MesosCon
NORTH AMERICA

Distributed Deep Learning on Mesos with GPUs and Gang Scheduling

Min Cai, Alex Sergeev, Paul Mikesell, Anne Holler, *UBER*

THE LINUX FOUNDATION

Who are we?

Min Cai

Alex Sergeev

Paul Mikesell

Anne Holler

UBER

THE LINUX FOUNDATION

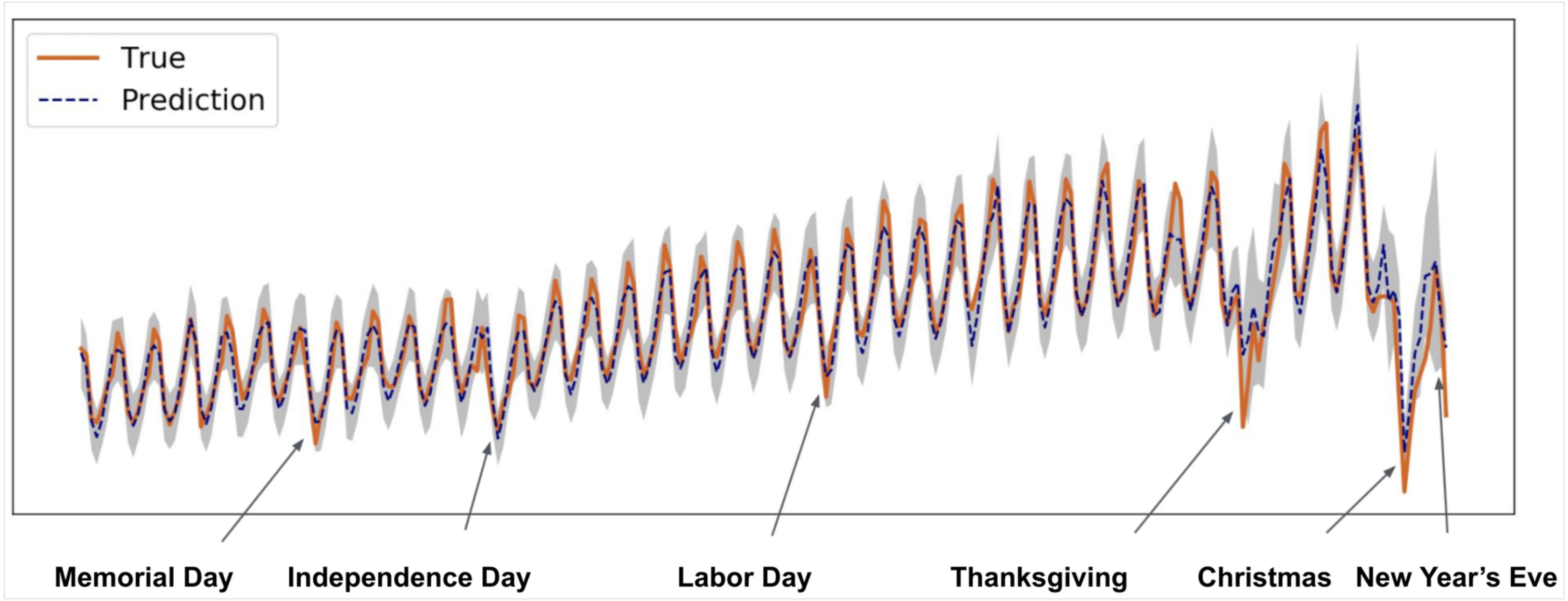
Deep Learning @ UBER

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

Self-Driving Vehicles

Self-Driving Vehicles

Trip Forecasting



Fraud Detection

Spam referral code

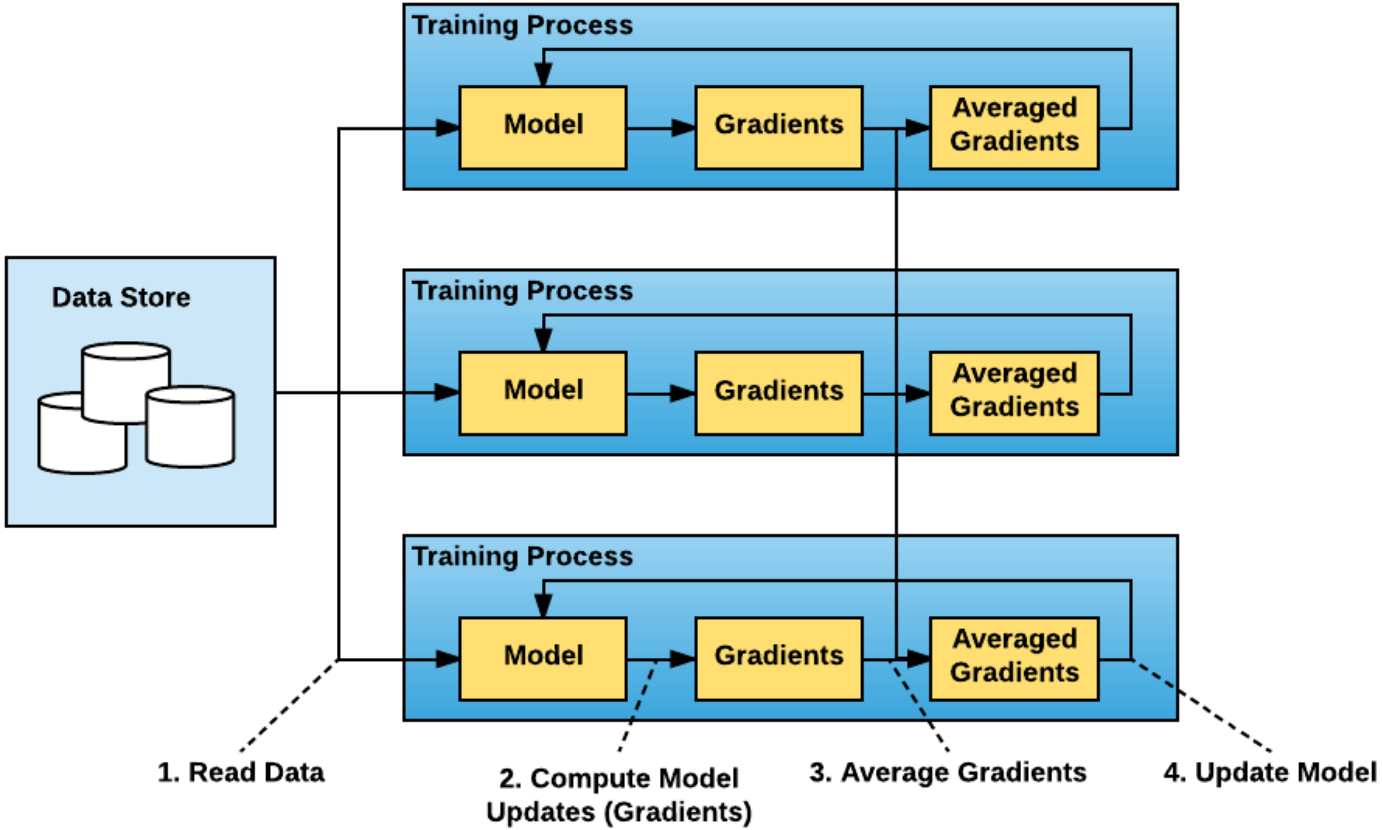
Partner up with the same driver

Cash out Uber credits

Why Distributed Deep Learning?

- Speed up model training
- Scale out to hundreds of GPUs
- Shard large models that can not fit into a single machine

How Distributed Deep Learning Works

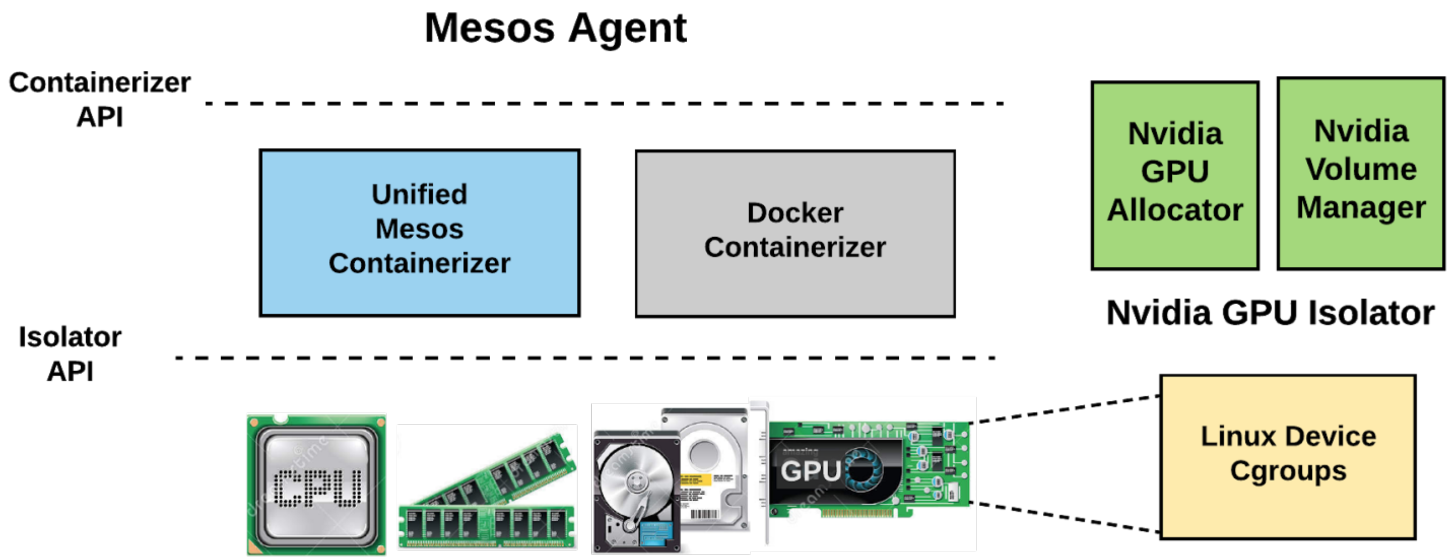


Why Mesos?

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

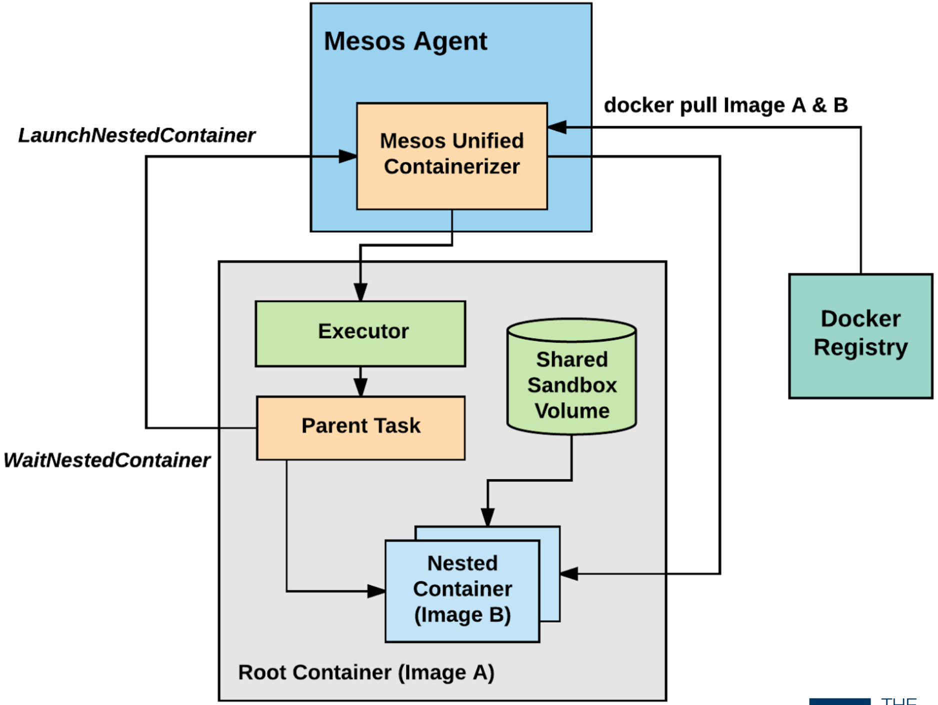
Mesos Support for GPUs

- 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



THE LINUX FOUNDATION

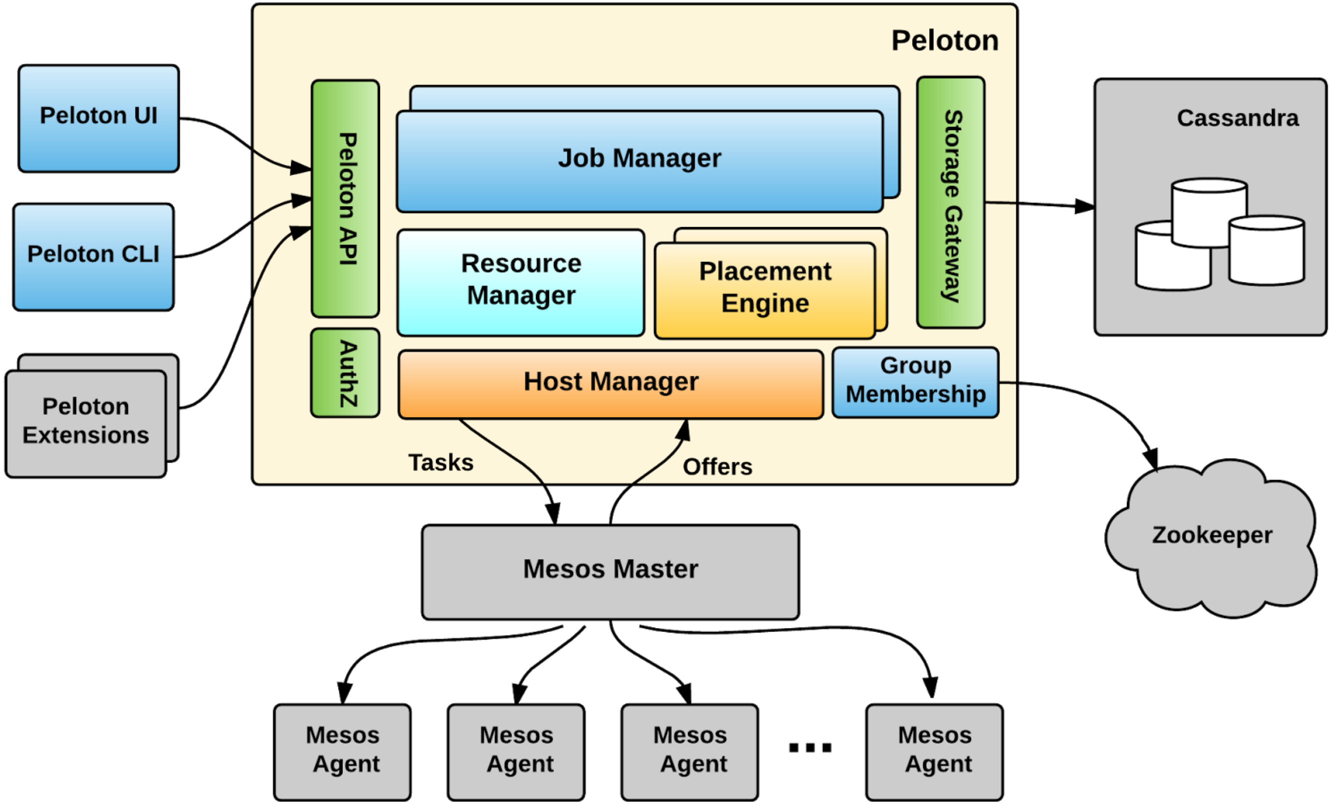
What is Missing?

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

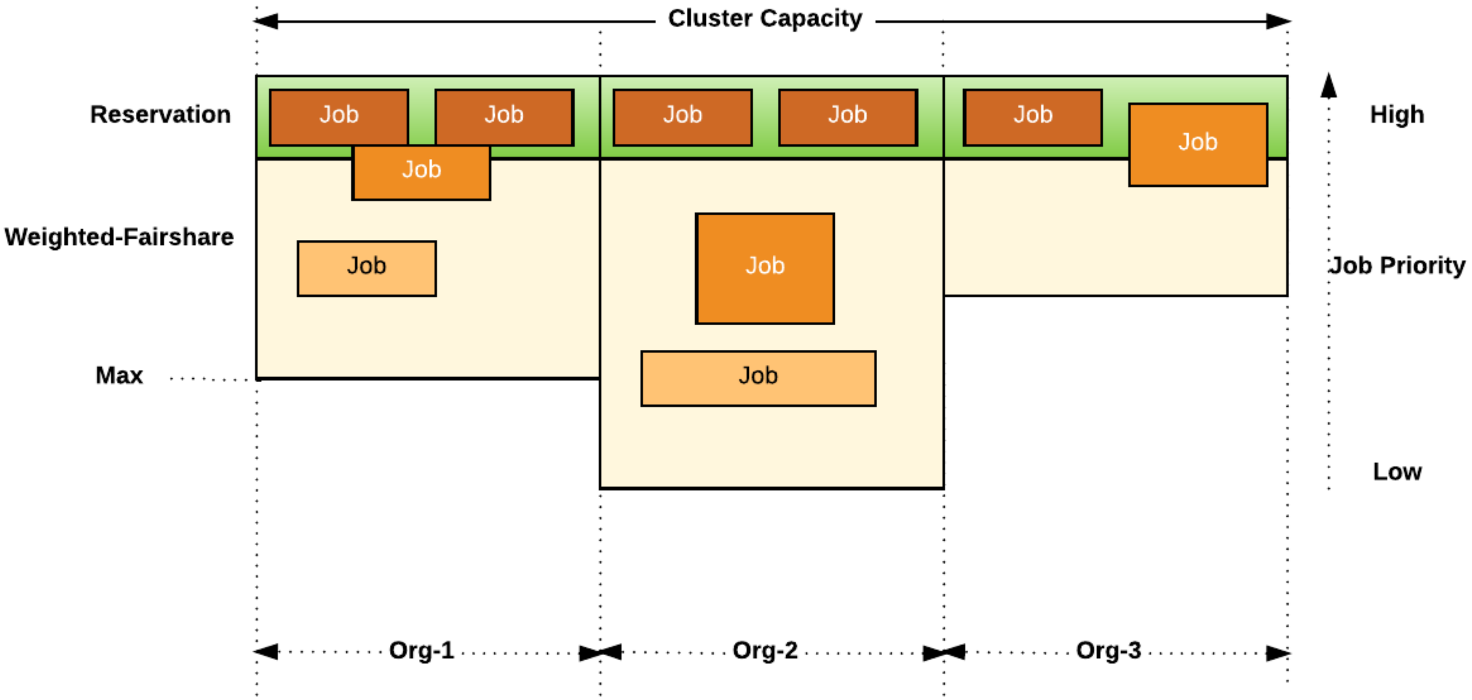
Peloton Overview

	Borg	YARN	Kubernetes	Peloton
Application Workflow (e.g Map-Reduce)	Controller Job	Application Master		Controller Job
Job/Task Lifecycle	Borgmaster		kube-controller	**Peloton**
Task Placement		Resource Manager	kube-scheduler	
Task Preemption				
Resource Allocation				Mesos Master
Task Execution	Borglet	Node Manager	kubelet	Mesos Agent

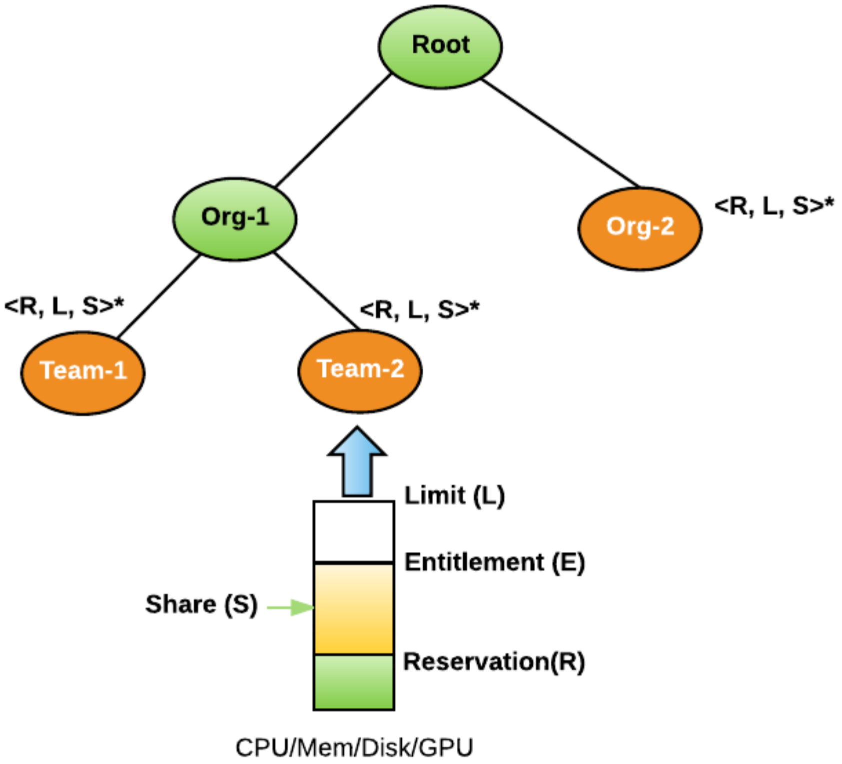
Peloton Architecture



Elastic GPU Resource Management



Resource Pools



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

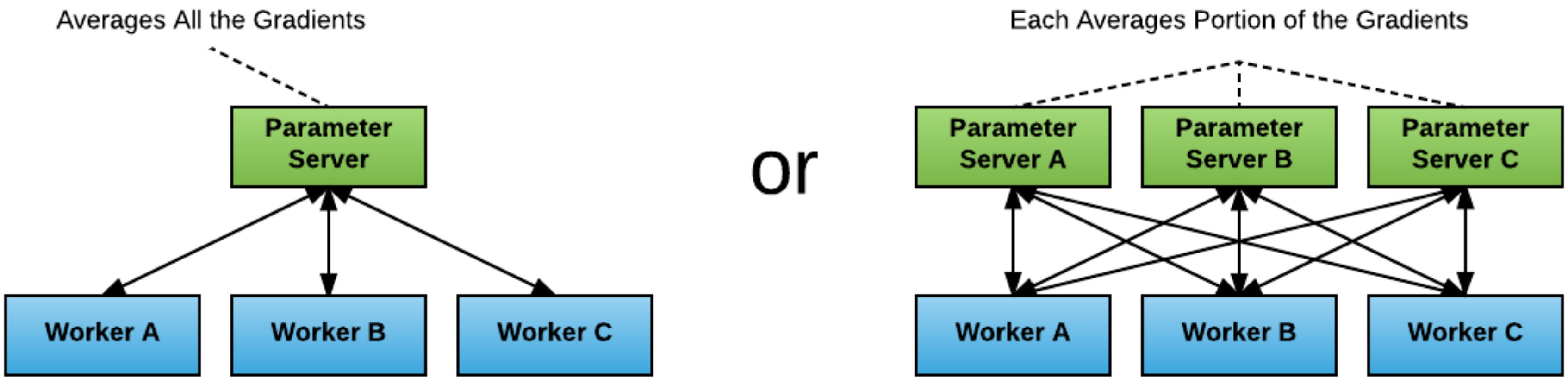
Placement Strategies

- 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

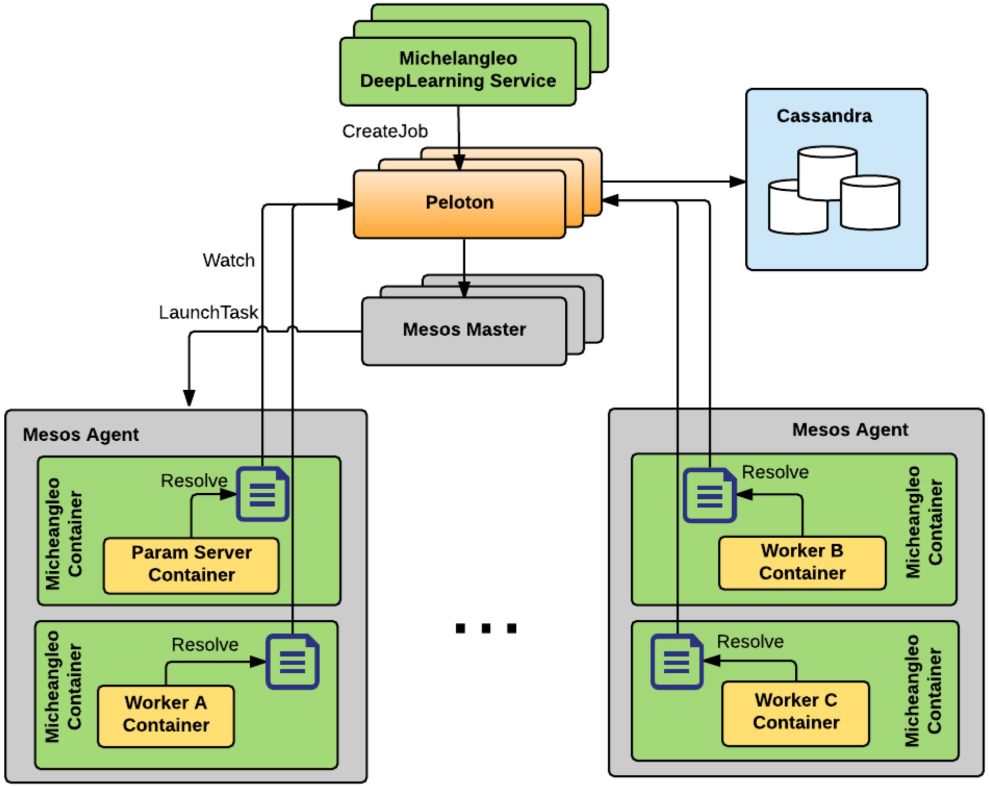
Why TensorFlow?

- 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

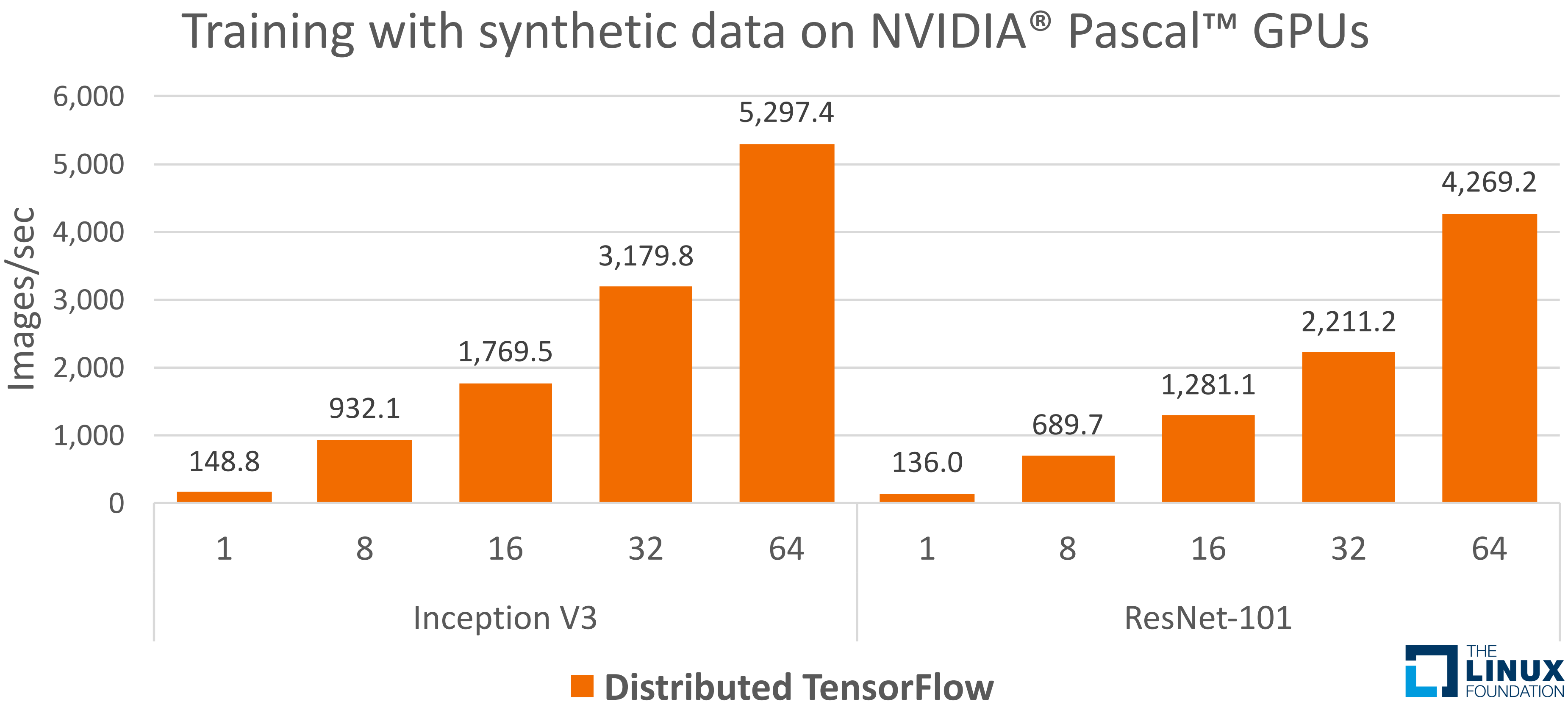
Architecture for Distributed TensorFlow



Architecture for Distributed TensorFlow on Mesos

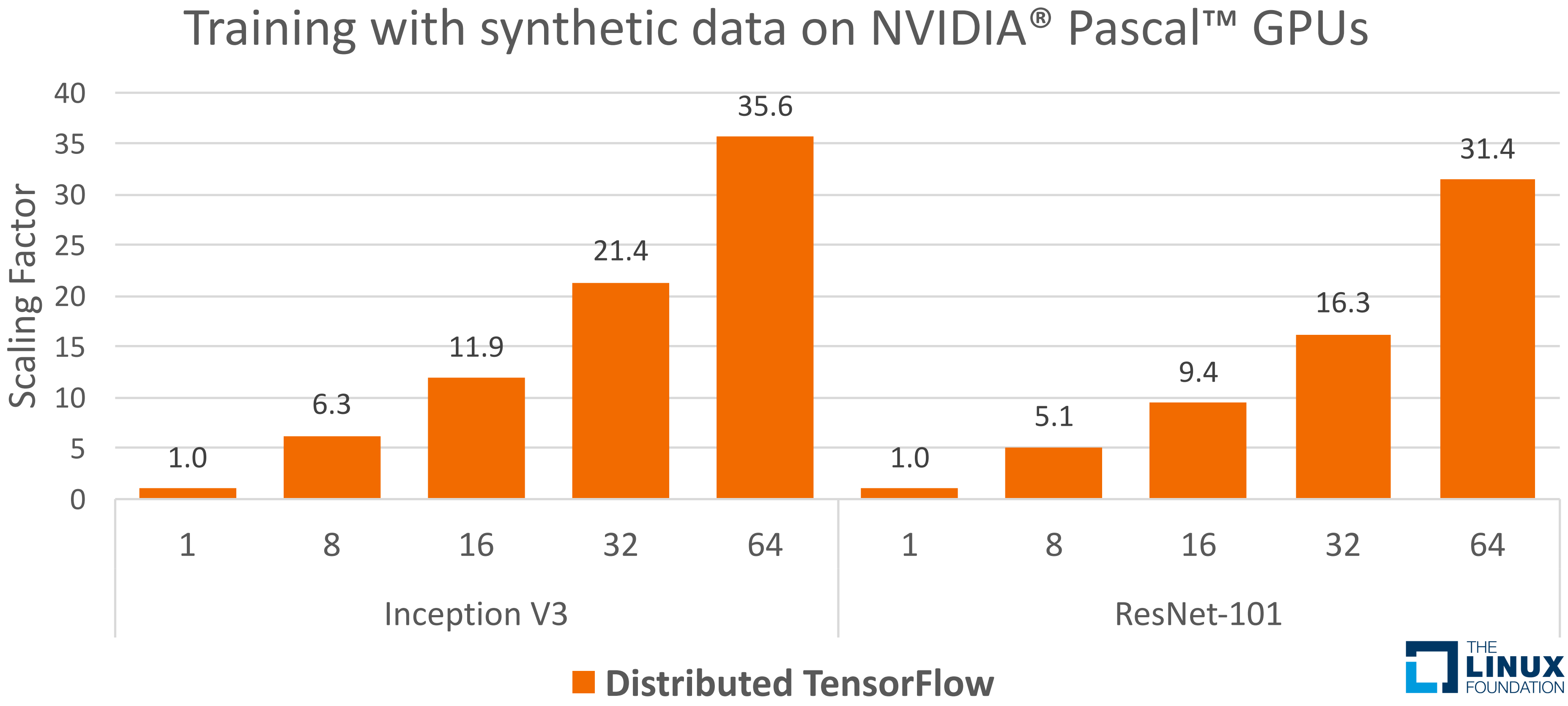


Distributed Training Performance



THE LINUX FOUNDATION

Distributed Training Performance



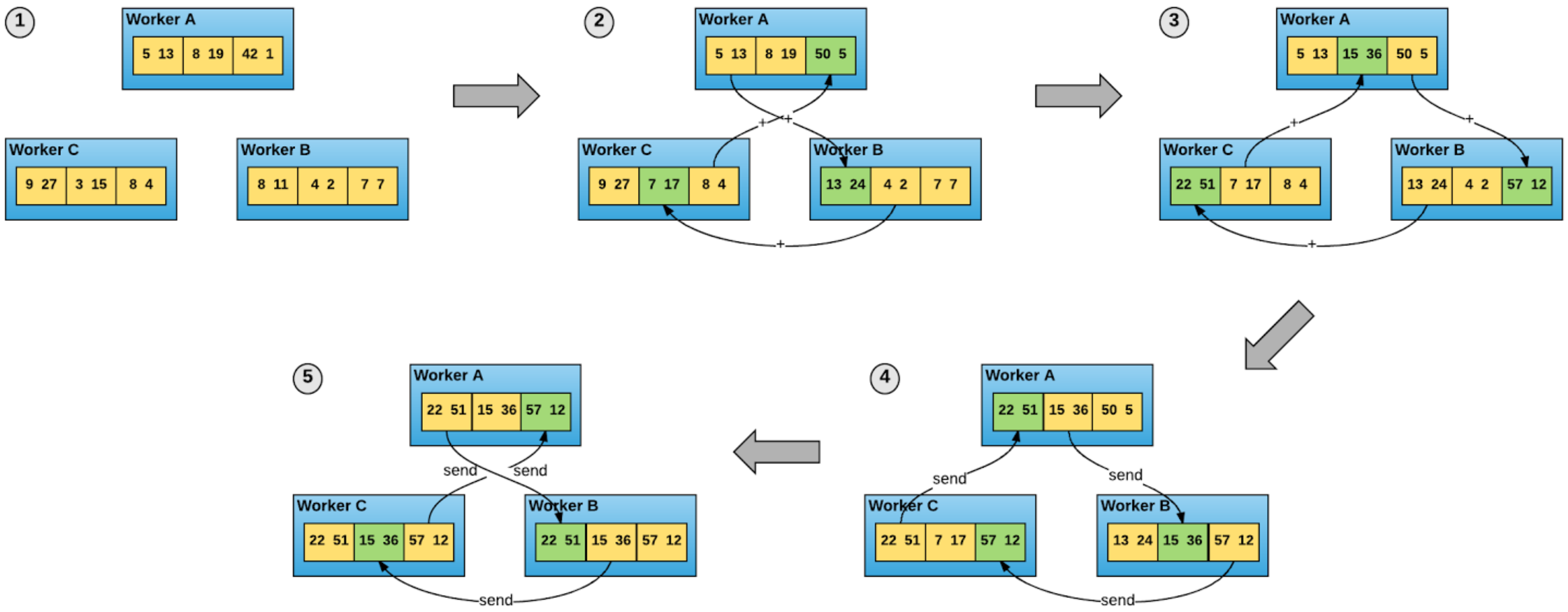
Can We Do Better?

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

Horovod

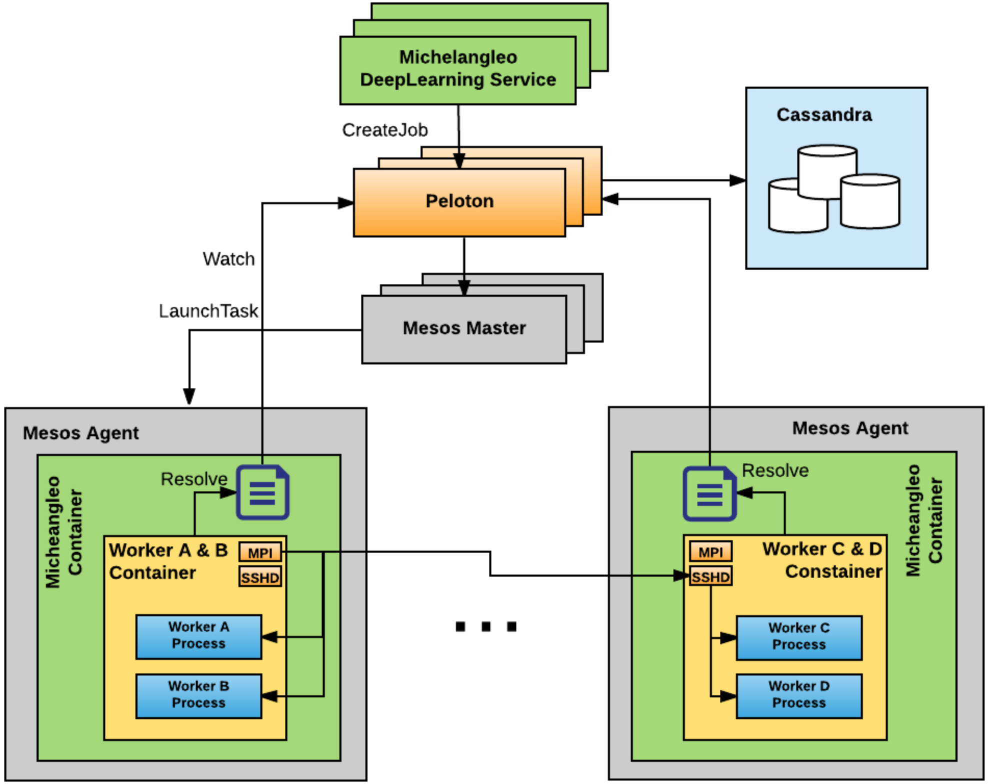
- Distributed training framework for TensorFlow
- Uses bandwidth-optimal communication protocols
  - Makes use of RDMA (RoCE, InfiniBand) if available
- Seamlessly installs on top of TensorFlow via **pip install horovod**

Architecture for Horovod

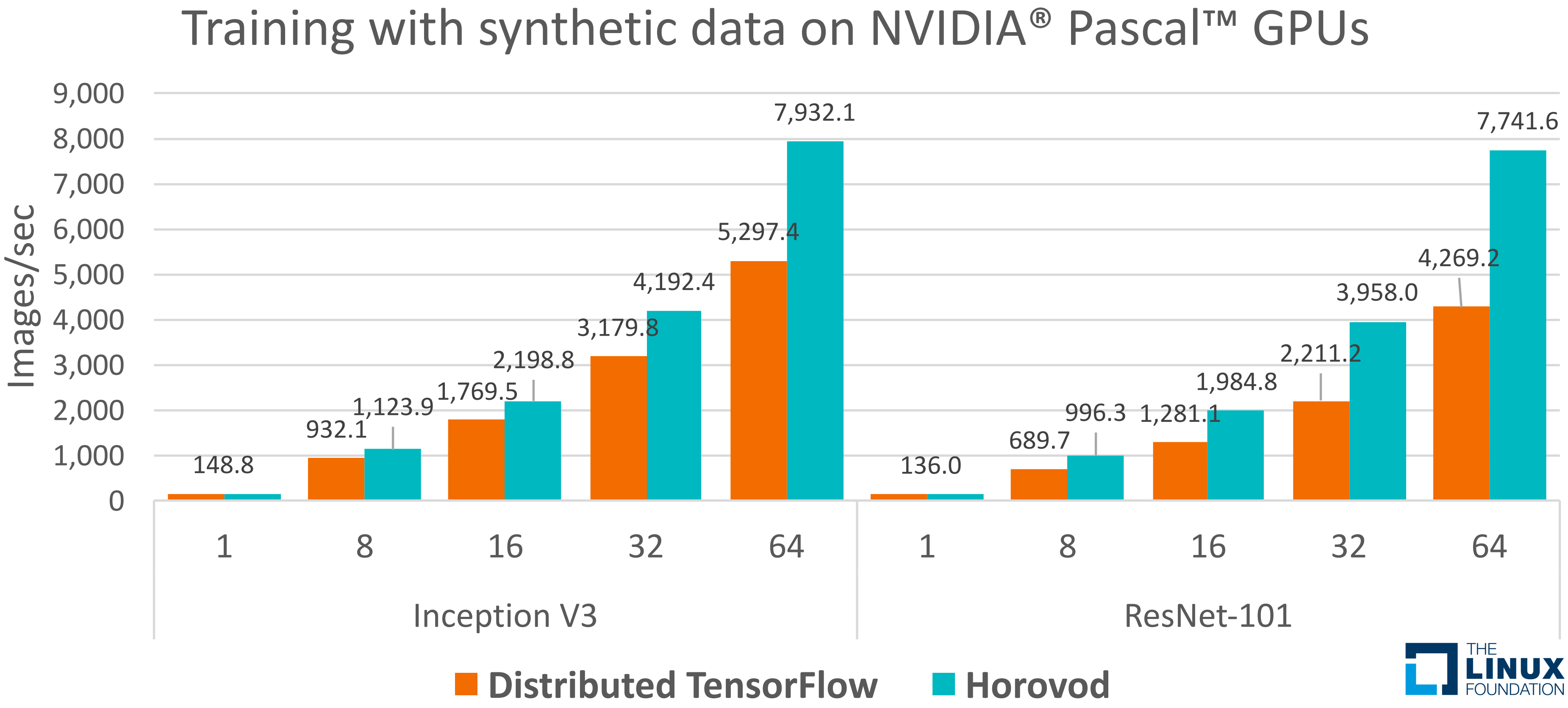


Patarasuk, P., & Yuan, X. (2009). Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2), 117-124. doi:10.1016/j.jpdc.2008.09.002

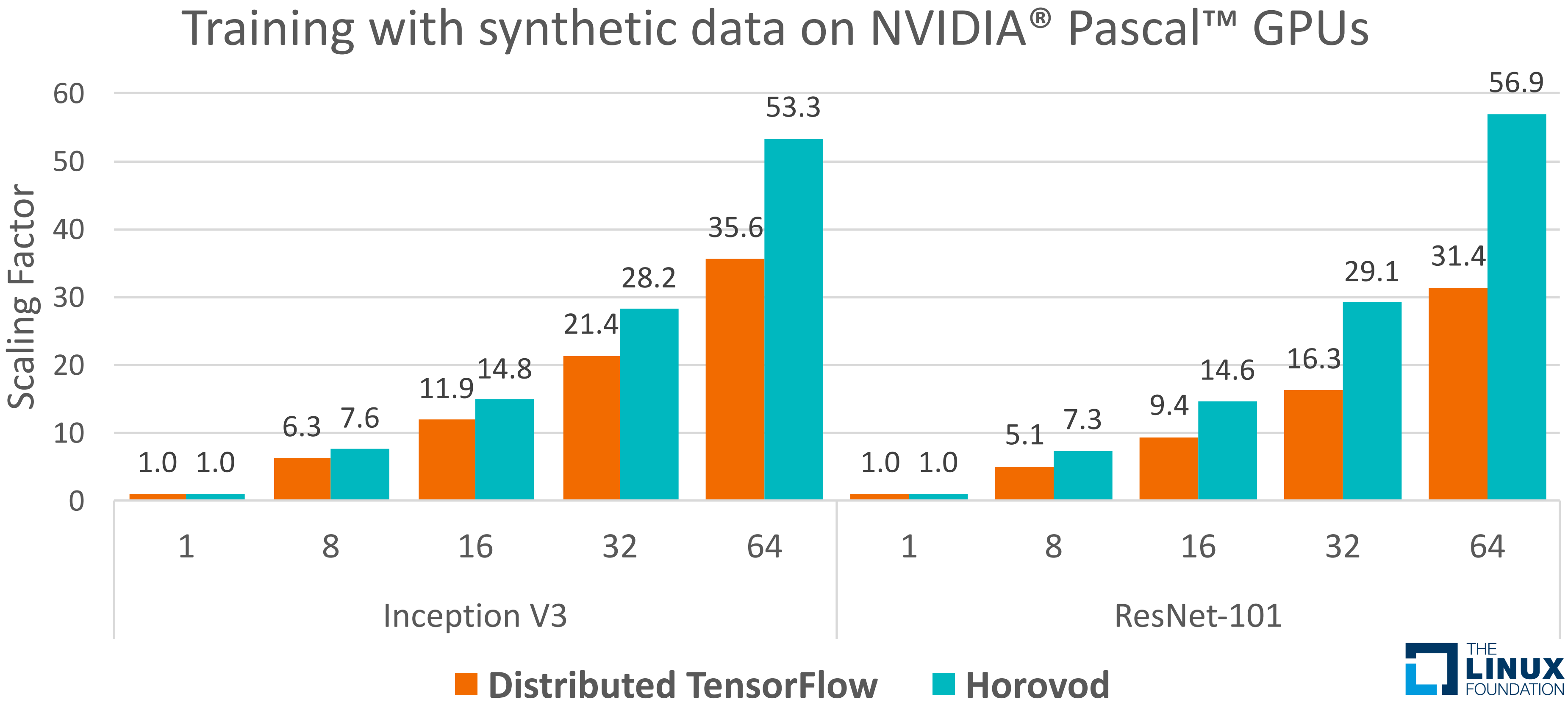
Architecture for Horovod on Mesos



Distributed Training Performance with Horovod



Distributed Training Performance with Horovod



THE LINUX FOUNDATION

What About Usability?

```
import argparse
import sys

import tensorflow as tf

FLAGS = None

def main(_):
  ps_hosts = FLAGS.ps_hosts.split(",")
  worker_hosts = FLAGS.worker_hosts.split(",")

  # Create a cluster from the parameter server and worker hosts.
  cluster = tf.train.ClusterSpec({"ps": ps_hosts, "worker": worker_hosts})

  # Create and start a server for the local task.
  server = tf.train.Server(cluster,
                           job_name=FLAGS.job_name,
                           task_index=FLAGS.task_index)

  if FLAGS.job_name == "ps":
    server.join()
  elif FLAGS.job_name == "worker":

    # Assigns ops to the local worker by default.
    with tf.device(tf.train.replica_device_setter(
        worker_device="/job:worker/task:%d" % FLAGS.task_index,
        cluster=cluster)):

      # Build model...
      loss = ...
      global_step = tf.contrib.framework.get_or_create_global_step()

      train_op = tf.train.AdagradOptimizer(0.01).minimize(
          loss, global_step=global_step)

    # The StopAtStepHook handles stopping after running given steps.
    hooks=[tf.train.StopAtStepHook(last_step=1000000)]

    # 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(master=server.target,
                                           is_chief=(FLAGS.task_index == 0),
                                           checkpoint_dir="/tmp/train_logs",
                                           hooks=hooks) as mon_sess:
      while not mon_sess.should_stop():
        # Run a training step asynchronously.
        # See `tf.train.SyncReplicasOptimizer` for additional details on how to
        # perform *synchronous* training.
        # mon_sess.run handles AbortedError in case of preempted PS.
        mon_sess.run(train_op)

if __name__ == "__main__":
  parser = argparse.ArgumentParser()
  parser.register("type", "bool", lambda v: v.lower() == "true")
  # Flags for defining the tf.train.ClusterSpec
  parser.add_argument(
      "--ps_hosts",
      type=str,
      default="",
      help="Comma-separated list of hostname:port pairs"
  )
  parser.add_argument(
      "--worker_hosts",
      type=str,
      default="",
      help="Comma-separated list of hostname:port pairs"
  )
  parser.add_argument(
      "--job_name",
      type=str,
      default="",
      help="One of 'ps', 'worker'"
  )
  # Flags for defining the tf.train.Server
  parser.add_argument(
      "--task_index",
      type=int,
      default=0,
      help="Index of task within the job"
  )
  FLAGS, unparsed = parser.parse_known_args()
```

```
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

Giving Back

Horovod is available on GitHub today

https://github.com/uber/horovod

Thank you!

Any questions?