you want.[2] If you have several servers on one machine, you will need to ensure that they don't all try to grab all the RAM of every GPU, as explained earlier. For example, in Figure 12-6 the `"ps"` task does not see the GPU devices, since presumably its process was launched with `CUDA_VISIBLE_DEVICES=""`. Note that the CPU is shared by all tasks located on the same machine.

If you want the process to do nothing other than run the TensorFlow server, you can block the main thread by telling it to wait for the server to finish using the `join()` method (otherwise the server will be killed as soon as your main thread exits). Since there is currently no way to stop the server, this will actually block forever:

```
server.join()  # blocks until the server stops (i.e., never)
```

Opening a Session

Once all the tasks are up and running (doing nothing yet), you can open a session on any of the servers, from a client located in any process on any machine (even from a process running one of the tasks), and use that session like a regular local session. For example:

```
a = tf.constant(1.0)
b = a + 2
c = a * 3

with tf.Session("grpc://machine-b.example.com:2222") as sess:
    print(c.eval())  # 9.0
```

This client code first creates a simple graph, then opens a session on the TensorFlow server located on machine B (which we will call the *master*), and instructs it to evaluate `c`. The master starts by placing the operations on the appropriate devices. In this example, since we did not pin any operation on any device, the master simply places them all on its own default device—in this case, machine B's GPU device. Then it just evaluates `c` as instructed by the client, and it returns the result.

The Master and Worker Services

The client uses the *gRPC* protocol (*Google Remote Procedure Call*) to communicate with the server. This is an efficient open source framework to call remote functions and get their outputs across a variety of platforms and languages.[3] It is based on HTTP2, which opens a connection and leaves it open during the whole session, allowing efficient bidirectional communication once the connection is established.

2 You can even start multiple tasks in the same process. It may be useful for tests, but it is not recommended in production.

3 It is the next version of Google's internal *Stubby* service, which Google has used successfully for over a decade. See *http://grpc.io/* for more details.

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.