11-695: AI Engineering
Devices Management

LTI/SCS

Spring 2020

Outline

Basic of Using Devices

Carnegie Mellon

tf_devices

```
1  print(tf.config.get_visible_devices()) # [PhysicalDevice(name='/physical_device:CPU:0',
2  #device_type='CPU'), PhysicalDevice(name='/physical_device:GPU:0',
3  #device_type='GPU'), PhysicalDevice(name='/physical_device:GPU:1',device_type='GPU')]
4
5  x = tf.constant([1.])
6  # <tf.Tensor: shape=(1,), dtype=float32, numpy=array([1.], dtype=float32)>
7
8  # specific GPU or CPU in the beginning
9  with tf.device("/gpu:0"): # or "/device:gpu:0"
10   x = tf.constant([1.0])
11
12 # GPUs
13 x = tf.identity(x).gpu()
14 x = tf.identity(x).gpu(gpu_index=1)
15 # or CPU
16 x = tf.identity(x).cpu() # no param. Note: same description!
```

- By default, GPU is used if available and visible
- You can move data between GPU or CPU

Basic of Using GPUs

Carnegie Mellon

tf_gpu

```
1  gpus = tf.config.list_physical_devices('GPU')
2
3  logical_gpus = tf.config.list_logical_devices('GPU')
4  # [LogicalDevice(name='/device:GPU:0', device_type='GPU'),
5  #   LogicalDevice(name='/device:GPU:1', device_type='GPU')]
6
7  # or can do: CUDA_VISIBLE_DEVICES=1
8  tf.config.set_visible_devices(gpus[1], device_type='GPU') # only works with physical
9
10 # allow memory to grow from small amount, only works with physical
11 # or can do: TF_FORCE_GPU_ALLOW_GROWTH=True
12 try:
13     tf.config.experimental.set_memory_growth(gpus[0], True) # experimental
14 except:
15     pass
```

- You can limit your usage for others
- You can also share a part of each GPU for others

Manual Place Models on Devices[1]

Carnegie Mellon

tf_manual

```
1  tf.debugging.set_log_device_placement(True)
2  gpus = tf.config.list_logical_devices('GPU')
3  if gpus:
4    c = []
5    for gpu in gpus: # Replicate your computation on multiple GPUs
6      with tf.device(gpu.name):
7        a = tf.constant([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
8        b = tf.constant([[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]])
9        c.append(tf.matmul(a, b))
10   with tf.device('/CPU:0'):
11     matmul_sum = tf.add_n(c)
12
13   print(matmul_sum)
```

tf_manual_keras_model

```
1    for gpu in gpus:
2      inputs = tf.keras.layers.Input(shape=(1,))
3      predictions = tf.keras.layers.Dense(1)(inputs)
4      model = tf.keras.models.Model(inputs=inputs, outputs=predictions)
5      model.compile(loss='mse', optimizer=tf.keras.optimizers.SGD(learning_rate=0.2))
```

[1] https://www.tensorflow.org/guide/gpu

Outline

Mirrored Strategy

Carnegie Mellon

tf_mnist_multi_gpus.py

```
1  tf.debugging.set_log_device_placement(True)
2
3  strategy = tf.distribute.MirroredStrategy()
4
5  with strategy.scope():
6    inputs = tf.keras.layers.Input(shape=(1,))
7    predictions = tf.keras.layers.Dense(1)(inputs)
8    model = tf.keras.models.Model(inputs=inputs, outputs=predictions)
9    model.compile(loss='mse',
10                 optimizer=tf.keras.optimizers.SGD(learning_rate=0.2))
```

- `tf` will help you replicate your model to all visible GPUs
- Using `tf.distribute.Strategy` is always recommended

Using Multiple GPUs - Strategy

Carnegie Mellon

tf_mnist_multi_gpus

```
# sync mode, single machine: Important!
strategy = tf.distribute.MirroredStrategy()
# or: strategy = tf.distribute.MirroredStrategy(devices=["/gpu:0", "/gpu:1"])

with strategy.scope():
  model = tf.keras.Sequential([
      tf.keras.layers.Conv2D(32, 3, activation='relu', input_shape=(28, 28, 1)),
      tf.keras.layers.MaxPooling2D(),
      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(64, activation='relu'),
      tf.keras.layers.Dense(10)
  ])

  model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                optimizer=tf.keras.optimizers.Adam(),
                metrics=['accuracy'])
```

- Turn on the Mirror Distributed Mode first
- Create datasets and a model as usual

Using Multiple GPUs - Batch size

Carnegie Mellon

tf_mnist_multi_gpus_batchsize

```
1  # sync mode, single machine: Important!
2  strategy = tf.distribute.MirroredStrategy()
3  # or: strategy = tf.distribute.MirroredStrategy(devices=["/gpu:0", "/gpu:1"])
4
5  BATCH_SIZE_PER_REPLICA = 64
6  BATCH_SIZE = BATCH_SIZE_PER_REPLICA * strategy.num_replicas_in_sync
```

- Batch size will be the one before splitting

Using Multiple GPUs - Train/Eval

Carnegie Mellon

tf_mnist_multi_gpus_2

```
1  # Callback for printing the LR at the end of each epoch.
2  class PrintLR(tf.keras.callbacks.Callback):
3    def on_epoch_end(self, epoch, logs=None):
4      print('\nLearning rate for epoch {} is {}'.format(epoch + 1,
5                                                       model.optimizer.lr.numpy()))
6  callbacks = [
7      tf.keras.callbacks.TensorBoard(log_dir='./logs'),
8      tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_prefix,
9                                         save_weights_only=True),
10     tf.keras.callbacks.LearningRateScheduler(decay),
11     PrintLR()
12 ]
13
14 # train
15 model.fit(train_dataset, epochs=12, callbacks=callbacks)
16
17 # eval
18 model.load_weights(tf.train.latest_checkpoint(checkpoint_dir))
19 eval_loss, eval_acc = model.evaluate(eval_dataset)
```

- Treat the rest the same: train/eval the model as usual
- `tf` will mirror the model to all GPUs and split data automatically

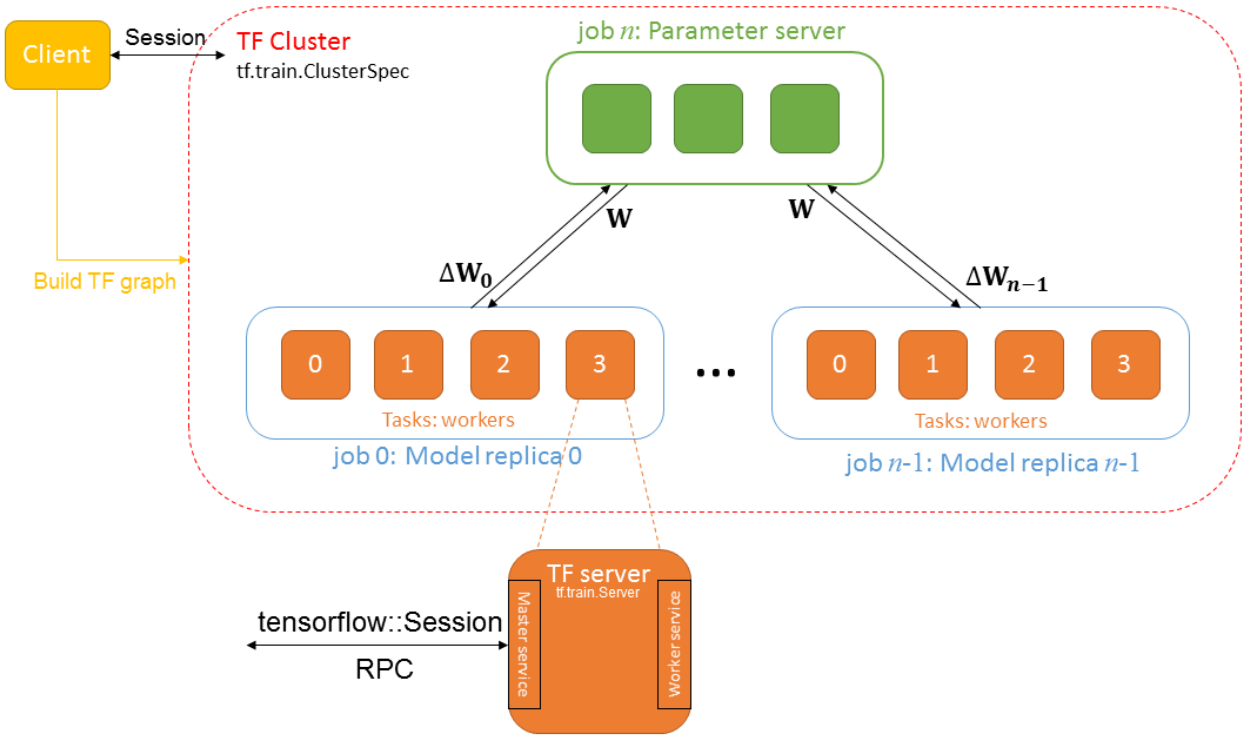
More Resources

Carnegie Mellon

- Strategy: API ▸ MirroredStrategy
- Tutorial: ▸ MNIST Multi-GPU
- Tutorial: ▸ Transformer Multi-GPU

Outline

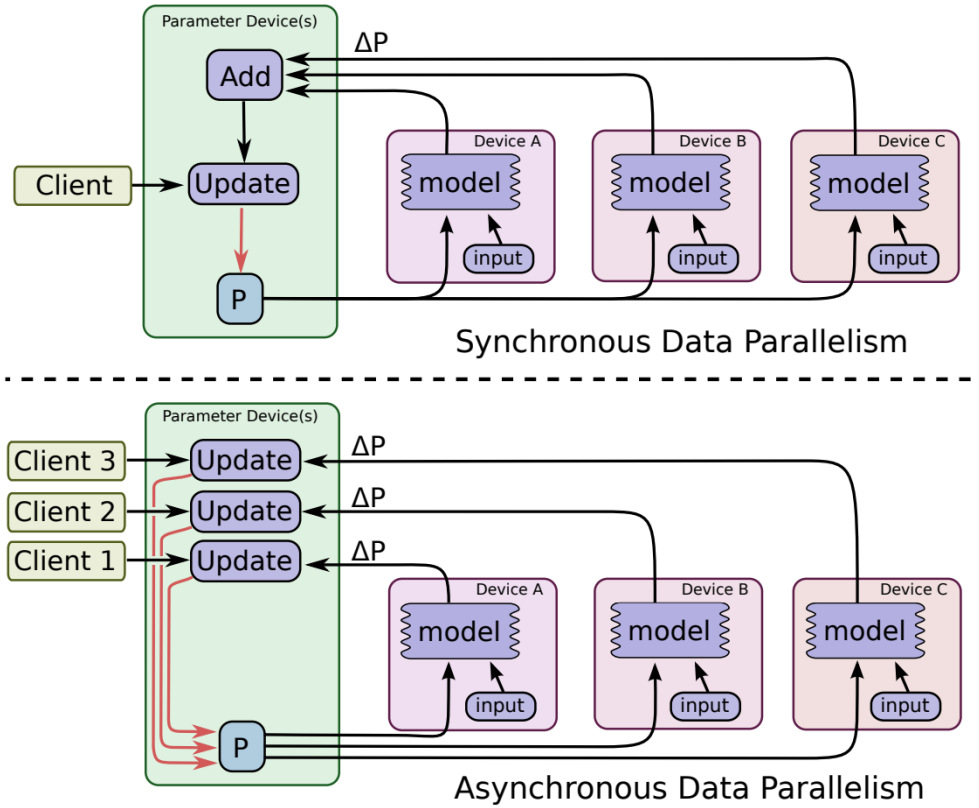
Terminologies



- A client builds a graph, creates a session
- A cluster = {jobs}, a job = {tasks}, a task = a process
- Param server is usually used in asynchronous settings

Image credit: pittnuts.com

Sync vs Async (Data Parallelism)



- Sync: W is *stateful*
- Async: state-*less*

Image credit: Tensorflow White Paper (2015)

Define A Cluster

Carnegie Mellon

tf_cluster_spec.py

```
tf.train.ClusterSpec({
    "worker": [
        "worker0.example.com:2222",
        "worker1.example.com:2222"],
    "ps": [
        "ps0.example.com:2222",]})

# 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()
else: # worker
    # define computational graphs for all available workers
        ...
```

- Need a cluster spec containings param server(s) and worker(s) addresses

Define Jobs

Carnegie Mellon

tf_cluster_graph.py

```
else: # worker
     with tf.device(tf.train.replica_device_setter(
        worker_device="/job:worker/task:%d" % FLAGS.task_index,
        cluster=cluster)):
        dataset = ...
        model = ...
        model.compile(...)

        ... # run
```

- Create separate jobs for param server(s) and worker(s)

Run Jobs

Carnegie Mellon

tf_cluster_run.sh

```
# On ps1.example.com:
$ python trainer.py \
    --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
    --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
    --job_name=ps --task_index=1
# On worker0.example.com:
$ python trainer.py \
    --ps_hosts=ps0.example.com:2222 \
    --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
    --job_name=worker --task_index=0
# On worker1.example.com:
$ python trainer.py \
    --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
    --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
    --job_name=worker --task_index=1
```

- Separate for each node.
- Tutorial: ▸ Distributed Tensorflow
- Example: ▸ Tensorflow Ecosystem

Run Jobs

Carnegie Mellon

tf_cluster_run.sh

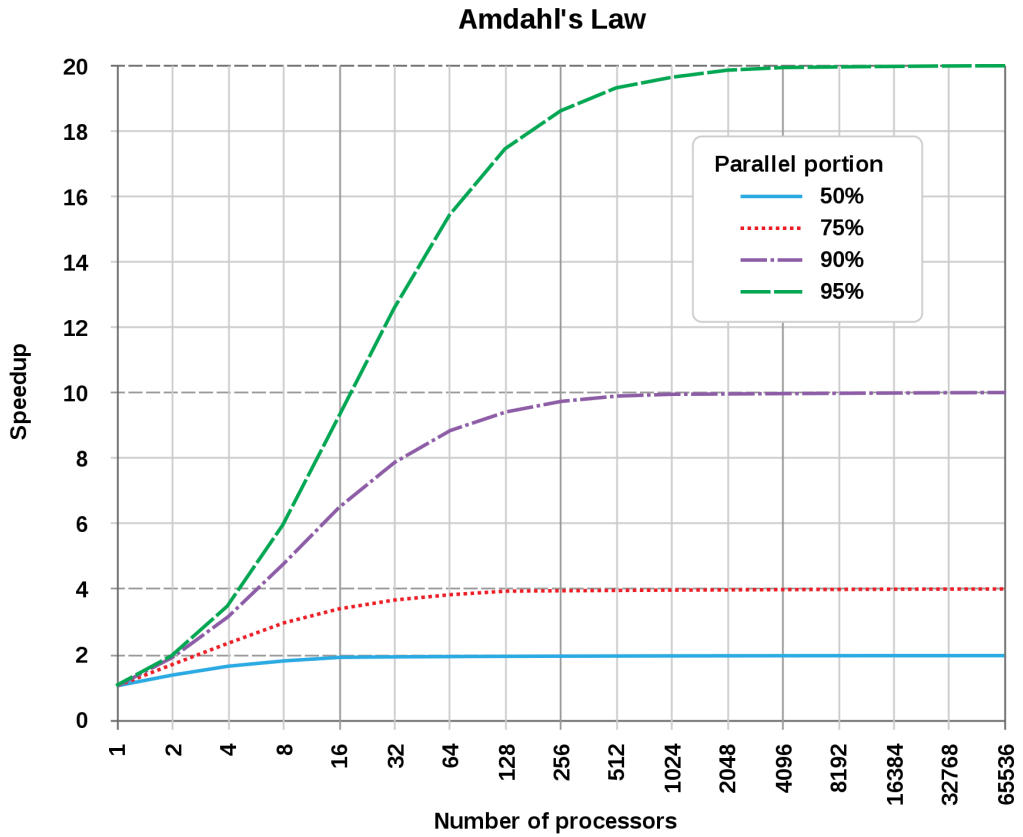
```
# On ps1.example.com:
$ python trainer.py \
    --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
    --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
    --job_name=ps --task_index=1
# On worker0.example.com:
$ python trainer.py \
    --ps_hosts=ps0.example.com:2222 \
    --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
    --job_name=worker --task_index=0
# On worker1.example.com:
$ python trainer.py \
    --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
    --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
    --job_name=worker --task_index=1
```

- Separate for each node.
- Tutorial: ▸ Distributed Tensorflow
- Example: ▸ Tensorflow Ecosystem
- Tutorial: ▸ SSGD (Sync) ▸ HogWild (Async)

Outline

Choose setting: Amdahl's Law (1967)

Carnegie Mellon

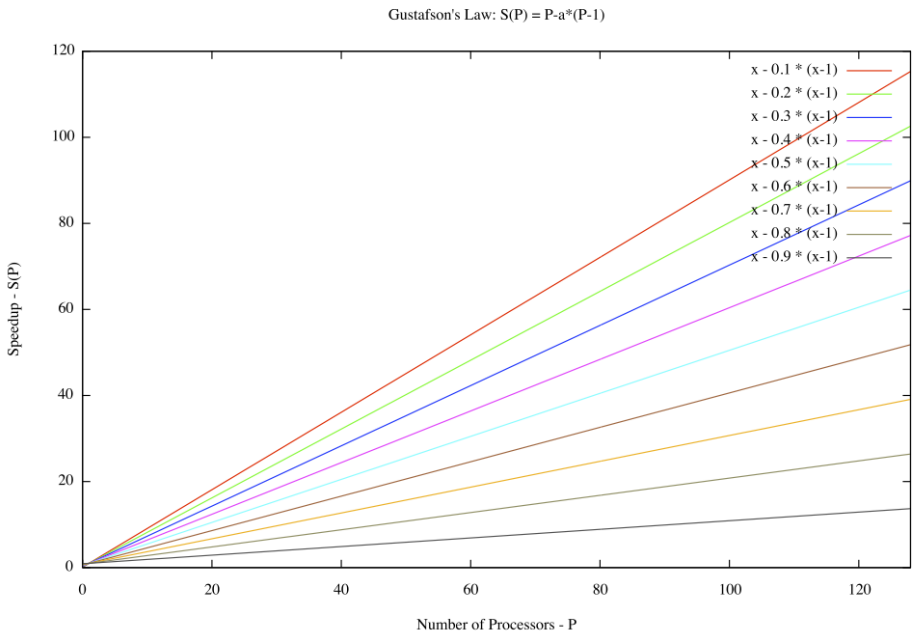


- Parallelism is useful *only* for highly parallelizable programs.
- Amdahl's is only for a fixed problem: resources are considered unchanged all the time.

Image credit: wikipedia

More realistic: Gustafson's Law (1988)

Carnegie Mellon



- Speed up with N processors: $N - (N-1)s$, where s is the fraction not amenable for parallelism
- But both are limited to processors
- In practice, clusters have a lot of latency in communication
- There's a tradeoff, such as GPU vs. CPU

Image credit: wikipedia