Distributed Deep Learning on Mesos with GPUs and Gang Scheduling

Min Cai, Alex Sergeev, Paul Mikesell, Anne Holler, UBER
Who are we?

Min Cai  Alex Sergeev  Paul Mikesell  Anne Holler
Deep Learning @ UBER

• Use cases:
  – Self-Driving Vehicles
  – Trip Forecasting
  – Fraud Detection
Self-Driving Vehicles
Self-Driving Vehicles
Trip Forecasting
Fraud Detection

Spam referral code

Partner up with the same driver

Cash out Uber credits
Why Distributed Deep Learning?

- Speed up model training
- Scale out to hundreds of GPUs
- Shard large models that cannot fit into a single machine
How Distributed Deep Learning Works

1. Read Data
2. Compute Model Updates (Gradients)
3. Average Gradients
4. Update Model
Why Mesos?

- Widely adopted
- GPU Support
- Nested Containers
- Highly Customizable
- Reliable and Scalable
Mesos Support for GPUs

- Mesos Containerizer only
- Docker Containerizer support is not landed to upstream yet
Mesos Nested Containers

- Separate management code from user docker images
- Avoid dependency conflict
What is Missing?

- Elastic GPU Resource Management
- Locality and Network aware Placement
- Gang Scheduling
- Task Discovery
- Failure Handling
Peloton Overview

<table>
<thead>
<tr>
<th>Application Workflow (e.g Map-Reduce)</th>
<th>Borg</th>
<th>YARN</th>
<th>Kubernetes</th>
<th>Peloton</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller Job</td>
<td></td>
<td></td>
<td></td>
<td>Controller Job</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Job/Task Lifecycle</th>
<th></th>
<th>kube-controller</th>
<th>Peloton</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Task Placement</th>
<th>Borgmaster</th>
<th></th>
<th>Peloton</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Task Preemption</th>
<th>Resource Manager</th>
<th>kube-scheduler</th>
<th>Mesos Master</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Resource Allocation</th>
<th></th>
<th></th>
<th>Mesos Agent</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Task Execution</th>
<th>Borglet</th>
<th>Node Manager</th>
<th>kubelet</th>
<th>Mesos Agent</th>
</tr>
</thead>
</table>
Elastic GPU Resource Management

Cluster Capacity

Reservation

Weighted-Fairshare

Max

Org-1

Org-2

Org-3

High

Job Priority

Low
Resource Pools

Diagram:
- Root
  - Org-1
    - Team-1
    - Team-2
  - Org-2
    - <R, L, S>*

Resources:
- Limit (L)
- Entitlement (E)
- Share (S)
- Reservation (R)
- CPU/Mem/Disk/GPU
Gang Scheduling

- A subset of Tasks in a Job can be specified for Gang Scheduling
- Gang tasks are a single scheduling unit
  - Admitted, placed, preempted and killed as a group
- Gang tasks are independent execution units
  - Run in separate containers and may fail independently
- Gang execution is terminated if a gang task fails and cannot be restarted
Placement Strategies

• Place as many as container into the same host or rack
• Best fit algorithm to tightly packing GPU containers
• Constraint based placement for same generation of GPUs
Why TensorFlow?

• Most popular Open Source framework for Deep Learning
• Combines high performance with ability to tinker with low level model details
• Has end-to-end support from research to production
Architecture for Distributed TensorFlow

Averages All the Gradients

Parameter Server

Worker A  Worker B  Worker C

or

Each Averages Portion of the Gradients

Parameter Server A  Parameter Server B  Parameter Server C

Worker A  Worker B  Worker C
Architecture for Distributed TensorFlow on Mesos

- Michelangelo Deep Learning Service
  - CreateJob
  - Watch
  - LaunchTask

- Peloton

- Mesos Master

- Cassandra

- Mesos Agent
  - Michelangelo Container
    - Param Server Container
    - Worker A Container
    - Resolve
  - Michelangelo Container
    - Worker B Container
    - Resolve
  - Michelangelo Container
    - Worker C Container
    - Resolve
## Distributed Training Performance

Training with synthetic data on NVIDIA® Pascal™ GPUs

<table>
<thead>
<tr>
<th>Images/sec</th>
<th>Inception V3</th>
<th>ResNet-101</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>148.8</td>
<td>136.0</td>
</tr>
<tr>
<td>8</td>
<td>932.1</td>
<td>689.7</td>
</tr>
<tr>
<td>16</td>
<td>1,769.5</td>
<td>1,281.1</td>
</tr>
<tr>
<td>32</td>
<td>3,179.8</td>
<td>2,211.2</td>
</tr>
<tr>
<td>64</td>
<td>5,297.4</td>
<td>4,269.2</td>
</tr>
</tbody>
</table>

*Distributed TensorFlow*
### Distributed Training Performance

**Training with synthetic data on NVIDIA® Pascal™ GPUs**

<table>
<thead>
<tr>
<th>Scaling Factor</th>
<th>Inception V3</th>
<th>ResNet-101</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>8</td>
<td>6.3</td>
<td>5.1</td>
</tr>
<tr>
<td>16</td>
<td>11.9</td>
<td>9.4</td>
</tr>
<tr>
<td>32</td>
<td>21.4</td>
<td>16.3</td>
</tr>
<tr>
<td>64</td>
<td>35.6</td>
<td>31.4</td>
</tr>
</tbody>
</table>

**Legend:**
- Distributed TensorFlow
Can We Do Better?

- Improve communication algorithm
- Use RDMA-capable networking (RoCE, InfiniBand)
Horovod

- Distributed training framework for TensorFlow
- Uses bandwidth-optimal communication protocols
  - Makes use of RDMA (RoCE, InfiniBand) if available
- Seamlessly installs on top of TensorFlow via pip install horovod
Architecture for Horovod

Architecture for Horovod on Mesos
### Distributed Training Performance with Horovod

<table>
<thead>
<tr>
<th>Images/sec</th>
<th>Training with synthetic data on NVIDIA® Pascal™ GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Inception V3</td>
</tr>
<tr>
<td>0</td>
<td>148.8</td>
</tr>
<tr>
<td>1,000</td>
<td>932.1</td>
</tr>
<tr>
<td>2,000</td>
<td>1,123.9</td>
</tr>
<tr>
<td>3,000</td>
<td>1,769.5</td>
</tr>
<tr>
<td>4,000</td>
<td>2,198.8</td>
</tr>
<tr>
<td>5,000</td>
<td>3,179.8</td>
</tr>
<tr>
<td>6,000</td>
<td>4,192.4</td>
</tr>
<tr>
<td>7,000</td>
<td>5,297.4</td>
</tr>
<tr>
<td>8,000</td>
<td>7,932.1</td>
</tr>
</tbody>
</table>

*Images/sec*
Distributed Training Performance with Horovod

Training with synthetic data on NVIDIA® Pascal™ GPUs

<table>
<thead>
<tr>
<th>Scaling Factor</th>
<th>Inception V3</th>
<th>ResNet-101</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>8</td>
<td>6.3</td>
<td>5.1</td>
</tr>
<tr>
<td>16</td>
<td>11.9</td>
<td>7.3</td>
</tr>
<tr>
<td>32</td>
<td>14.8</td>
<td>9.4</td>
</tr>
<tr>
<td>64</td>
<td>21.4</td>
<td>14.6</td>
</tr>
<tr>
<td></td>
<td>28.2</td>
<td>16.3</td>
</tr>
<tr>
<td></td>
<td>35.6</td>
<td>29.1</td>
</tr>
<tr>
<td></td>
<td>53.3</td>
<td>31.4</td>
</tr>
<tr>
<td></td>
<td>56.9</td>
<td></td>
</tr>
</tbody>
</table>

- Orange: Distributed TensorFlow
- Green: Horovod
What About Usability?

```python
import tensorflow as tf
import horovod.tensorflow as hvd

# Initialize Horovod
hvd.init()

# Pin GPU to be used
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())

# Build model...
loss = ...
opt = tf.train.AdagradOptimizer(0.01)

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)

# Add hook to broadcast variables from rank 0 to all other processes during initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# Make training operation
train_op = opt.minimize(loss)

# The MonitoredTrainingSession takes care of session initialization,
# restoring from a checkpoint, saving to a checkpoint, and closing when done
# or an error occurs.
with tf.train.MonitoredTrainingSession(checkpoint_dir='/tmp/train_logs',
                                       config=config, hooks=hooks) as mon_sess:
    while not mon_sess.should_stop():
        # Perform synchronous training.
        mon_sess.run(train_op)
```

THE LINUX FOUNDATION
Horovod is available on GitHub today

https://github.com/uber/horovod
Thank you!

Any questions?