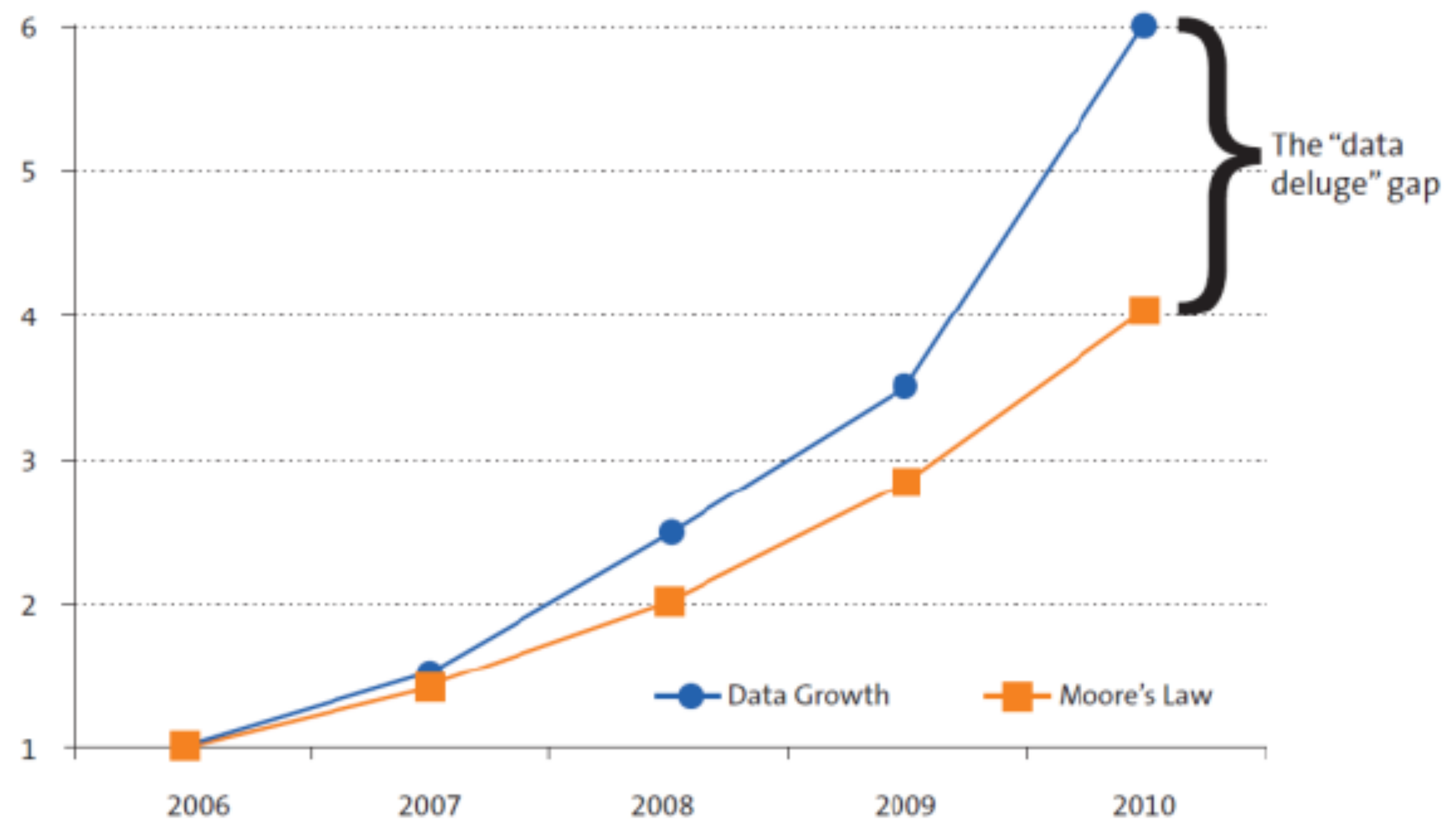
Distributed Tensorflow with Kubernetes

Jakob Karalus, @krallistic,

codecentric

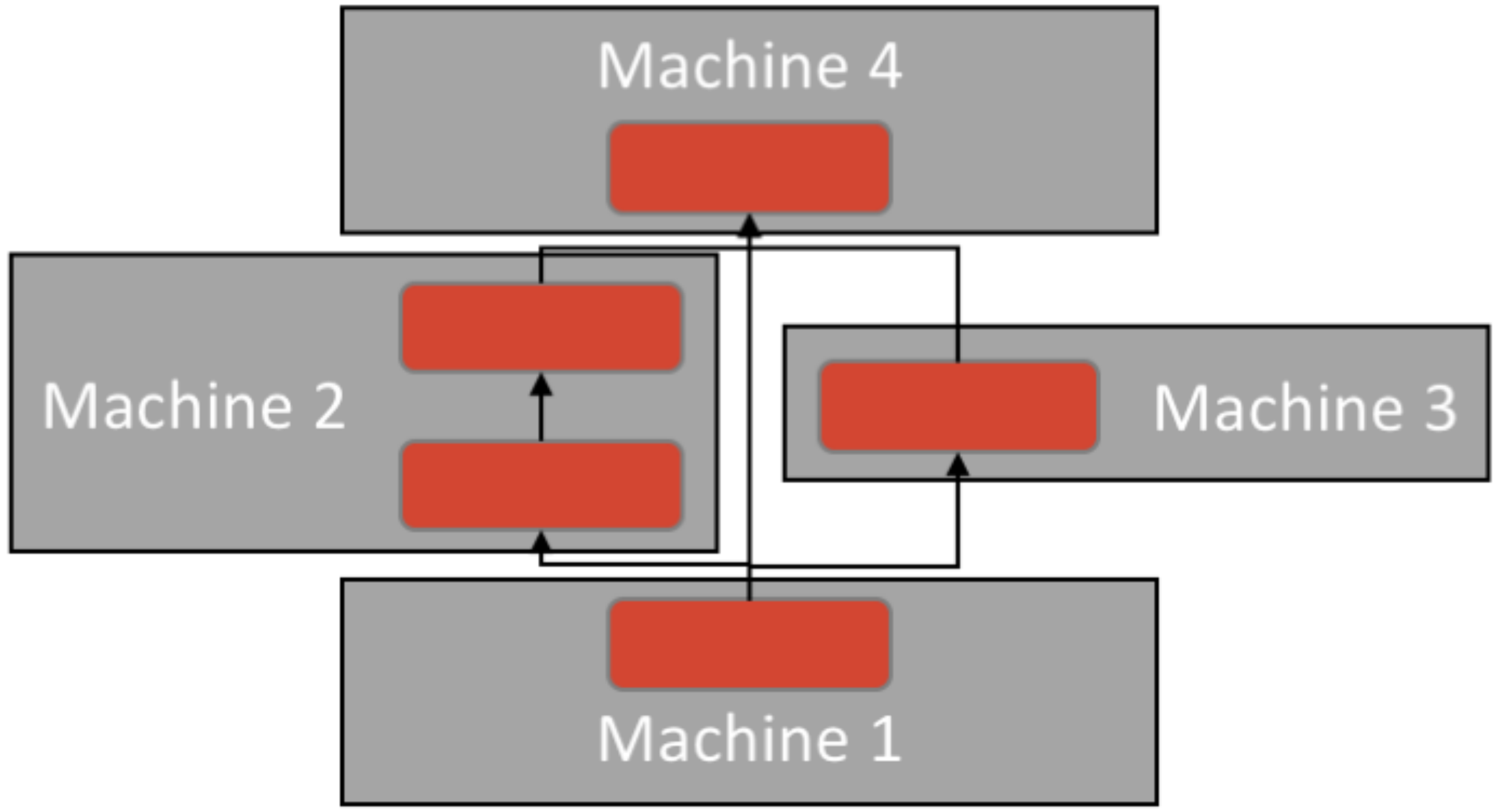
Training Neural Networks

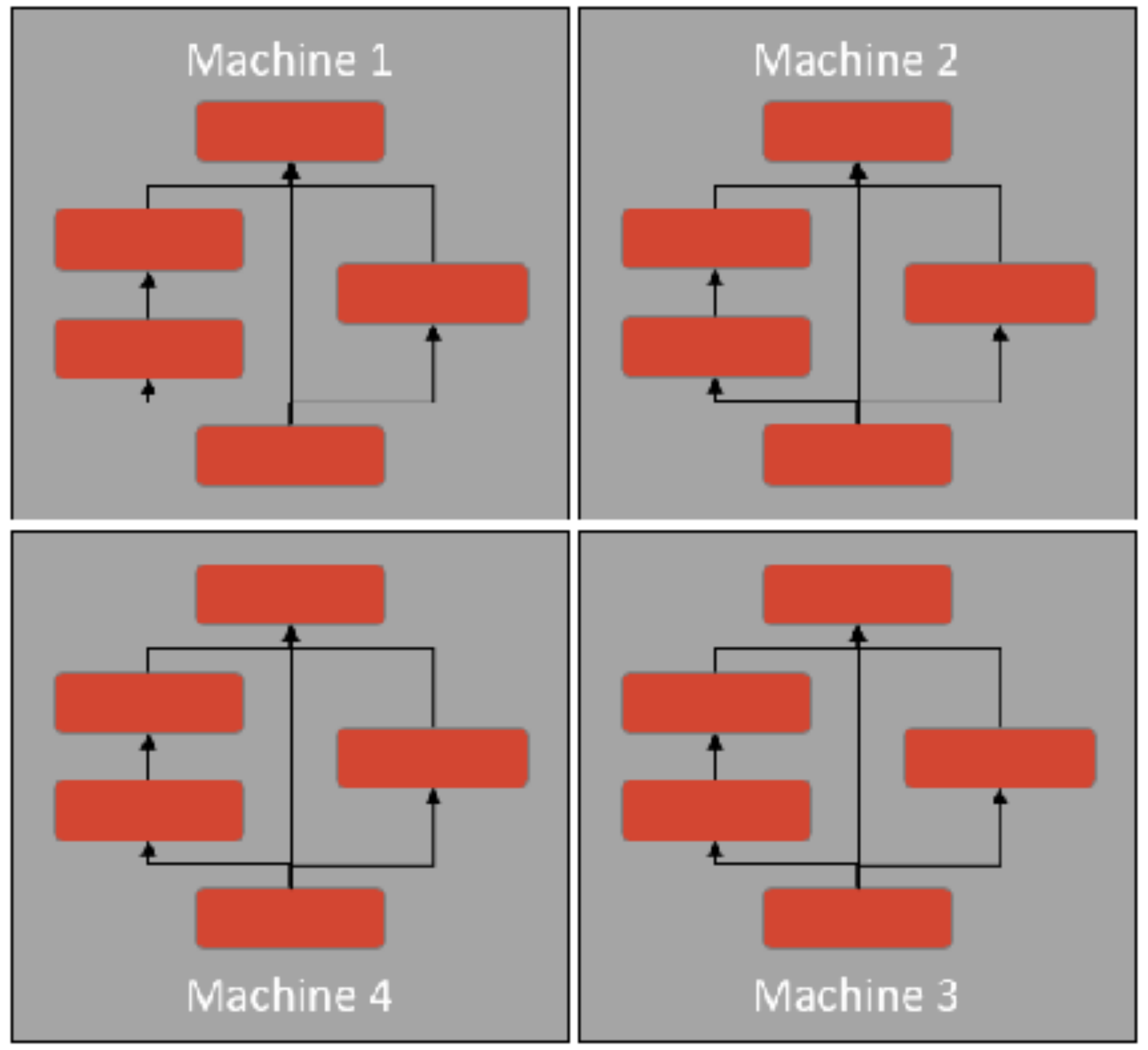
- First steps are quick and easy.
  - Single Node Neural Networks
- We want:
  - More Data!
  - Deeper Models!
  - Wider Model!
  - Higher Accuracy!
- (Single Node) Compute cant keep up
- Longer Trainingstime -> Longer Cycles -> Lower Productivity



Distributed & Parallel

- We need to distribute and train in parallel to be efficient
  - Data Parallelism
  - Model Parallelsim
  - Grid Search
  - Predict
  - -> Build in TF
- How can we deploy that to a cluster
  - Schedule TF onto Nodes
  - Service Discovery
  - GPUs
  - -> Kubernetes





Requirements & Content

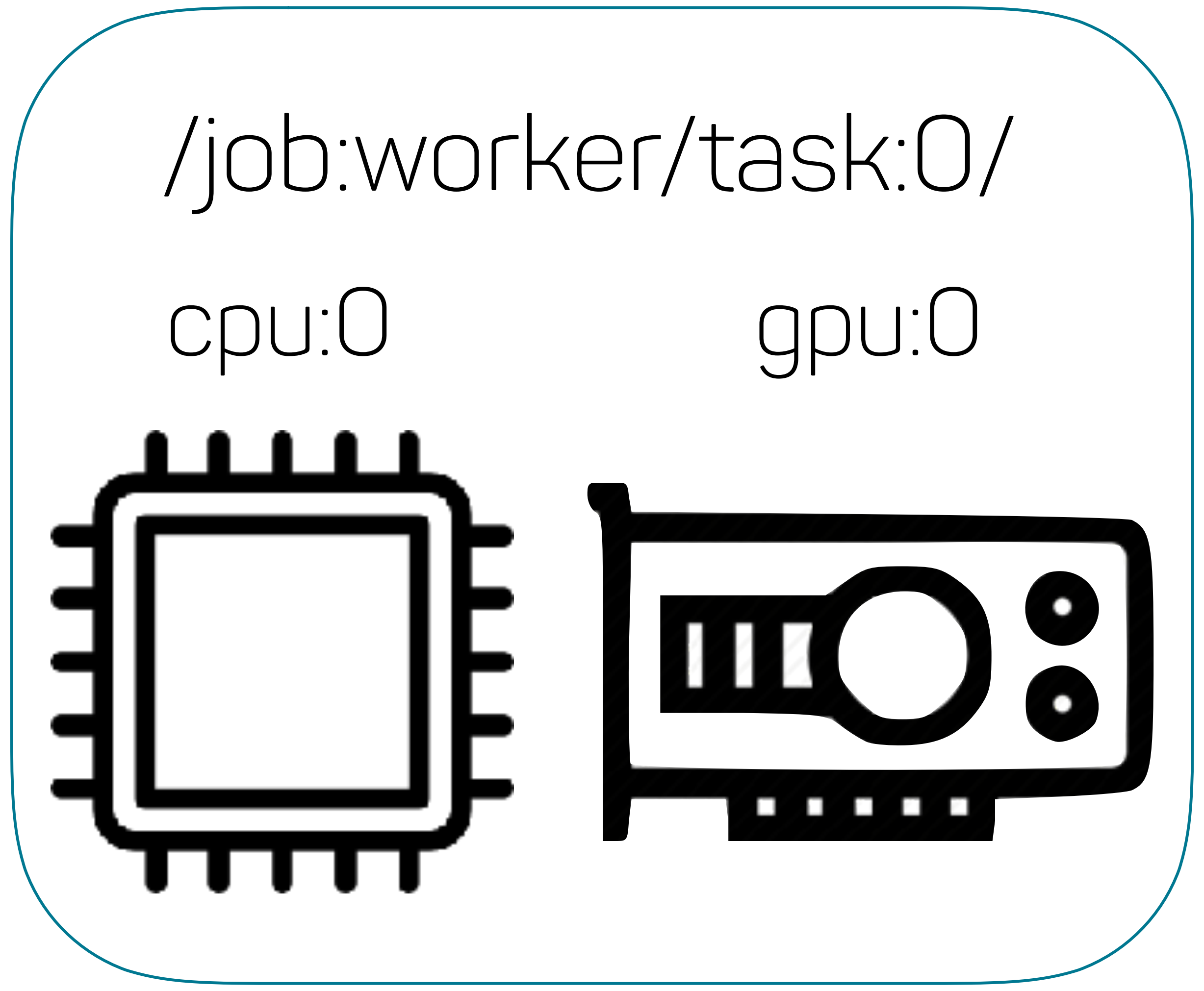
- Basic Knowledge of Neural Networks
- Knowledge of Tensorflow
- Basic Docker/Kubernetes knowledge
    - (Docker) Containers: Mini VM (Wrong!)
    - Kubernetes: Schedulurer/Orchestration Tool for Containers
- Only focus on the task of parallel and/or distributed training
    - We will not look at architectures like CNN/LTSM etc

Tensorflow on a single node

- Build your Graph
- Define which Parts on each device
  - TF places data
  - DMA for coordination/communication
- Define loss, accuracy etc
- Create Session for training
  - Feed Data into Session
  - Retrieve results

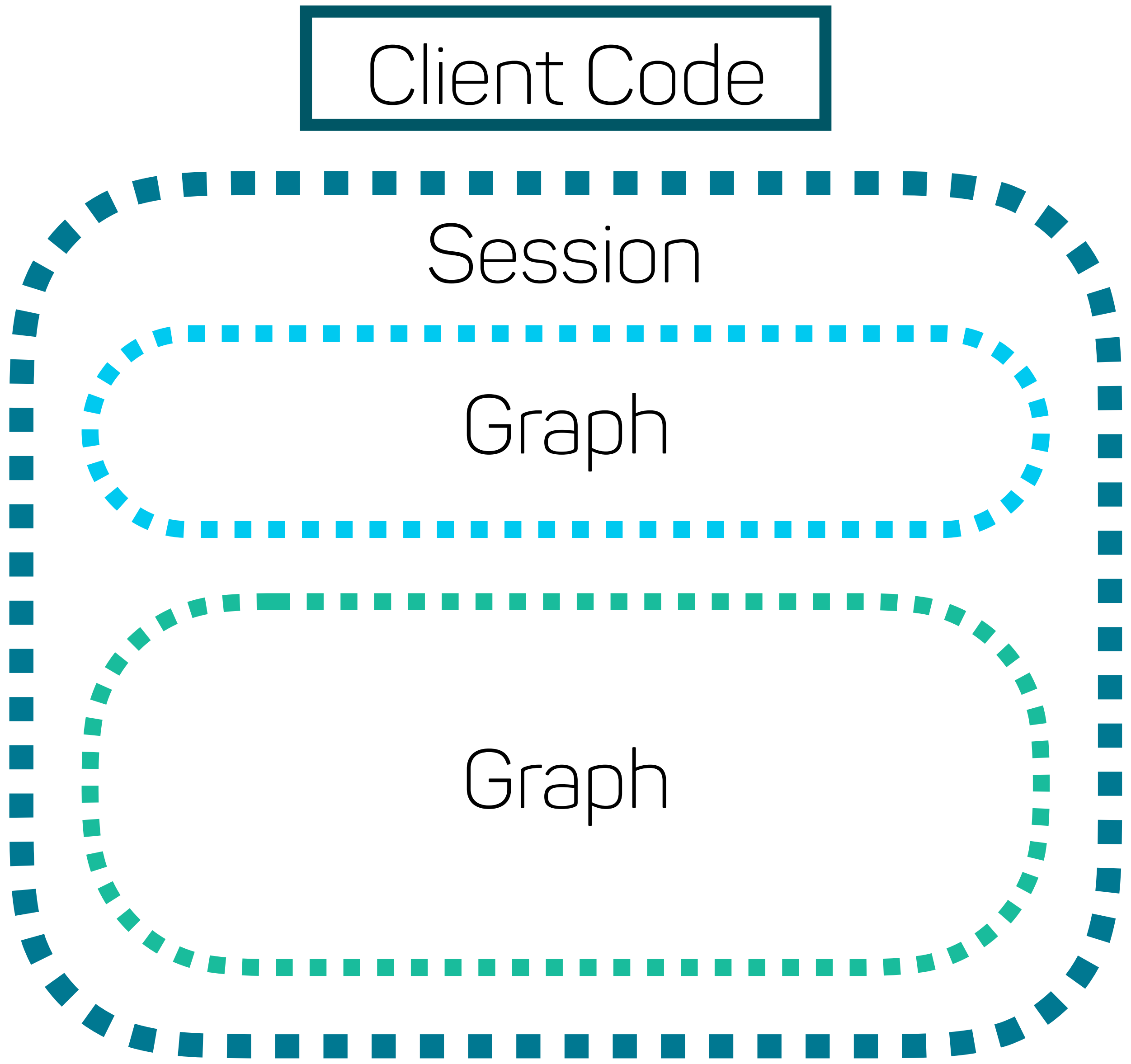
```
with tf.device("/cpu:0"):
    W = tf.Variable()
    b = tf.Variable()
with tf.device("/gpu:0"):
    y = tf.matmul(input, W) + b
    train_op = optimze_loss(y, y_)
```

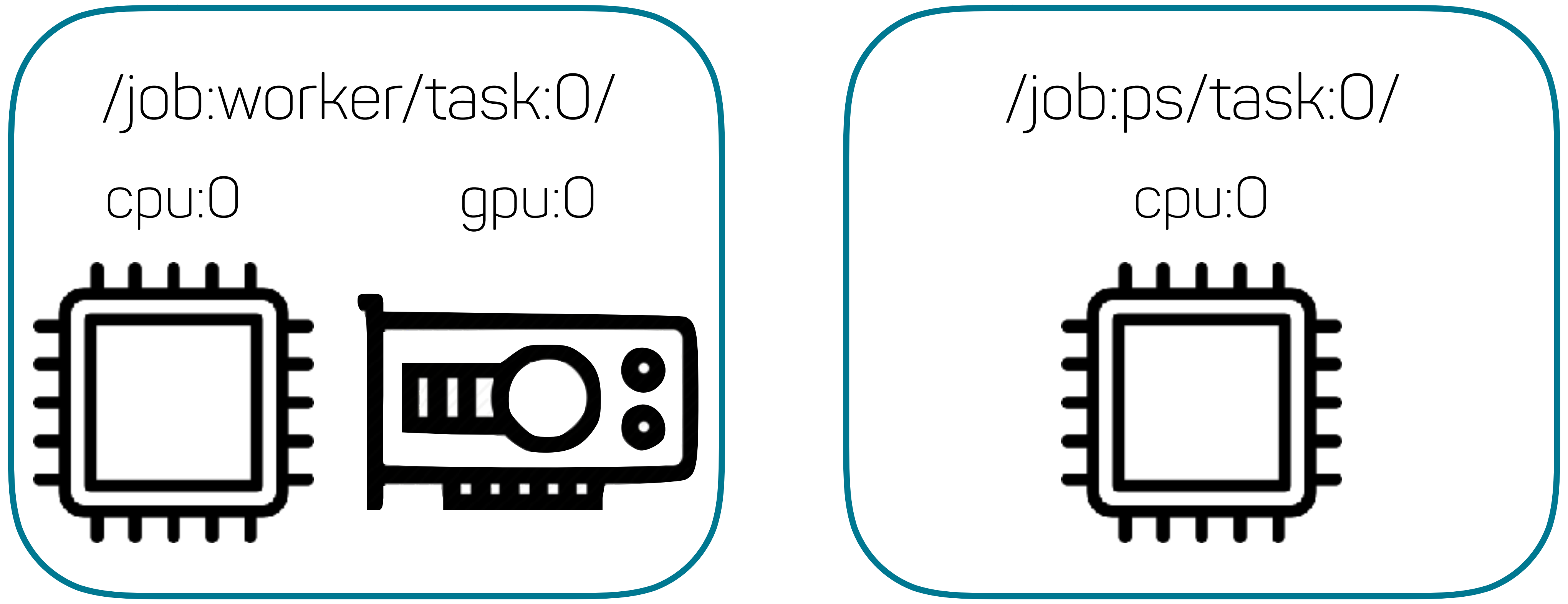
Client Code



Parameter Server & Worker Replicas

- Client: Code that builds the Graph, communicates with cluster, builds the session
- Cluster: Set of nodes which have jobs (roles)
- Jobs
  - Worker Replicas: compute intensive part
  - Parameter Servers(ps): Holds model state and reacts to updates
  - Each job can hold 0..* task
- Task
  - The actual server process
- Worker Task 0 is by default the chief worker
  - Responsible for checkpointing, initialising and health checking
- CPU 0 represents all CPUs on the Node



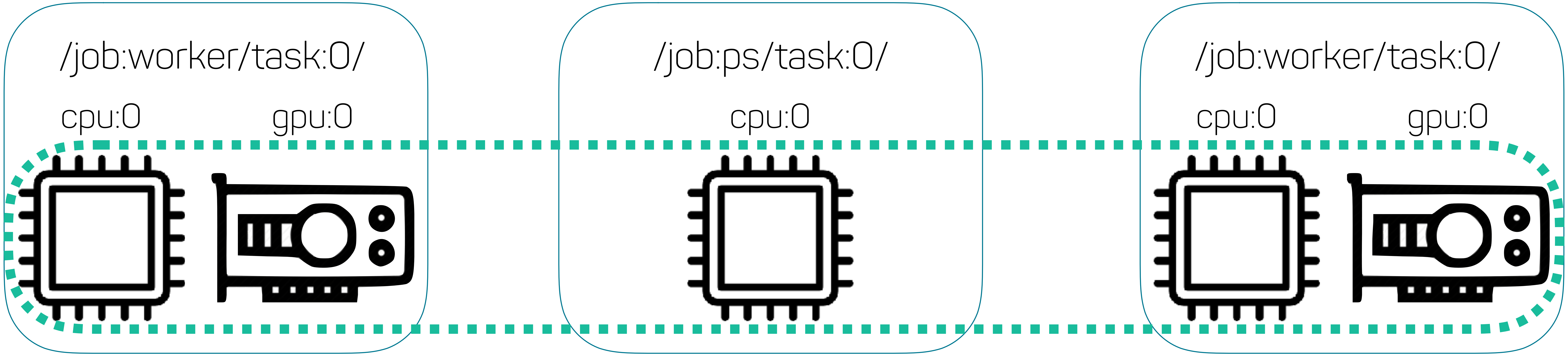


In Graph Replication

- Split up input into equal chunks,
- Loops over workers and assign a chunk
- collect results and optimise
- Not the recommended way
- Graph get big, lot of communication overhead
- Each device operates on all data

```
with tf.device("/job:ps/task:0/cpu:0"):
    W = tf.Variable()
    b = tf.Variable()
y_split = tf.split(0, num_workers, y)
for i in range(num_workers):
    with tf.device("job:worker/task:%d/gpu:0" % i):
        y = tf.matmul(y_split[i], W) + b
train_op = optimze_loss(y, y_)
```

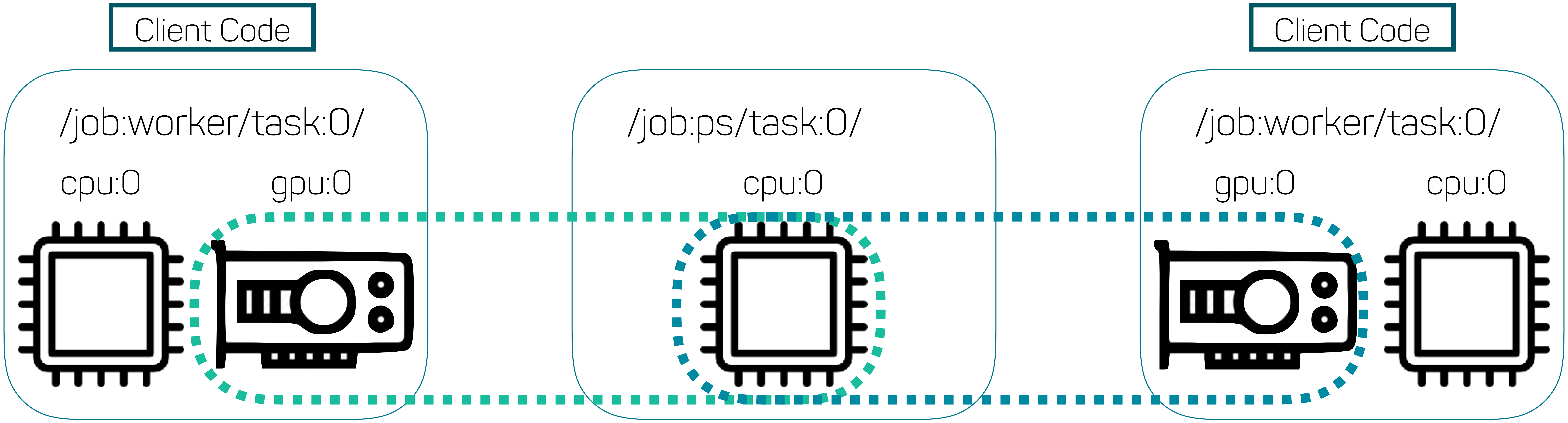
Client Code



Between Replication

- Recommend way of doing replication
- Similiar to MPI
- Each device operates on a partition
- Different Client Program on each worker
    - Assign itself to local resources
    - Small graph independently

```
with tf.device("/jop:ps/task:0/cpu:0"):
    W = tf.Variable()
    b = tf.Variable()
with tf.device("/job:worker/task:0/gpu:0"):
    y = tf.matmul(input, W) + b
    train_op = optimze_loss(y, y_)
```



Variable Placement

- How to place the Variable onto different devices
- Manual Way
    - Easy to start, full flexibility
    - Gets annoying soon
- Device setter
    - Automatic assign variables to ps and ops to workers
    - Simple round robin by default
    - Greedy Load Balancing Strategy
- Partitioned Values
    - Needed for really large variables (often used in text embeddings)
    - Splits variables between multiple Parameter Server

Training Modes

How to update the parameters between instances?

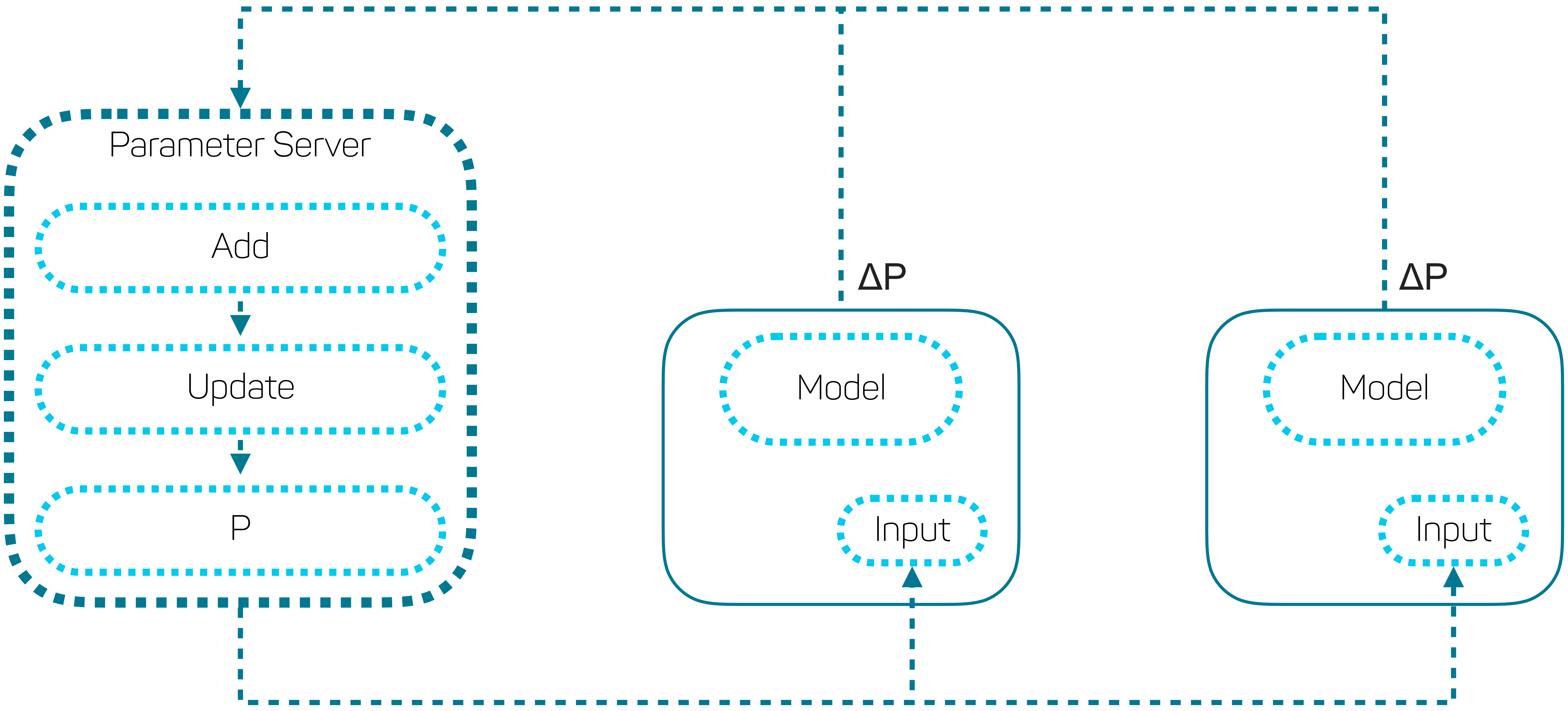
Syncronous Replication

Every Instances reads the same values for current parameters, computes the gradient in parallel and the app them together.

Asycronoues Replication

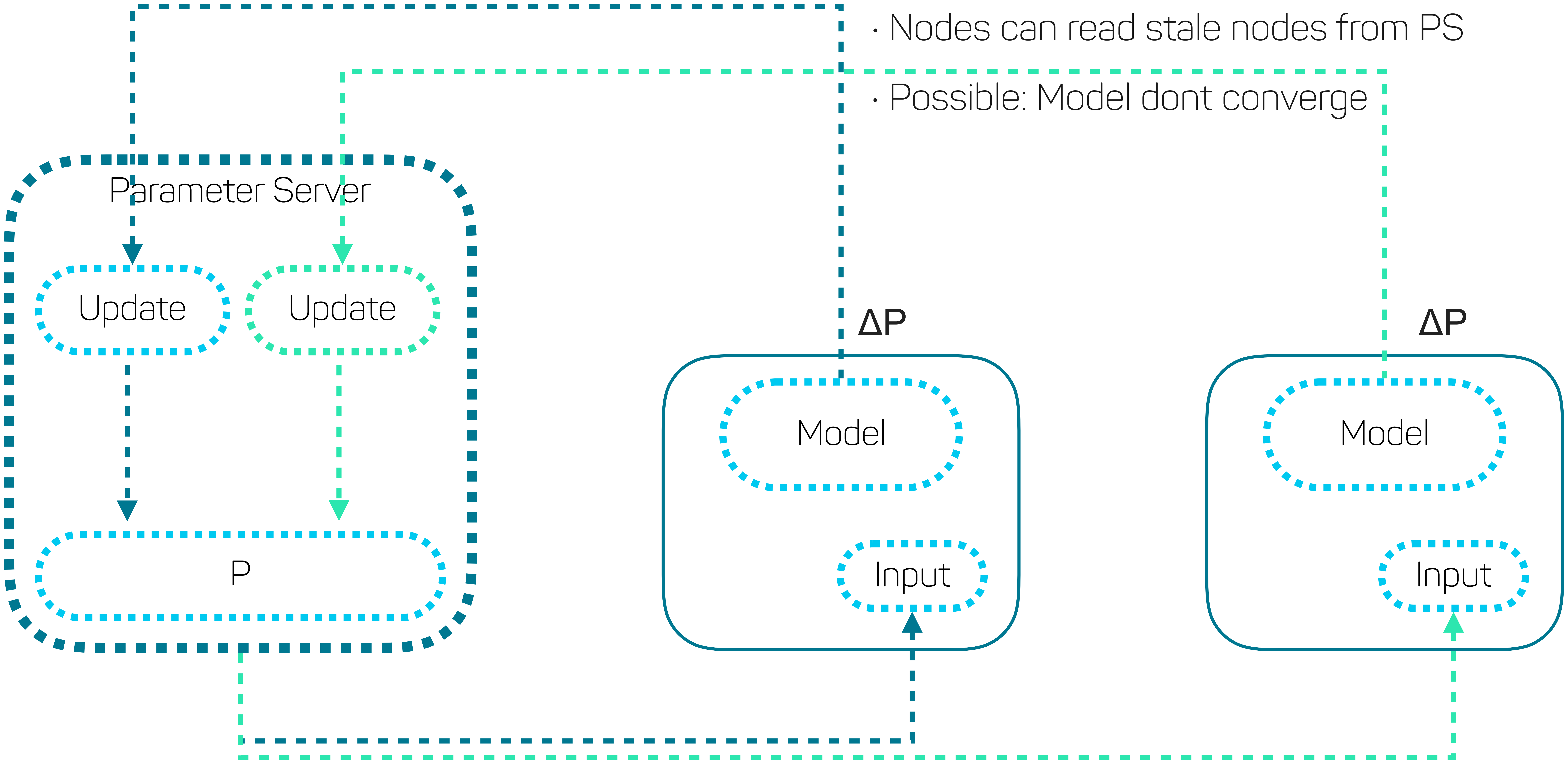
Independent training loop in every Instance, without coordination. Better performance but lower accuracy.

Synchronous Training



Asyncronous Training

- Each Updates Independently
- Nodes can read stale nodes from PS
- Possible: Model dont converge



1. Define the Cluster

- Define **ClusterSpec**
    - List Parameter Servers
    - List Workers
- PS & Worker are called ***Jobs***
- ***Jobs*** can contain one ore more ***Tasks***
- *Create* ***Server*** *for every* ***Task***

```
# 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":
  print("ps job, joining server")
  server.join()
elif FLAGS.job_name == "worker":
  print("worker job, building graph")
```

2. Assign Operation to every Task

- Same on every Node for In-Graph
- Different Devices for Between-Graph
- Can also be used to set parts to GPU and parts to CPU

```
with tf.device("/jop:ps/task:0/cpu:0"):
    W_0 = tf.Variable()
    b_0 = tf.Variable()
with tf.device("/jop:ps/task:1/cpu:0"):
    W_1 = tf.Variable()
    b_1 = tf.Variable()
with tf.device("/job:worker/task:1/gpu:0"):
    y_1 = tf.matmul(input, W_0) + b_0
    #compute intesive Part
with tf.device("/job:worker/task:2/gpu:0"):
    y = tf.matmul(y_1, W_1) + b_1
    #compute intensive Part
    train_op = optimze_loss(y, y_)
```

3. Create a Training Session

- **tf.train.MonitoredTrainingSession** or **tf.train.Supervisor** for Asyncronous Training
  - Takes care of initialisation
  - Snapshotting
  - Closing if an error occurs
  - Hooks
  - Summary Ops, Init Ops

```python
sv = tf.train.Supervisor(is_chief=(FLAGS.task_index == 0),
                         logdir="/tmp/train_logs",
                         init_op=init_op,
                         summary_op=None,
                         saver=tf.train.Saver(),
                         global_step=global_step,
                         save_model_secs=600)
with sv.managed_session(server.target) as mon_sess:
    for epoch in range(20):
        #Train
        mon_sess.run(train_op)
```

- **tf.train.SyncReplicaOptimizer** for synchronous training:
  - Also create a supervisor that takes over the role a a master between workers.

All Together - Server Init

```
# cluster specification
parameter_servers = ["pc-01:2222"]
workers = [ "pc-02:2222", "pc-03:2222", "pc-04:2222"]
cluster = tf.train.ClusterSpec({"ps":parameter_servers, "worker":workers})

tf.app.flags.DEFINE_string("job_name", "", "Either 'ps' or 'worker'")
tf.app.flags.DEFINE_integer("task_index", 0, "Index of task within the job")
FLAGS = tf.app.flags.FLAGS

# start a server for a specific task
server = tf.train.Server(cluster, job_name=FLAGS.job_name,
                          task_index=FLAGS.task_index)

# config
batch_size = 100
learning_rate = 0.001
training_epochs = 20
logs_path = "/mnist/1"

from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)

if FLAGS.job_name == "ps":
  server.join()
elif FLAGS.job_name == "worker":
  # Between-graph replication
```

All Together - Building Graph

```python
# Between-graph replication
with tf.device(tf.train.replica_device_setter(
  worker_device="/job:worker/task:%d" % FLAGS.task_index,
  cluster=cluster)):

  global_step = tf.get_variable('global_step', [],
                                initializer = tf.constant_initializer(0),
                                trainable = False)

  x = tf.placeholder(tf.float32, shape=[None, 784], name="x-input")
  y_ = tf.placeholder(tf.float32, shape=[None, 10], name="y-input")

  W1 = tf.Variable(tf.random_normal([784, 100]))
  W2 = tf.Variable(tf.random_normal([100, 10]))
  b1 = tf.Variable(tf.zeros([100]))
  b2 = tf.Variable(tf.zeros([10]))

  #Softmax
  z2 = tf.add(tf.matmul(x,W1),b1)
  a2 = tf.nn.sigmoid(z2)
  z3 = tf.add(tf.matmul(a2,W2),b2)
  y  = tf.nn.softmax(z3)
```

All Together - TrainingOP

```
y  = tf.nn.softmax(z3)

cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
grad_op = tf.train.GradientDescentOptimizer(learning_rate)
train_op = grad_op.minimize(cross_entropy, global_step=global_step)
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

# create a summary for our cost and accuracy
tf.scalar_summary("cost", cross_entropy)
tf.scalar_summary("accuracy", accuracy)
summary_op = tf.merge_all_summaries()
init_op = tf.initialize_all_variables()
```

All Together - Session & Training

```
sv = tf.train.Supervisor(is_chief=(FLAGS.task_index == 0),
                         global_step=global_step,
                         init_op=init_op)

with sv.prepare_or_wait_for_session(server.target) as sess:

  writer = tf.train.SummaryWriter(logs_path, graph=tf.get_default_graph())
  for epoch in range(training_epochs):
    batch_count = int(mnist.train.num_examples/batch_size)

    for i in range(batch_count):
      batch_x, batch_y = mnist.train.next_batch(batch_size)

      _, cost, summary, step = sess.run(
                      [train_op, cross_entropy, summary_op, global_step],
                      feed_dict={x: batch_x, y_: batch_y})
      writer.add_summary(summary, step)

      print(  " Epoch: %2d," % (epoch+1),
              " Batch: %3d of %3d," % (i+1, batch_count),
              " Cost: %.4f," % cost)
```

Deployment & Distribution

Packaging

- The Application (and all its) dependencies needs to be packaged into a deployable
- Wheels
  - Code into deployable Artefact with defined dependencies
  - Dependent on runtime
- Container
  - Build Image with runtime, dependencies and code
  - Additional Tooling for building and running required (Docker)

Kubernetes

Kubernetes is a the leading Container Orchestration.

GPU Support
Alpha since 1.6
(experimental before)

Developer friendly API
Quick deployments
through simple and
flexible API.

Huge Community
One of the fastest
growing community

Open Source
Open Sourced by Google, now
member of Cloud Computing
Foundation.

Auto Scaling
Build in Auto Scaling Feature
based on Ultitisation

Bin Packing
Efficient resource utilisation

Kubernetes in 30 Seconds

The Basic you need to know for the Rest of the Talk

Pods

Pods can be 1 or more Container grouped together, smallest scheduling Unit..

API First

Everything is a Object inside the Rest API. Declarative Configuration with YAML files.

Deployments

Higher Level Abstraction to say run Pod X Times.

Service Discovery

Services are used to make Pods discovery each other.

How to enable GPU in your K8S cluster?

—KUBELET FLAG
—feature-gates=
"Accelerators=true"

```
Capacity:
 alpha.kubernetes.io/nvidia-gpu:        1
 cpu:                                   4
 memory:                                62880144Ki
 pods:                                  110
Allocatable:
 alpha.kubernetes.io/nvidia-gpu:        1
 cpu:                                   4
 memory:                                62777744Ki
 pods:                                  110
```

- Install Docker, nvidia-docker-bridge, cuda

Out of Scope for a Data Conference

Single Worker Instance

- Prepare our Docker Image

```
FROM gcr.io/tensorflow/tensorflow:1.2.0-gpu

#Add additional requirements/data
ADD mnist_cnn.py /

ENTRYPOINT ["python", "/mnist_cnn.py"]
```

- Use prebuild Tensorflow image and add additional Libraries & Custom Code (gcr.io/tensorflow/tensorflow)
- special images form cpu/gpu builds, see docker hub tags
- Build & Push to a Registry

Name	Tags	Virtual size	Uploaded
3b85732dfed7	1.3.0-devel-gpu-py3 latest-devel-gpu-py3	1.9 GB	Aug 17, 2017
f506fa8b8dcc	1.3.0-devel-py3 latest-devel-py3	838.8 MB	Aug 17, 2017
a04d9a0e180a	1.3.0-gpu-py3 latest-gpu-py3	1.4 GB	Aug 17, 2017
8570e2031528	1.3.0-py3 latest-py3	365.4 MB	Aug 17, 2017
351d3cc2e06c	1.3.0-devel-gpu latest-devel-gpu	1.8 GB	Aug 17, 2017
ea77357aa5d8	1.3.0-devel latest-devel	793.2 MB	Aug 17, 2017

Write Kubernetes Pod Deployment

- Tell kubernetes to use GPU Resource

```yaml
resources:
  requests: #Optional
    alpha.kubernetes.io/nvidia-gpu: 1
  limits:
    alpha.kubernetes.io/nvidia-gpu: 1
```

- Mount NVIDIA Libraries from Host

```yaml
volumes:
- name: nvidia-driver-375-26
  hostPath:
    path: /var/lib/nvidia-docker/volumes/nvidia_driver/375.26
- name: libcuda-so
  hostPath:
    path: /usr/lib/x86_64-linux-gnu/libcuda.so
- name: libcuda-so-1
  hostPath:
    path: /usr/lib/x86_64-linux-gnu/libcuda.so.1
- name: libcuda-so-375-26
  hostPath:
    path: /usr/lib/x86_64-linux-gnu/libcuda.so.375.26
```

```yaml
volumeMounts:
- name: nvidia-driver-375-26
  mountPath: /usr/local/nvidia
  readOnly: true
- name: libcuda-so
  mountPath: /usr/lib/x86_64-linux-gnu/libcuda.so
- name: libcuda-so-1
  mountPath: /usr/lib/x86_64-linux-gnu/libcuda.so.1
- name: libcuda-so-375-26
  mountPath: /usr/lib/x86_64-linux-gnu/libcuda.so.375.26
```

Full Pod Yaml

```yaml
kind: Pod
apiVersion: v1
metadata:
  name: gpu-pod
spec:
  containers:
  - name: gpu-container
    image: gcr.io/tensorflow/tensorflow:1.2.0-gpu
    imagePullPolicy: Always
    command: ["python"]
    args: ["-u", "-c", "import tensorflow"]
    resources:
      requests: #Optional
        alpha.kubernetes.io/nvidia-gpu: 1
      limits:
        alpha.kubernetes.io/nvidia-gpu: 1
    volumeMounts:
    - name: nvidia-driver-375-66
      mountPath: /usr/local/nvidia
      readOnly: true
    - name: libcuda-so
      mountPath: /usr/lib/x86_64-linux-gnu/libcuda.so
    - name: libcuda-so-1
      mountPath: /usr/lib/x86_64-linux-gnu/libcuda.so.1
    - name: libcuda-so-375-66
      mountPath: /usr/lib/x86_64-linux-gnu/libcuda.so.375.66
  restartPolicy: Never
```

```yaml
volumes:
- name: nvidia-driver-375-66
  hostPath:
    path: /var/lib/nvidia-docker/volumes/nvidia_driver/375.66
- name: libcuda-so
  hostPath:
    path: /usr/lib/x86_64-linux-gnu/libcuda.so
- name: libcuda-so-1
  hostPath:
    path: /usr/lib/x86_64-linux-gnu/libcuda.so.1
- name: libcuda-so-375-66
  hostPath:
    path: /usr/lib/x86_64-linux-gnu/libcuda.so.375.66
```

Distributed Tensorflow - Python Code

- Add clusterSpec and server information to code
  - Use Flags/Envirmoent Variable to inject dynamically this information

```
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":
```

- Write your TF Graph
  - Either Manual Placement or automatic
- Dockerfile stays same/similiar

Distributed Tensorflow - Kubernetes Deployment

- Slightly different deployments for worker and ps nodes
- Service for each woker/ps task
- Job Name/worker index by flags

```yaml
---
kind: Service
apiVersion: v1
metadata:
  name: mnist-worker-0
spec:
  selector:
    name: mnist
    job: worker
    task: "0"
  ports:
  - port: 2222
```

```yaml
  name: worker-0
  labels:
    name: mnist
    job: worker
    task: "0"
spec:
  containers:
  - name: gpu-container
    image: krallistic/mnist-between:latest
    imagePullPolicy: Always
    command:
      - "python"
      - "/mnist.py"
    args:
      - "--job_name=worker"
      - "--task_index=0"
      - "--worker_hosts=mnist-worker-0:2222"
      - "--ps_hosts=mnist-ps-0:2222"
    ports:
      - containerPort: 2222
```

Distributed Kubernetes - Parameter Server

```yaml
---
kind: ReplicaSet
apiVersion: extensions/v1beta1
metadata:
  name: mnist-ps-0
spec:
  replicas: 1
  template:
    metadata:
      name: ps-0
      labels:
        name: mnist
        job: ps
        task: "0"
    spec:
      containers:
      - name: ps-container
        image: krallistic/mnist-between-ps:latest
        imagePullPolicy: Always
        command:
          - "python"
          - "/mnist.py"
        args:
          - "--job_name=ps"
          - "--task_index=0"
          - "--worker_hosts=mnist-worker-0:2222"
          - "--ps_hosts=mnist-ps-0:2222"
        ports:
          - containerPort: 2222
        volumeMounts:
```

```yaml
---
kind: Service
apiVersion: v1
metadata:
  name: mnist-ps-0
spec:
  selector:
    name: mnist
    job: ps
    task: "0"
  ports:
  - port: 2222
```

Automation - Tensorflow Operator

- Boilerplate code for larger cluster
- Official Documentation: Jinja Templating
- Tensorflow Operator:
  - Higher level description, creates lower level objects.
  - Still in the Kubernetes API (though CustomResourceDefinition)
  - ***kubectl get tensorflow***
  - Comping Soon: https://github.com/krallistic/tensorflow-operator

```
apiVersion: "krallistic.github.com/v1"
kind: "Tensorflow"
metadata:
  name: test-mnist-1
spec:
    image: krallistic/mnist-between:v1
    ps_count: 2
    worker_count: 3
    gpu_worker: true
    tensorboard:
      active: true
      folder: "/summary"
```

Additional Stuff

- Tensorboard:
  - Needs a global shared filesystem
  - Instances write into subfolder
  - Tensorboard Instances reads full folder
- Performance
  - Scales amount of Parameter Servers
  - Many CPU nodes can be more cost efficient

Questions?

Address

codecentric AG
Gartenstraße 69a
76135 Karlsruhe

Contact Info

E-Mail: jakob.karalus@codecentric.de
Twitter: @krallistic
Github: krallistic
www.codecentric.de