KubeCon

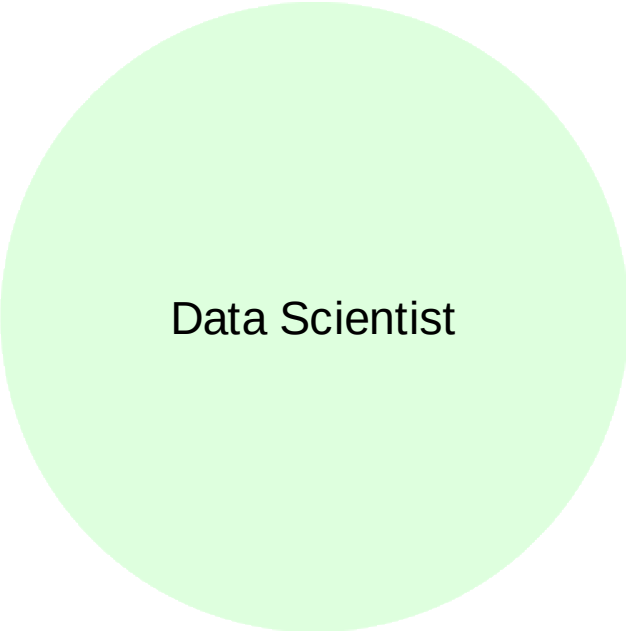
North America 2017

Democratizing Machine Learning on Kubernetes

Joy Qiao, Senior Solution Architect - AI and Research Group, Microsoft
Lachlan Evenson - Principal Program Manager – AKS/ACS, Microsoft

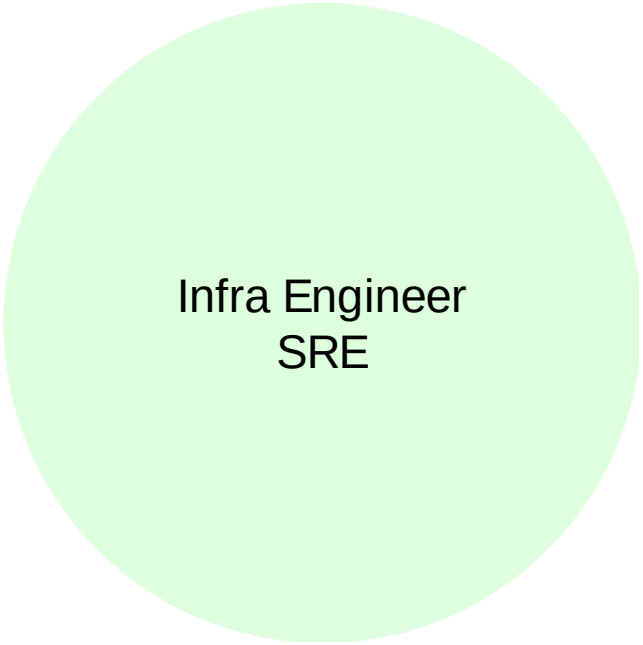
Who are we?

- The Data Scientist
- Building and training models
- Experience in Machine Learning libraries
- Basic understanding of computer hardware
- Lucky to have Kubernetes experience



Who are we? (*continued*)

- The Infra Engineer/SRE
- Build and maintain baremetal/cloud infra
- Kubernetes experience
- Little to no Machine Learning library experience



ML on Kubernetes

Data Scientist

Infra Engineer
SRE

Why this matters

- We have the RIGHT tools and libraries to build and train models
- We have the RIGHT platform in Kubernetes to run and train these models

What we've experienced

- Two discrete worlds are coming together
- The knowledge is not widely accessible to the right audience
- Nomenclature
- Documentation and use-cases are lacking
- APIs are evolving very fast, sample code gets out of date quickly

How do we?

- Enable Data scientists to be successful on Kubernetes
- How do we enable Infrastructure engineers/SREs to build ML platforms
- Lower the barrier to entry
- Begin to build some best-practices and baselines

Let’s get started

Running Distributed TensorFlow on Kubernetes

In just 4 simple steps

Running Distributed TensorFlow on Kubernetes *(continued)*

1. Create a Kubernetes cluster
   - PV for central storage (e.g. for saving model checkpoints, etc.)
2. Setup GPU drivers on the agent host VMs with GPUs
3. Create a set of pods for distributed TensorFlow
4. Run Distributed TensorFlow training job

Detailed instructions at https://github.com/joyq-github/TensorFlowonK8s

Running Distributed TensorFlow on Kubernetes *(continued)*

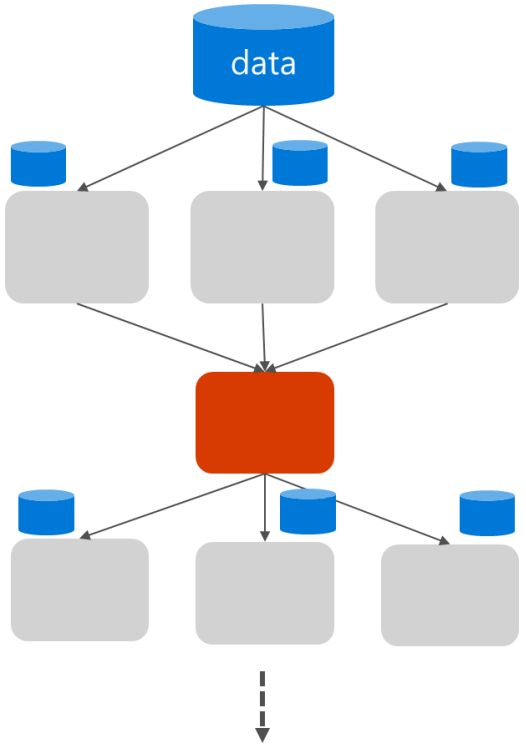
- Sample YAML for a TensorFlow worker pod with GPUs
- Check to make sure your K8s has your GPU resources data.

$kubectl describe nodes

```
Capacity:
 alpha.kubernetes.io/nvidia-gpu:        4
 cpu:                                   24
 memory:                                231108308Ki
 pods:                                  110
Allocatable:
 alpha.kubernetes.io/nvidia-gpu:        4
 cpu:                                   24
 memory:                                231005908Ki
 pods:                                  110
```

Distributed Deep Learning

Distributed Training Architecture

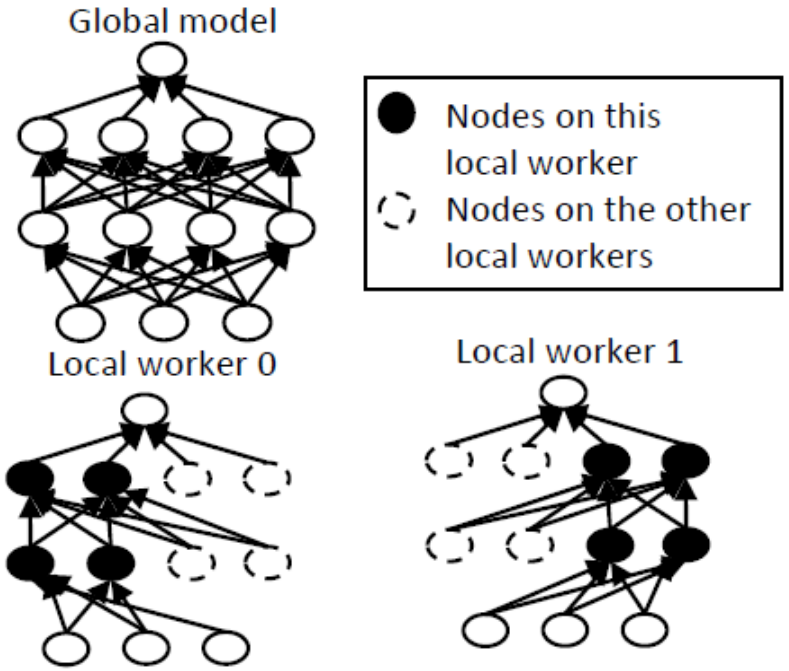


Data Parallelism

- 1. Parallel training on different machines
- 2. Update the parameter server synchronously/asynchronously
- 3. Refresh the local model with new parameters, go to 1 and repeat

Credits: Taifeng Wang, DMTK team

Distributed Training Architecture



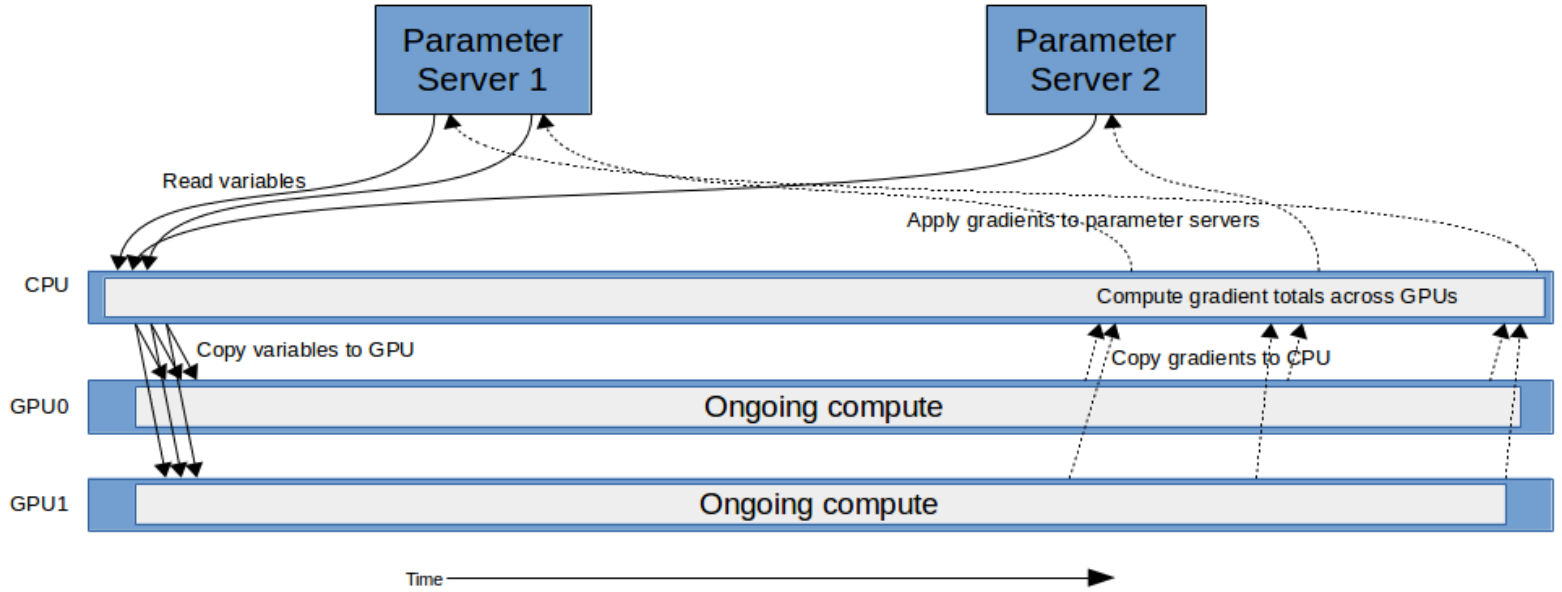
Model Parallelism

1. The global model is partitioned into K sub-models.
2. The sub-models are distributed over K local workers and serve as their local models.
3. In each mini-batch, the local workers compute the gradients of the local weights by back propagation.

Credits: Taifeng Wang, DMTK team

Distributed TensorFlow Architecture

- **For Variable Distribution & Gradient Aggregation**
- Parameter_server



Single worker's view of variable reads and updates in parameter_server mode, with three variables.

Source: https://www.tensorflow.org/performance/performance_models

Distributed Training Performance on Kubernetes

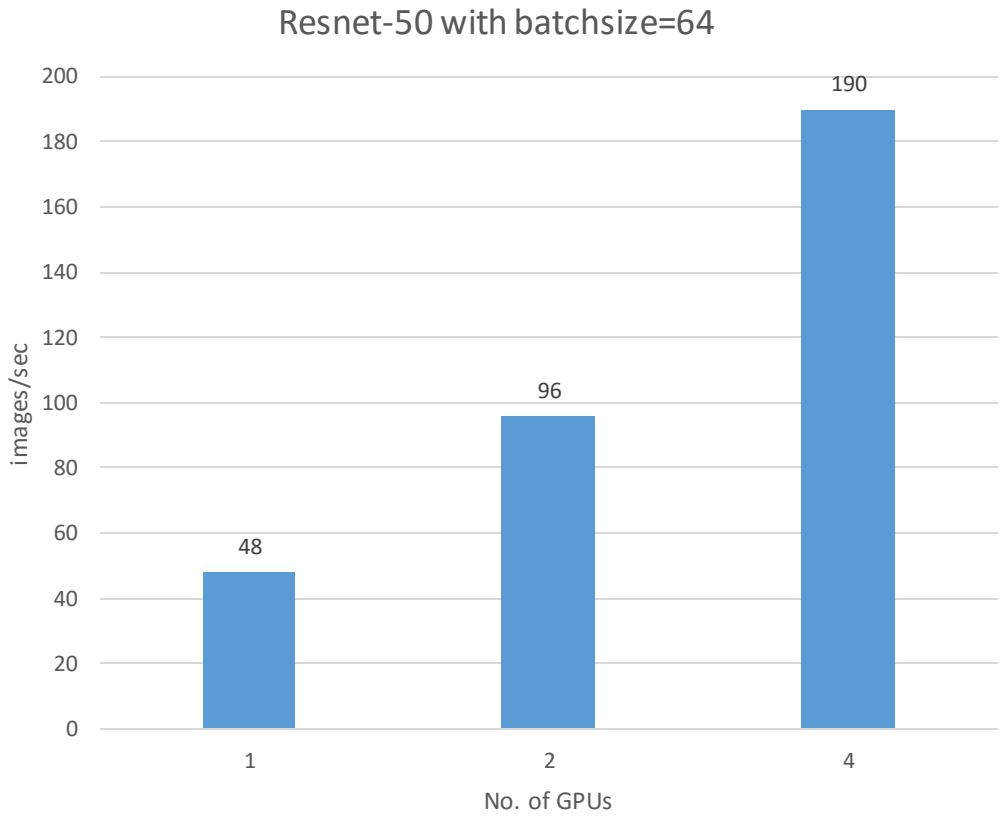
Training Environment on Azure

- **VM SKU**
  - **NC24r** for workers
    - 4x NVIDIA® Tesla® K80 GPU
    - 24 CPU cores, 224 GB RAM
  - **D14_v2** for parameter server
    - 16 CPU cores, 112 GB RAM
- **Kubernetes:** 1.6.6 (created using ACS-Engine)
- **GPU:** NVIDIA® Tesla® K80
- **Benchmarks scripts:** https://github.com/tensorflow/benchmarks/tree/master/scripts/tf_cnn_benchmarks

- **OS:** Ubuntu 16.04 LTS
- **TensorFlow**: 1.2
- **CUDA / cuDNN**: 8.0 / 6.0
- **Disk**: Local SSD
- **DataSet**: ImageNet (real data, not synthetic)

Training on Single node, Multi-GPU

- Linear scalability
- GPUs are fully saturated



- variable_update mode: parameter_server
- local_parameter_device: cpu

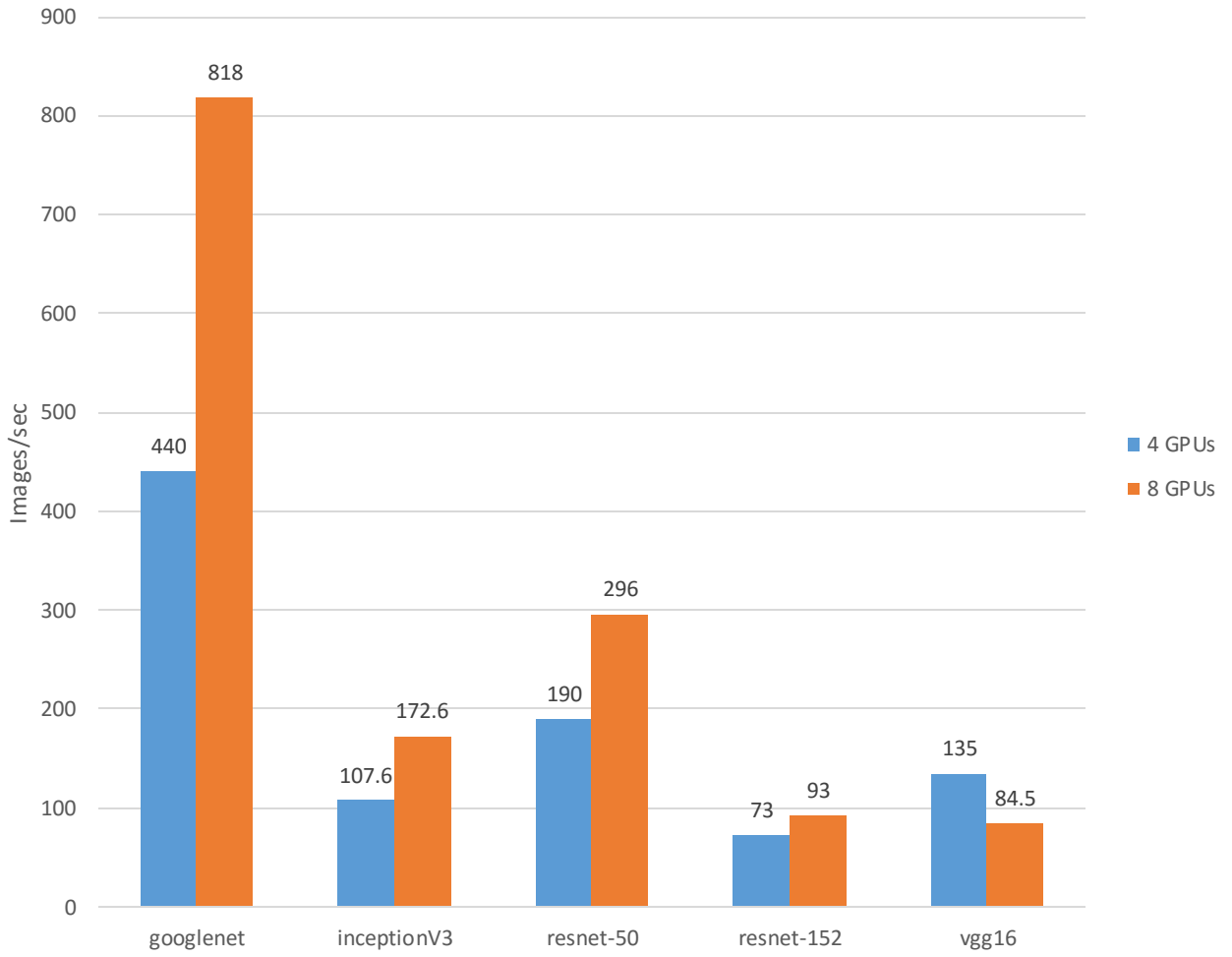
Distributed Training

Settings:

- Topology: 1 ps and 2 workers
- Async variables update
- Using cpu as the local_parameter_device
- Each ps/worker pod has its own dedicated host
- variable_update mode: parameter_server
- Network protocol: gPRC

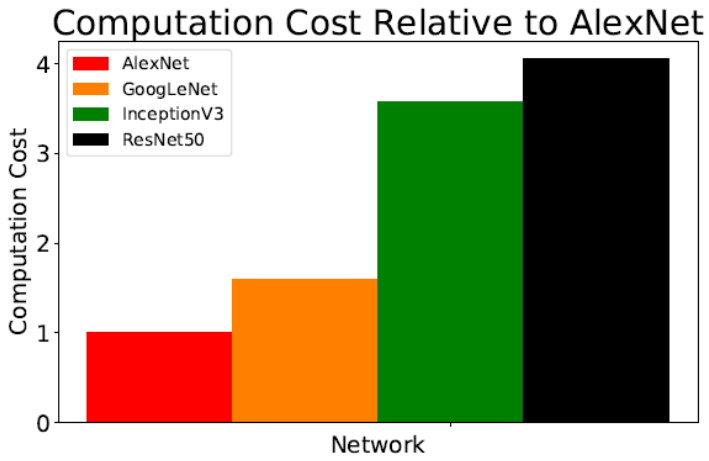
Distributed Training (*continued*)

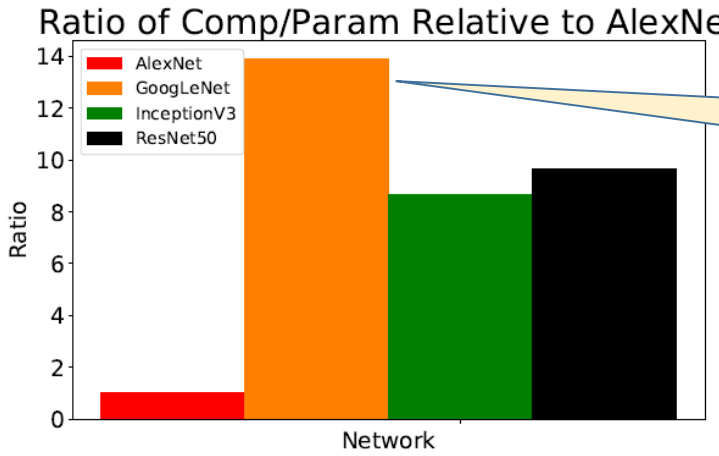
Single-node Training with 4 GPUs
vs Distributed Training with 2 workers with 8 GPUs in total



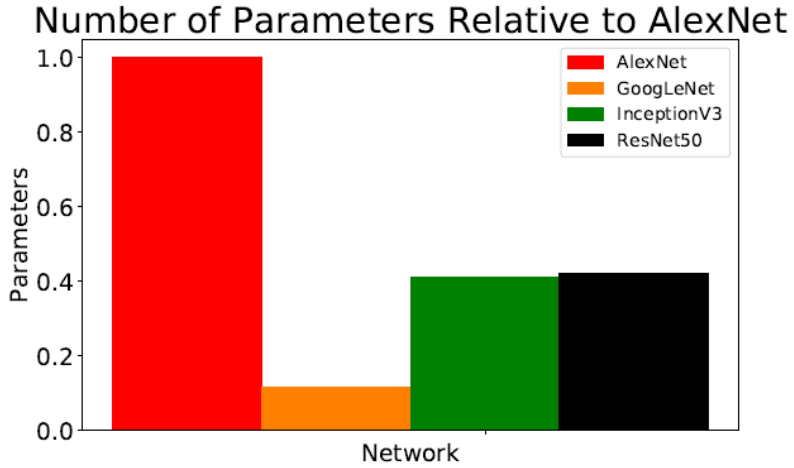
Distributed Training (*continued*)

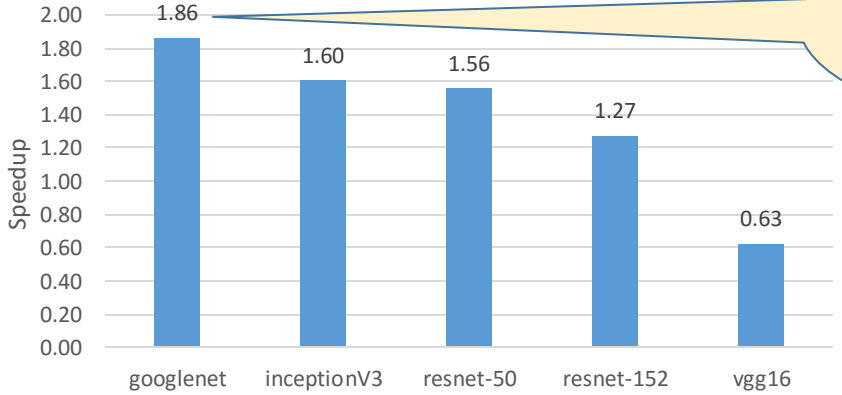
Distributed training scalability depends on the **compute/bandwidth ratio of the model**





The model with a higher ratio scales better.





GoogLeNet scales pretty well. VGG16 is suboptimal, due to its large size

Source: https://arxiv.org/abs/1704.04560

Distributed Training (*continued*)

Observations during test:

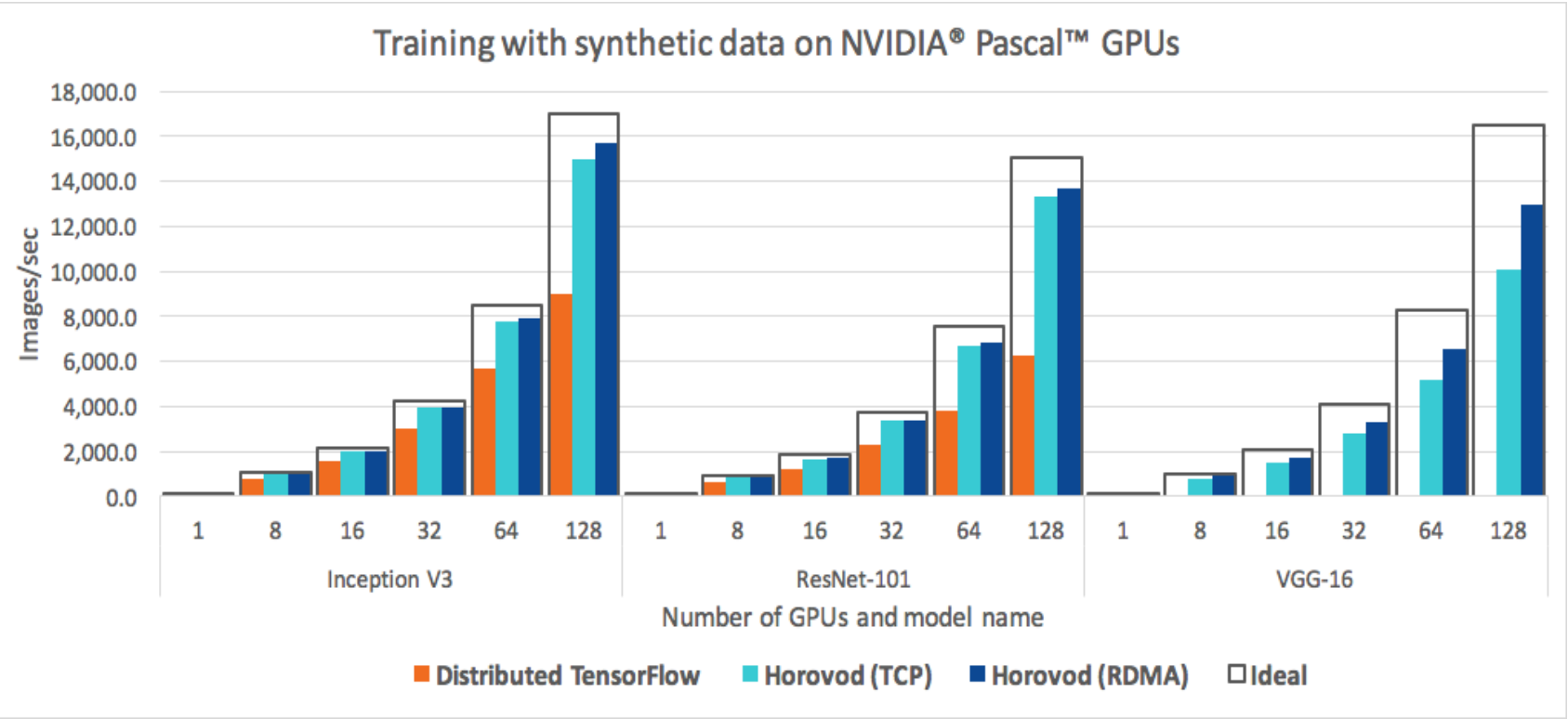
- Linear scalability largely depends on the model and network bandwidth.
- GPUs not fully saturated on the worker nodes, likely due to network bottleneck.
- VGG16 had suboptimal performance than single-node training. GPUs “starved” most of the time.
- Running directly on Host VMs rather than K8s pods did not make a huge difference, in this particular test environment.
- Having ps servers running on the same pods as the workers seem to have worse performance
- Tricky to decide the right ratio of workers to parameter servers
- Sync vs Async variable updates

Distributed Training (*continued*)

How can we do better?

Horovod: Uber's Open Source Distributed Deep Learning Framework for TensorFlow

Benchmark on 32 servers with 4 Pascal GPUs each connected by RoCE-capable 25 Gbit/s network
(source: https://github.com/uber/horovod)



- A stand-alone python package
- Seamless install on top of TensorFlow
- Uses NCCL for ring-allreduce across servers instead of parameter server
- Uses MPI for worker discovery and reduction coordination
- Tensor Fusion

Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training

Source: https://openreview.net/forum?id=SkhQHMW0W&noteId=SkhQHMW0W

Paper Summary:

- 99.9% of the gradient exchange in distributed SGD is redundant
- Propose Deep Gradient Compression (DGC) to greatly reduce the communication bandwidth
- DGC achieves a gradient compression ratio from 270x to 600x without losing accuracy, cutting the gradient size of
  - ResNet-50 from 97MB to 0.35MB
  - DeepSpeech from 488MB to 0.74MB
- Enables large-scale distributed training on inexpensive commodity 1Gbps Ethernet and facilitates distributed training on mobile.

Deep Learning Workspace by Microsoft Research Powered by Kubernetes

- Alpha release available at https://github.com/microsoft/DLWorkspace/
  Documentation at https://microsoft.github.io/DLWorkspace/

- Note that DL Workspace is NOT a MS product/service.
  It's an open source toolkit, and we welcome contribution!

Deep Learning Workspace by Microsoft Research Powered by Kubernetes

- **“FreeFlow” CNI plugin** from Microsoft Research
  - Leverage shared memory and RDMA to improve network performance
  - Higher throughput, lower latency, and less CPU overhead
  - Transparent to the containers & the apps
  - Deployed as DaemonSet

- **Custom CRI & Scheduler: GPU-related resource scheduling on K8s (Credits: Sanjeev Mehrotra from MS Research)**
  - Pods with no. of GPUs with how much memory
  - Pods with no. of GPUs interconnected via NVLink, etc.
  - Eventually may go into the device plugins

Resources

- **Getting Started with Kubernetes on Azure**
  https://github.com/Azure/acs-engine
  https://docs.microsoft.com/en-us/azure/container-service/kubernetes/

- **Running Distributed TensorFlow on Kubernetes using ACS/ACS-Engine**
  https://github.com/joyq-github/TensorFlowonK8s

Resources (*continued*)

- **Deep Learning Workspace powered by Kubernetes**
  https://github.com/microsoft/DLWorkspace/
  https://microsoft.github.io/DLWorkspace/

- **TensorFlow resources**
  https://www.tensorflow.org/performance/
  https://eng.uber.com/horovod/
  https://arxiv.org/abs/1704.04560

- **FreeFlow: High Performance Container Networking**
  https://www.microsoft.com/en-us/research/publication/freeflow-high-performance-container-networking-3/