Jesi Merrick

Ron Pyka

CS553-01

Manual

To run these programs, two EC2 instances must be started (t2.micro works in this case, with the Ubuntu AMI). One of these will serve as the client, and the client.py program should be loaded onto it accordingly. The second will serve as the scheduler. The python package used, boto, must be installed. See here for instructions, making note of the instructions for the required configuration file as well. Along with this, transfer the scheduler.py program to the scheduler instance.

The workers are created using a private AMI based on the Ubuntu AMI. The private AMI has boto installed on it (with the required credentials file), the worker.py file, and uses cron to launch the worker.py file at the start. To add the cron job when setting up the instance that will be used to create the AMI, use the following command:

```
crontab -e
```

In the file that opens, add the following line:

```
@reboot /usr/bin/python /home/ubuntu/worker.py -i [IDLE_TIME]
```

where IDLE_TIME is the number of seconds a worker will idle for before shutting down. Save the file. At this point, an AMI can be made from the instance. The id of this AMI is used in trackRWorkers() in scheduler.py. You may have to check additional settings in this method (such as the key name and security groups to be used for the instance).

To run the programs, ssh into the scheduler instance and start the scheduler.

To use local workers:

```
python scheduler.py -s [PORT] -lw [NUMBER OF WORKERS]
```

To use remote workers:

```
python scheduler.py -s [PORT] -rw
```

With the scheduler running, the client may now be started to load the work. Simply ssh into the client instance and use the following command:

```
python client.py [SCHEDULER_IP:PORT] -w [WORKFILE]
```

Note that the scheduler public IP can be found through the AWS EC2 portal, the port numbers of the scheduler and client should match, and the workfile should contain sleep() commands, one per line, in the following form:

```
sleep [TIME TO SLEEP]
```