Data is transmitted in the form of *protocol buffers*, another open source Google technology. This is a lightweight binary data interchange format.

All servers in a TensorFlow cluster may communicate with any other server in the cluster, so make sure to open the appropriate ports on your firewall.

Every TensorFlow server provides two services: the *master service* and the *worker service*. The master service allows clients to open sessions and use them to run graphs. It coordinates the computations across tasks, relying on the worker service to actually execute computations on other tasks and get their results.

This architecture gives you a lot of flexibility. One client can connect to multiple servers by opening multiple sessions in different threads. One server can handle multiple sessions simultaneously from one or more clients. You can run one client per task (typically within the same process), or just one client to control all tasks. All options are open.

Pinning Operations Across Tasks

You can use device blocks to pin operations on any device managed by any task, by specifying the job name, task index, device type, and device index. For example, the following code pins `a` to the CPU of the first task in the `"ps"` job (that's the CPU on machine A), and it pins `b` to the second GPU managed by the first task of the `"worker"` job (that's GPU #1 on machine A). Finally, `c` is not pinned to any device, so the master places it on its own default device (machine B's GPU #0 device).

```
with tf.device("/job:ps/task:0/cpu:0")
    a = tf.constant(1.0)

with tf.device("/job:worker/task:0/gpu:1")
    b = a + 2

c = a + b
```

As earlier, if you omit the device type and index, TensorFlow will default to the task's default device; for example, pinning an operation to `"/job:ps/task:0"` will place it on the default device of the first task of the `"ps"` job (machine A's CPU). If you also omit the task index (e.g., `"/job:ps"`), TensorFlow defaults to `"/task:0"`. If you omit the job name and the task index, TensorFlow defaults to the session's master task.

Sharding Variables Across Multiple Parameter Servers

As we will see shortly, a common pattern when training a neural network on a distributed setup is to store the model parameters on a set of parameter servers (i.e., the tasks in the "ps" job) while other tasks focus on computations (i.e., the tasks in the "worker" job). For large models with millions of parameters, it is useful to shard these parameters across multiple parameter servers, to reduce the risk of saturating a single parameter server's network card. If you were to manually pin every variable to a different parameter server, it would be quite tedious. Fortunately, TensorFlow provides the `replica_device_setter()` function, which distributes variables across all the "ps" tasks in a round-robin fashion. For example, the following code pins five variables to two parameter servers:

```
with tf.device(tf.train.replica_device_setter(ps_tasks=2):
    v1 = tf.Variable(1.0)  # pinned to /job:ps/task:0
    v2 = tf.Variable(2.0)  # pinned to /job:ps/task:1
    v3 = tf.Variable(3.0)  # pinned to /job:ps/task:0
    v4 = tf.Variable(4.0)  # pinned to /job:ps/task:1
    v5 = tf.Variable(5.0)  # pinned to /job:ps/task:0
```

Instead of passing the number of `ps_tasks`, you can pass the cluster spec `cluster=cluster_spec` and TensorFlow will simply count the number of tasks in the "ps" job.

If you create other operations in the block, beyond just variables, TensorFlow automatically pins them to "/job:worker", which will default to the first device managed by the first task in the "worker" job. You can pin them to another device by setting the `worker_device` parameter, but a better approach is to use embedded device blocks. An inner device block can override the job, task, or device defined in an outer block. For example:

```
with tf.device(tf.train.replica_device_setter(ps_tasks=2)):
    v1 = tf.Variable(1.0)  # pinned to /job:ps/task:0 (+ defaults to /cpu:0)
    v2 = tf.Variable(2.0)  # pinned to /job:ps/task:1 (+ defaults to /cpu:0)
    v3 = tf.Variable(3.0)  # pinned to /job:ps/task:0 (+ defaults to /cpu:0)
    [...]
    s = v1 + v2            # pinned to /job:worker (+ defaults to task:0/gpu:0)
    with tf.device("/gpu:1"):
        p1 = 2 * s         # pinned to /job:worker/gpu:1 (+ defaults to /task:0)
        with tf.device("/task:1"):
            p2 = 3 * s     # pinned to /job:worker/task:1/gpu:1
```

This example assumes that the parameter servers are CPU-only, which is typically the case since they only need to store and communicate parameters, not perform intensive computations.