USING LIBFABRIC FOR SCALABLE DISTRIBUTED MACHINE LEARNING: USE CASES, LEARNINGS, AND BEST PRACTICES

Rashika Kheria, Senior SDE
Amazon Web Services
June, 2020
AGENDA

- Introduction to distributed machine learning training
- Distributed machine learning training stack
- NCCL and its integration with libfabric
- Performance impact of running EFA for machine learning
- Deploying at scale – learnings and challenges
INTRODUCTION
DISTRIBUTED MACHINE LEARNING TRAINING

- **Supervised Machine learning (ML) training**
  - Train a model to determine optimal values for all the weights and bias from labelled examples
  - Apply learnt model to infer results for new real world data
  - Some applications: Image and Speech recognition, classification, prediction

- **Size of ML models is growing way faster than Moore’s law**
  - One of most popular ML algorithm (BERT) takes more than 9 days to train on biggest available single server

- **Meanwhile, ML researchers and commercial users are looking for results and experiments in hours, not days**

- **Solution: Run distributed ML training with Data Parallelism**
  - Same model, different training samples
SYNCHRONOUS STOCHASTIC GRADIENT DESCENT (SGD)

- Iterative algorithm to calculate optimal training parameters
- Results of each round is incorporated into the model for the next round.
- Applies to only set of inputs called as mini-batch

Diagram:
1. Send local gradients
2. Wait for gradients from all GPUs
3. Update model and send it to all GPUs before next round
DISTRIBUTED ML TRAINING IN REAL LIFE

- Ideal goal: Scale linearly with number of GPU servers
- Catch: You need to balance Time-to-Converge while scaling the Teraflops
- Classic trade-off:

<table>
<thead>
<tr>
<th>Compute per epoch</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large</td>
<td>Reduce communication overhead per compute</td>
<td>Slower convergence time</td>
</tr>
<tr>
<td>Short</td>
<td>Faster convergence time</td>
<td>Higher communication overhead per compute</td>
</tr>
</tbody>
</table>

➔ Idea: A faster and lower latency interconnect allows scaling to large number of nodes while keeping short epoch
  - Or allows model to converge faster for same number of nodes by offering lower communication per compute
DISTRIBUTED ML STACK
DISTRIBUTED ML TRAINING ARCHITECTURE IN AWS

- Bert
- MaskRCNN
- Jasper
- Transformer

- ML Training Models
- ML Training Frameworks
- Collective Communication Primitives
- Networking
- Hardware Infrastructure

NCCL
AWS OFI NCCL
Libfabric
EFA
EC2

© OpenFabrics Alliance
DISTRIBUTED ML TRAINING ARCHITECTURE IN AWS

ML Training Models
- Bert
- MaskRCNN
- Jasper
- Transformer

ML Training Frameworks
- TensorFlow
- PyTorch
- mxnet

Collective Communication Primitives
- NCCL

Networking

Hardware Infrastructure
USING LIBFABRIC WITH NCCL

- **Why NCCL?**
  - Multi-GPU and Multi-node collective communication primitives
  - Provides routines such as all-gather, all-reduce, broadcast, reduce, reduce-scatter
  - Performance optimized for NVIDIA GPUs

- **NCCL collective communication flow**
  - Determines system topology for efficient communication
  - Builds collective operations topology like ring, double binary tree
  - Schedules CUDA kernels on GPU which reduces and moves the data

- **AWS OFI NCCL**
  - Open source plug in which enables to use libfabric as a network provider while running NCCL based applications
  - Enables to use Elastic Fabric Adapter (EFA) on AWS EC2 cloud
ELASTIC FABRIC ADAPTER

- Libfabric provider optimized for high performance and machine learning workloads
- Low latency for inter node communication in cloud
- Scales to thousands of CPUs / GPUs
- Bypasses operating system
- Uses underlying Scalable Reliable Datagram Protocol
- To learn more: https://aws.amazon.com/hpc/efa/
PERFORMANCE RESULTS
PERFORMANCE BENCHMARKS

▪ **BERT training**
  • Dataset by NVIDIA - concatenation of Wikipedia as well as Book Corpus
  • Gradient accumulation to simulate larger batch size
  • Smaller sequence length and higher batch size to speed up the training
  • **Throughput Scaling efficiency: 87%** - Scale training time from 9 days to <1 hour
  • EFA improves training time by 2X for a cluster of 256 GPUs vs TCP

▪ **Maskrcnn Training for Object Detection**
  • Used COCO dataset
  • Configured model hyperparameters for different system scale.
  • **EFA outperformed TCP at all scales, processing >1000 images/second with 192 GPUs**

© OpenFabrics Alliance
TRAINING TIME IMPROVEMENT WITH BERT

Time to Train: BERT-Large, TensorFlow

- 9 days, 8 hours
- 1 hour
PERFORMANCE IMPROVEMENT WITH MASKRCNN

Mask R-CNN Performance with COCO dataset

<table>
<thead>
<tr>
<th>NUMBER OF GPUs</th>
<th>IMAGES/SEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>67.95</td>
</tr>
<tr>
<td>16</td>
<td>71.03</td>
</tr>
<tr>
<td>16</td>
<td>95.187</td>
</tr>
<tr>
<td>32</td>
<td>126.4</td>
</tr>
<tr>
<td>32</td>
<td>176.61</td>
</tr>
<tr>
<td>192</td>
<td>864.3</td>
</tr>
<tr>
<td>192</td>
<td>1031.5</td>
</tr>
</tbody>
</table>

- p3dn.24x1 (100Gbps, TCP)
- p3dn.24x1 (100Gbps, EFA)
DEPLOYING AT SCALE
LEARNING FROM DEMANDING CUSTOMER WORKLOADS

- **Working with customers running different ML models for training**
  - Spanning from 8 GPUs to 1200 GPUs
  - Using different ML frameworks like PyTorch, MxNet and Tensorflow

- **P3DN servers**
  - Complex instances built for running ML training at scale
  - More than 5kW per each server, 1TB of DRAM, 10 chips with 200Watt+ each and 100s of billions of transistors in one chip
  - Careful design for reliability

- **Successful multi day trainings with consistent performance, reliability, and no hick-ups**
  - Robust testing and qualification of all software stack: Drivers / Kernel / NCCL / LibFabric / ML Framework
  - Non distributive firmware updates
  - Robust power and network delivery
  - Developed auto-recovery mechanisms to recover from transient failures like GPU memory errors
BEST PRACTICES FOR SCALE

▪ Know your software stack
  • Challenge: Many components involved and small bugs magnify at scale
  • Fast moving code which addresses these bugs
  • Use latest NVIDIA toolkit, OS distribution etc.

▪ Choose appropriate hardware / services
  • Challenge: Highly demanding and resource intensive applications.
  • Make sure you’re sizing appropriately to avoid bottlenecks
  • Amazon FSx for lustre, EC2 Hardware, VPC and 100G EFA

▪ Design for robustness
  • Challenge: Complicated components and applications which are sensitive to failure
  • Add capability to snapshot and restart from saved checkpoint. Eg: ParallelCluster with Slurm
  • AWS monitors hardware components for failure and recovers automatically. Adding similar monitoring and resiliency for VMs is also helpful.
SUMMARY

- Lower latency interconnect allows distributed ML models to converge faster
- EFA and libfabric enables fast, low latency communication
- Order of magnitude improvements in scalability compared to a TCP networking stack

Running demanding ML applications at scale requires:

- Careful hardware sizing
- Failure tolerance
- Keeping up to date with latest versions
THANK YOU

Rashika Kheria, Senior SDE

Amazon Web Services

(rashika@amazon.com)