Minimal Configuration for ipyparallel over SSH

Lee Phillips

The simplest case for using ipyparallel is when you want to create a compute cluster consisting of machines that you can reach through SSH. This note assumes that you have distributed public and private keys to the remote machines so that you can connect to them from your local machine, and they can connect to you, without being prompted for a password. The other requirement is that you adjust any firewalls or routers to allow connections into your local machine. IPython seems to connect on various ports from about 30000 to about 60000.

After your SSH networking is sorted out, you need to create a new "profile", with the command

```
ipython profile create --parallel --profile=snet
```

You can use any name of your choice in place of "snet". Now you will have a new directory in your `.ipython` directory called `profile_snet`, within which you will find four configuration files. You will need to make some

edits to these files to spell out the details of your networked computing cluster. Everything in these files is initially commented out, so you can either add the following edits to the end or find the appropriate lines and edit them.

In `ipcluster_config.py`, you will need the lines

```
c.IPClusterEngines.engine_launcher_class = 'SSH'
c.IPClusterStart.controller_launcher_class = 'Local'
c.LocalControllerLauncher.controller_args = ["--ip=*"]
c.IPClusterStart.controller_location = 'IP'
c.SSHLauncher.hostname = 'LOCALHOSTNAME'
c.SSHEngineSetLauncher.engines = {'host1.example.com' : 2, 'LOCALHOSTNAME': 4}
```

where you must replace `IP` by the public-facing IP of your local machine, and `LOCALHOSTNAME` by a its hostname.

The list of "engines" in the last line specifies how to assemble the compute cluster. Here we've defined two engines on `host1` and, as before, four engines on our local machine.

In `ipcontroller_config.py`, insert the line

```
c.HubFactory.client_ip = 'IP'
```

using the same `IP` as above.

With that, the preliminaries are complete. We will again use the `ipcluster` command, but with an extra argument that refers to our new profile: `ipcluster start --profile="snet"`

This command will copy files to the remote hosts with connection and security information. After the command completes, you can interact with your distributed compute cluster from IPython just as you do using a local cluster. The cell magic `%%px` will now automatically send data and code to the engines defined in your configuration file, and retrieve the results from the network.

IPython supports many other strategies for distributed computation, including MPI and various batch systems, explained in the voluminous documentation, but SSH is the easiest to explain and the quickest to set up.

These techniques are not without certain security implications. They increase the attack surface for an adversary with access to any of the networked machines, who may be able to take advantage of exploits to compromise the other machines taking part in the distributed computation. For this reason it is generally regarded as prudent to only use these methods to connect to computers with trusted users.