Blue Waters: python profiling webinar

**Function profiling**

The python cProfile builtin function profiler works with the bwpy module. To use it with your script, it can be as simple as:

```bash
$ aprun -n 1 python -m cProfile <myscript.py>
```

Optionally add the `-o <bw.profile>` and your `bw.profile` may be analyzed with any of the python profile viewers you may install in a virtualenv or via the python `pstats` module:

```bash
$ python
>>> import pstats
>>> p = pstats.Stats('bw.profile')
>>> p.sort_stats('cumulative').print_stats(5)
```

**function profiling is built-in**

Most python installations will have `cProfile` included. Its usage is highly recommended before you transition to "production" python code.

---

A new ssh will soon be deployed that can leverage Globus Authentication and any of your linked identities. For Xsede, we map `<Xsede_portal_identity>` to `<local_account>` for each resource provider in Xsede. Because the python code will be production and deployed at a variety of sites, we want to verify that performance is reasonable and look for any hot spots in the code. Functional profiling revealed the code to be performant with no noticeable hotspots save for the data query of the REST interface in front of the Xsede Central Database. This was expected.
Line level profiling
You may setup the line_profiler in a virtualenv as shown and use it to profile functions you decorate with @profile.

arnoldg@h2ologin2:~> module load bwpy
arnoldg@h2ologin2:~> virtualenv line_profiler
Using base prefix '/mnt/bwpy/single/usr'
New python executable in /mnt/a/u/staff/arnoldg/line_profiler/bin/python3.5
Also creating executable in /mnt/a/u/staff/arnoldg/line_profiler/bin/python
Installing setuptools, pip, wheel...done.
arnoldg@h2ologin2:~> source line_profiler/bin/activate
(line_profiler) arnoldg@h2ologin2:~> pip install line_profiler
Collecting line_profiler
  Downloading https://files.pythonhosted.org/packages/14/fc/ecf4e238bb601ff829068e5a72cd1bd67b0ee0ae379db172eb6a0779c6b6/line_profiler-2.1.2.tar.gz (83kB)
  100% |################################| 92kB 3.9MB/s
...
(line_profiler) arnoldg@h2ologin2:~> kernprof
Usage: kernprof [-s setupfile] [-o output_file_path] scriptfile [arg] ...

# switching into a batch job here...

(line_profiler) arnoldg@h2ologin2:~> grep --after-context=5 @profile ep_task_errors_bw.py
@profile
def my_endpoint_manager_task_list(tclient, endpoint):
    """
    Get tasks from an endpoint, then look through them for error events.
    Also mark as SRC, DEST, or SRC_DEST as the case may be.
    """

(line_profiler) arnoldg@h2ologin2:~> aprun -n 1 kernprof -l ep_task_errors_bw.py
...

(line_profiler) arnoldg@h2ologin2:~> python -m line_profiler ep_task_errors_bw.py.lprof

This is a production code we run on the backend to monitor our Globus Online endpoints for some transfer error conditions. The code was developed for the ncsa#Nearline endpoint. When it was ported to the ncsa#BlueWaters endpoint, cycle time for the python script became much longer (because that endpoint is typically involved in more transfers). We used both functional and line-level profiling to diagnose the issue. Line-level profiling pointed out a single line consuming most of the time. Some research into the Globus API for the call on that line led us to a better query formulation which greatly minimized the data returned from GO. +1 for python profiling.
+ Timer unit: 1e-06 s
+
+ Total time: 451.817 s
+ File: ep_task_errors_bw.py
+ Function: my_endpoint_manager_task_list at line 98
+
+ Line #   Hits  Time  Per Hit  % Time  Line Contents
+ ==============  ===========  ===========  ===========  ===========
+ 98          @profile
+ 99        def
+ my_endpoint_manager_task_list(tclient, endpoint):
+ 100        """
+ 101    Get tasks from an endpoint, then
+ 102    look through them for error events.
+ 103        Also mark as SRC, DEST, or SRC_DEST
+ 104        as the case may be.
+ 105        """
+ 106        source_total_files = 0
+ 107        dest_total_files = 0
+ 108        source_total_bps = 0
+ 109        dest_total_bps = 0
+ 110        source_total_tasks = 0
+ 111        dest_total_tasks = 0
+ 112    for task in
+ 113    tclient.endpoint_manager_task_list(filter_endpoint=endpoint, num_results=None):
+ 114    179509    787031.0    4.4    0.2 if task["status"] == "ACTIVE":
+ 115    191    1040.0    5.4    0.0 if
```
@profile
def my_endpoint_manager_task_list(tclient, endpoint):
    ***
    Get tasks from an endpoint, then look through them for error events.
    Also mark as SRC, DEST, or SRC_DEST as the case may be.
    ***
    source_total_files = 0
    dest_total_files = 0
    source_total_bps = 0
    dest_total_bps = 0
    source_total_tasks = 0
    dest_total_tasks = 0
    for task in tclient.endpoint_manager_task_list(filter_endpoint=endpoint,
                                                filter_status="ACTIVE",
                                                num_results=None):
        if task["destination_endpoint_id"] == endpoint:
```

References:

https://bluewaters.ncsa.illinois.edu/Python-profiling

https://github.com/ncsa/endpoint_task_errors

https://docs.globus.org/api/