

payload payload payload

packet1 packet2 packets

value aggregate
Results

- Performance gain from both INA and RDMA
  - AlexNet is promoted by 45%.
Summary

- Replace PS by switch for performance gain.
- Achieve transport transparency for further gain.

**PS:** We make another implementation of NetReduce on Tofino.
Solution 2: New Background

Training jobs in production clusters

- Multi-tenancy
  - Multiple jobs share the cluster
- Multi-rack topology
  - e.g., BERT-Large training.

<table>
<thead>
<tr>
<th>Time</th>
<th>System</th>
<th>Number of Nodes</th>
<th>Number of V100 GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>47min</td>
<td>DGX SuperPOD</td>
<td>92 DGX-2H</td>
<td>1472</td>
</tr>
<tr>
<td>67min</td>
<td>DGX SuperPOD</td>
<td>64 DGX-2H</td>
<td>1024</td>
</tr>
</tbody>
</table>
Goals

- Primary Goal: Accelerate Machine Learning
- Secondary Goal: Multi-tenancy support
Solution: Aggregation Transmission Protocol (ATP)

Features

- In-network aggregation
  - Reliability
- Multi-tenancy
  - Congestion Control
  - Across racks
  - Floating-point support
Multi-jobs: Static memory allocation

- Divide switch memory into regions, assign regions to tenants.
- Shortcomings
  - Switch memory efficiency is low due to iterative computation and communication.
  - Management complexity is high, needing to integrate the INC and ML controllers.

![Diagram showing multi-jobs with switch, workers, and registers]
Dynamic Switch Memory Allocation

- Switch memory as an aggregator pool
  - Serve packets with First-Come-First-Serve (FCFS)
- Decentralized Addressing: $\text{Agtr.idx} \leftarrow \text{Hash(JobID, PSN)}$

Presented by Wenfei Wu
Architecture (Changes with Solution 1)

- Keep PS, because
  - Addressing could fail, which needs a fallback
  - Switch cannot handle retransmission packets correctly.
- Aggregator
  - Keep $<\text{JobID}, \text{PSN}>$ to detect hash conflicts
  - Released by ACK
Workflow (Ideal case)

- Worker sends gradient packets in window
- When a gradient packet arrives at the switch
  - Addressing succeeds, perform aggregations, send result to PS
  - Addressing fails, pass to PS
- PS completes aggregation, and replies ACK
- ACK arrives at the switch, and **releases the aggregator**
- ACK is multicast to workers

Do not release an aggregator "one-window" away "hash(JobID, PSN+W)", which may be used by another job.
ATP Workflow (ideal case)
Ampere Tensor Processing (ATP) Workflow (ideal case)

\[
d_1 c_1 b_1 a_1 \\
d_2 c_2 b_2 a_2 \\
...... \\
d_n c_n b_n a_n
\]
A TP Workflow (ideal case)

\[
\begin{array}{cccc}
  d_1 & c_1 & b_1 & a_1 \\
  d_2 & c_2 & b_2 & a_2 \\
  \vdots & \vdots & \vdots & \vdots \\
  d_n & c_n & b_n & a_n \\
\end{array}
\]
Problem: non-ideal cases

ACK is lost, and worker retransmits gradient packets
Switch behavior?

- It should never restart aggregator for retransmit packets.
ATP Workflow (Packet Loss)

\[a_1 + a_2 + ... + a_n\]

Switch

Job 2

Worker 1

Worker 2

Worker n

\[d_1 c_1 b_1 a_1\]

\[d_2 c_2 b_2 a_2\]

\[......\]

\[d_n c_n b_n a_n\]

Or timeout

\[a_2 + a_3 + ... + a_n\]
ATP Workflow (Packet Loss)
ATP Workflow (Packet Loss)
ATP Workflow (Packet Loss)

One Aggregator

Bitmap

Values

Switch

Job 2

Worker 1

Worker 2

......

Worker n

[Diagram of job and worker distribution with values associated]
Single-Job Performance

- Metric: Average images per second on each server
- ATP outperforms the PS architecture, by eliminating the bottleneck
- ATP outperforms Ring AllReduce, but consumes half network capacity.
Multi-job: Dynamic vs. Static Allocation

- Three VGG16 jobs
- Tune switch memory to obtain peak throughput
  - Called Peak Throughput Aggregators (PTA)
  - Reduce switch memory to cause contention
- Dynamic allocation degrades more gracefully.
Summary

- Statistical multiplexing switch memory is more efficient
- But ATP has a higher complexity for correctness guarantee.
## Comparison of two solutions

<table>
<thead>
<tr>
<th></th>
<th>Solution 1</th>
<th>Solution 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Organization</strong></td>
<td>Switch</td>
<td>Switch+PS</td>
</tr>
<tr>
<td><strong>Addressing</strong></td>
<td>Collision Avoidance</td>
<td>Collision Detection</td>
</tr>
<tr>
<td><strong>Switch Memory Usage</strong></td>
<td>Twice of window</td>
<td>Once of Window</td>
</tr>
<tr>
<td><strong>Switch Memory Allocation</strong></td>
<td>Isolated</td>
<td>Shared</td>
</tr>
<tr>
<td><strong>Scenarios</strong></td>
<td>Single Job</td>
<td>Multiple Jobs</td>
</tr>
<tr>
<td><strong>Advantage</strong></td>
<td>Predictable Performance</td>
<td>High Resource Utilization</td>
</tr>
</tbody>
</table>
Summary

- Switch could accelerate distributed model training.
  - e.g., ATP, SwitchML, NetReduce
- To guarantee computation correctness in unreliable networks is challenging.
- An application scenario has specific requirements, leading to different designs.
  - Compatibility
  - Multitenancy
  - etc.
Contents

- INC Background
- Our practice
  - Accelerating Machine Learning [NetReduce, ASPLOS23; ATP, NSDI21]
  - Accelerating Data Analytics [ASK, ASPLOS23]
- Conclusion
Data analytics has stream aggregation

- `ReduceByKey()` in Big Data
- `sum`, `count` followed by `group by` in database
The aggregation is asynchronous aggregation

Value Stream

Synchronous Aggregation (ML)
- Keys (or index) are the same.
- Keys are linearly aligned.
- Key appearance is bounded.

Key-value Stream

Asynchronous Aggregation
- Keys are different
- Keys are unforeseeable.
- Keys are unbounded.
Goal

INC to accelerate data analytic systems.

- Be a generic service
- Has correctness guaranteee
Strawman Solution

A solution to show the feasibility, with three assumptions.

- Each packet carries one key-value tuple
- No packet loss in the network
- Switch memory can hold all keys with pre-computed addressing.

(On a 56-core server) Effect: accelerate by three times.
ASK Architecture and Workflow

- Host runs an agent to support applications
- Agent processes requests in FIFO
- Each request is a job
ASK Architecture and Workflow

- Each job has multiple senders and one receiver.
  - Dynamic addressing $agtr.idx \leftarrow \text{Hash}(key)$
  - Switch: Best-effort + Fallback
  - Receiver fetches results from the switch
Challenge 1: Improve Goodput

Problem: low goodput

- 100Gbps throughput has 9.76Gbps goodput
- Improvement 1: one packet carries multiple tuples.

New problem: one-time memory access on switch
Challenge 1: Improve Goodput

- Design
  - Multi-key packet
- Improvement 2: 2D Aggregator Array (AA)
  - Each AA for one tuple.

New Problem: Single-key-multiple-spot
Challenge 1: Improve Goodput

- Design
  - Multi-key packet
  - 2D Aggregator Array
- Improvement 3: Reconstruct key orders
  - Divide key space into non-overlapping subspaces
  - Associate subspaces, packet tuple slots, and AAs.
Challenge 1: Improve Goodput

- Design
  - Multi-key packet
  - 2D Aggregator Array
  - Sender-Assisted packet construction and addressing
- Result: 155 times acceleration
Challenge 2: Reliability and Correctness

- Multi-key packet causes a special case.
  - A packet can be partially aggregated.
  - How to handle a retransmitted partially aggregated packet at the switch?
    - Drop?
    - Aggregate?
    - Pass?
    - None is right.
Challenge 2: Reliability and Correctness

- Record each packet's appearance \textit{seen} and aggregation state \textit{AggState}
  - First Appearance: record appearance and Aggregation State
    \[
    \text{Switch.AggState} = \text{Pkt.Bitmap}
    \]
  - Later Appearance: copy first-time state
    \[
    \text{Pkt.bitmap} = \text{Switch.AggState}
    \]
- State Explosion? \textit{seen} and \textit{AggState} are bounded by window size; jobs multiplex persistent connections.

Presented by Wenfei Wu
Result: Performance

- Workload: WordCount
- Result: Accelerate the job by 3-5 times.
Result: Overhead

- ASK (4-core) JCT is smaller than Spark (56-core) by 50%.
Summary

- INC can promote data analytics
  - Performance gain is from computation offload
- System design is constrained by hardware capability.
- Correctness is still challenging.
Takeaway

- INC is an effective approach to accelerate and scale distributed systems
  - Leverage on-path switch, without extra overhead
  - Switch is more power-efficient than CPU and GPU
- There is a large research space in this direction.
- Our practice
  - Two applications: Machine learning, Data Analytics (NSDI21, ASPLOS23*2)
  - Programming Frameworks (NSDI23, SIGCOMM23)
  - Runtime Resource Management (INFOCOM23)
Future Work

Distributed Systems
- Machine Learning (ATP/NetReduce)
- Data Analytics (ASK)
- Others

Runtime Management
- INAlloc
- Controller

Communication Library
- NetRPC, ClickINC

Device
- Programmable Network Devices

Presented by Wenfei Wu
End

• Thanks!
• Q&A
• Collaborations are welcome!
• Homepage: http://wenfei-wu.github.io/
• Email: wenfeiwu@pku.edu.cn