Sparrow

Replicating Sparrow in Python

1. Installation

A. Setting up a virtual environment

Because we require Pyro4, a Python package not installed on the CS machines, we'll need to establish a Python virtual environment, which is already installed on the machines.

To setup the virtual environment, SSH to an RIT CS machine, navigate to a directory where you would like to place your environment, and simply type the command, `virtualenv <name_of_environment>`. One common naming convention is to call your environment directory `venv`. In this case, you would enter `virtualenv venv`. Within the directory is now an installation of Python and a few basic packages.

To activate the virtual environment: `source <name_of_environment>/bin/activate`. Example: `source venv/bin/activate`. To deactivate the virtual environment, simply type the command `deactivate`.

If you're interested in learning more/didn't understand fully, feel free to check out further instructions here.

B. Pyro4 Installation

This project uses an RMI-like library for Python, called Pyro4. As mentioned previously, this library has to be installed manually into our virtual environment.

Installation in PyCharm

1. Go to `Settings > Project: name_of_project > Project Interpreter`
2. Hit the `+`
3. Search for `Pyro4` (not `Pyro`!)
4. Hit `Install Package`

Installation on an RIT machine

Make sure your virtual environment as listed above.

```
source venv/bin/activate
pip install --upgrade pip # Gets the latest version of pip
pip install Pyro4 # Installs Pyro4
```

If this didn't work, try the instructions found here

C. Code

Copy the entire code directory over (as directory structure matters for the web interface).

2. Getting Started

Be sure to launch the nodes in the following order, or you may run into connection issues.

A. Name Server

By default, the name server should be run at `newyork`, as the defaults in all of the files are set to look here. You can overwrite this with a `nameserver_hostname` parameter provided in most of the class constructors if you would like, but do so at your own risk.

From `newyork`, run:

```
cd project-dir
source ./venv/bin/activate # May change depending on your virtual env location
pyro4-ns -n newyork
```

You should launch this

B. Scheduler

Launching the scheduler is really straightforward. Activate your virtual env and execute:

```
python Scheduler.py
```

C. Workers

We tested up to 10 workers running in parallel, but the system should be able to handle any number of workers. One assumption we make is that there is only one worker per machine - that is, `arizona` should only have a single worker. This is in order to avoid namespacing issues, and could be rectified, but was low priority. To start a worker, activate your virtual env and run:

```
python Worker.py
```

If the scheduler ever goes down, you will have to restart the workers to renew their connection to the scheduler.

D. Visualization

Before you launch any jobs, you probaby want to launch the interface.

To ensure smooth running, please replicate the directory structure as given in order to produce the visualization.

```
src - directory containing Scheduler.py, __init__.py, Worker.py, SparrowClient.py,
Job.py
web - directory containing Chart bundle min.js, bootstrap.min.css, index.html,
main.js
server.py
```

To execute, SSH to an RIT server and run:

```
python server.py
```

This will print out a base address, which you will use to view the interface.

To view the visualization, open your web browser to the server address where `server.py` is running.

For example, if `server.py` is running on `newyork` (`newyork.cs.rit.edu`), go to the address http://newyork.cs.rit.edu:8901/sparrow.

E. Executing a Routine

This is where we assign jobs and tasks to the scheduler. To simplify proceedings, we chose to have each task as simply telling the worker to sleep for a certain duration of time. Jobs are comprised of tasks, and can be hetero- and homogeneous.

To execute a routine, SSH into an RIT server and run:

```
python SparrowClient.py  <method>  <no_of_jobs>  <no_of_tasks>  <duration>
[task_spread]
```

where,

`method` is a string describing the method to be used (don't include the ticks around the variable name)

```
Options are -

`RANDOM` - random method

`CHOOSE_TWO` - choose two

`BATCH` - for batch processing

`BATCH+LATE_BINDING` - for bath processing with late binding
```

`no_of_jobs` is an int describing the number of jobs to be completed

`no_of_tasks` is an int describing the number of tasks per job

`duration` is the amount of time (in seconds) each worker will sleep for

`task_spread` is a float describing variations between each job (if no argument is provided here, then spreaad is set at 0.0 by default)

Example execution: `python SparrowClient.py BATCH+LATE_BINDING 10 10 1`, which runs 10 jobs with 10 tasks each of 1 second, using batch with late binding.

If you have any further questions, feel free to email us! Apologies in advance for the complex setup process.