How to Launch Pydpiper Executors

Command Line Flags

There are a number of command line flags included in each application and pipeline_executor.py.

Each Pydpiper application has the option of launching its own executors or they may be launched separately as detailed below

```
group = OptionGroup(parser, "Executor options",
         "Options controlling how and where the code is run.")

group.add_option("--uri-file", dest="urifile",
                 type="string", default=None,
                 help="Location for uri file if NameServer is not used. If not specified, default is current
working directory.")

group.add_option("--use-ns", dest="use_ns",
                 action="store_true",
                 help="Use the Pyro NameServer to store object locations")

group.add_option("--num-executors", dest="num_exec",
                 type="int", default=-1,
                 help="Number of independent executors to launch. [Default = -1. Code will not run without an
explicit number specified.]")

group.add_option("--time", dest="time",
                 type="string", default="2:00:00:00",
                 help="Wall time to request for each executor in the format dd:hh:mm:ss. Required only if --
queue=pbs.")

group.add_option("--proc", dest="proc",
                 type="int", default=1,
                 help="Number of processes per executor. If not specified, default is 8. Also sets max value
for processor use per executor.")

group.add_option("--mem", dest="mem",
                 type="float", default=6,
                 help="Total amount of requested memory for all processes the executor runs. If not specified,
default is 16G.")

group.add_option("--ppn", dest="ppn",
                 type="int", default=8,
                 help="Number of processes per node. Default is 8. Used when --queue=pbs")

group.add_option("--queue", dest="queue",
                 type="string", default=None,
                 help="Use specified queueing system to submit jobs. Default is None.")

group.add_option("--sge-queue-opts", dest="sge_queue_opts",
                 type="string", default=None,
                 help="For --queue=sge, allows you to specify different queues. If not specified, default is
used.")

group.add_option("--time-to-seppuku", dest="time_to_seppuku",
                 type="int", default=15,
                 help="The number of minutes an executor is allowed to continuously sleep, i.e. wait for an
available job, while active on a compute node/farm before it kills itself due to resource hogging. [Default=15
minutes]")

group.add_option("--time-to-accept-jobs", dest="time_to_accept_jobs",
                 type="int", default=180,
                 help="The number of minutes after which an executor will not accept new jobs anymore. This can
be useful when running executors on a batch system where other (competing) jobs run for a limited amount of
time. The executors can behave in a similar way by given them a rough end time. [Default=3 hours]")

parser.add_option_group(group)
```

Ignore the --use-ns option for now

The `--use-ns` option is PYRO specific and requires a name server to be running. It was included to make the code as general as possible, but for now, DO NOT USE THIS OPTION when running the code. If this changes, this wiki page will be updated.

URI Specification

Knowing the location of your URI file is critical for the pipeline executors to communicate with the pipeline appropriately. The defaults for `--uri-file` are as follows:

- When an application is launched, it creates a file called uri in the current directory (i.e. the directory where it was launched). If you would like the URI file to have a different name or location, then `--uri-file` should be specified when the application is launched, e.g. `MBM.py --uri-file=/projects/souris/username/directory/uri-file-name`.
- If executors are launched independently of the application (see below for details), it assumes the name of the URI file is uri and is in the directory where the executor is being launched. If this is not the case, then the path to the uri file created by the application must be specified.
- To avoid specifying a `--uri-file` altogether, simply launch the application and executors from the same directory (or launch executors with the application) and this will take care of itself.

Number of executors

Usage for this option is as follows:

- If `--num-executors` is not specified, your application will not run and will exit with an error message.
- If `--num-executors` = 0, then the server will set itself up and create a URI file, but will not launch any executors.
- `--num-executors`=X will launch X executors, with the appropriate amount of memory and processors (see below)

Memory and Processors

The `proc` and `mem` flags indicate the number of processes each executor should launch (managed under the hood as a pool) and the total amount of memory needed to run the pipeline.

If, for example, you are running on an 8 core machine and each of your pipeline stages uses a single cpu, then launching an executor with `--proc=8` will have each core handling a single stage in a separate thread. Similarly, you could specify `--num-executors=8 --proc=1.` Note that 1 executor with 8 processes will wait until a machine with 8 CPUs is open, while 8 executors with a single process each can be spread out over multiple machines. Neither is inherently better or worse, it just depends on cluster usage and the particulars of a given piece of code. ***For the way we currently run at MICe, `--num-executors=8 --proc=1` is preferred to `--num-executors=1 --proc=8`.***

Pypdiper also handles parallel code. For instance, let's assume several pipeline stages require 2 cpus, and you specify `--num-executors=1 --proc=8`. Then, at the time those stages are running, only 7 threads will be able to run concurrently, since a single stage will be using 2/8 cpus. (Note: This design may be updated in the future.)

Killing and Re-launching executors

The `--time-to-seppuku` and `--time-to-accept-jobs` options allow the pipeline to dynamically manage executors. Sometimes, in the course of running a pipeline, there will be a bottleneck: a single stage must complete before any other stages can be run. When this happens, having many idle executors can waste resources and prevent others from utilizing them, particularly when running on a cluster. Conversely, when there are many pipeline stages that can be run concurrently, having too few executors can cause a slowdown in execution time.

The default value for `--time-to-seppuku` is 15 minutes. This means that if a single executor is idle and cannot run any pipeline stages for 15 minutes, it will kill itself.

The default value for `--time-to-accept-jobs` is 180 minutes (3 hours). This means that if a single executor has been running for more than 3 hours, it will complete whatever stage is running and then kill itself. This happens regardless of the number of runnable stages that are available in the queue.

The pipeline also monitors the number of runnable stages in its queue and the number of executors that it initially launched (see the following section for details on launching executors from the pipeline). For example, consider a scenario where a pipeline was launched with 10 executors, and 5 executors have killed themselves after exceeding their time-to-seppuku. Once the pipeline has more than 5 stages in its runnable queue, it will launch executors until it has up to 10 executors running. In addition, if an executor has recently killed itself because it exceeded its time-to-accept jobs, but there are still runnable stages in the queue, the pipeline will launch another executor to replace it. This functionality helps ensure fair cluster access for all users.

If you are running at MICe, we recommend using the default settings for these parameters.

If you are running on scinet, the value for each of these options should be None, as the queueing system on scinet works differently, and jobs cannot be launched from compute nodes!

Launching Executors from the Pipeline

In order to allow a pipeline to launch its own executors, the `num-executors` flag should be specified at the same time that the command is specified. As is detailed above, not specifying this option means your pipeline won't run. ***Launching executors when the pipeline itself is launched is the simplest thing to do and we recommend it!!***

Using MAGeT.py as an example, the following is a sample command for starting a pipeline that will launch its own executors:

```
./MAGeT.py [--MAGeT-specific-options-here] --uri-file=/home/user/directory-name/uri --num-executors=7 --proc=1
--mem=6
```

The above pipeline will, after initialization, launch eight executors, each of which has a single process. Each executor will use 6GB of RAM.

Launching Executors from the Command Line

To run the same pipeline used in the above example, but with executors launched from the command line, you would first need to run MAGeT.py:

```
./MAGeT.py [--MAGeT-specific-options-here] --uri-file=/home/user/directory-name/uri --num-executors=0
```

Then, in a separate terminal, you would run:

```
pipeline_executor.py --uri-file=/home/user/directory-name/uri --num-executors=7 --proc=1 --mem=6
```

This would then launch eight executors, each of which has a single process and uses 6GB of RAM.

For either of the above examples, if the number of processes is larger than 1, the amount of RAM is split over all processes. So for instance, `--num-executors=6 --proc=3 --mem=6` will distribute 6GB of RAM across the three processes for each executor.

Using Batch Queueing

PydPiper currently has support for two batch queueing systems: sge and pbs. sge support uses a previously written perl script, sge_batch, currently available at MICe. `--queue=sge` should be used when running the main pipeline on your local machine, but you would like to submit the executors to a cluster that uses the sge queueing system. Executors may be launched from the command line (after the pipeline itself has been constructed and initialized) or can be launched directly from the pipeline. In the case of MAGeT, it would be done like this:

```
./MAGeT.py [--MAGeT-specific-options-here] --uri-file=/home/user/directory-name/uri --num-executors=7 --proc=1
--mem=6 --queue=sge
```

In addition, the --sge-queue-opts flag allows you to specify the name of the queue you'd like the submit to. The default is all.q, which is the default queue for the MICe cluster. To specify alternate queues, or more than one, the --sge-queue-opts flag can be used. (e.g. --sge-queue-opts=all.q,bigmem.q) This option only works when --queue=sge is also specified.

pbs support is currently implemented in queueing.py, which is included as part of pydpiper. Using `--queue=pbs` constructs and submits a batch file to a pbs queueing system based on the command line options specified by the user. In the above example, if `sge` were replaced with `pbs`, two pbs job files would be created: the first would include the MAGeT command and one executor with four processes; the second would be a single executor with four processes. Although it appears as though the pipeline is launching its own executors, in the script itself, they are launched separately. This is due to the fact that on some large clusters, jobs cannot be started from compute notes. Although the pbs implementation was coded based on the queueing system at SciNet (the cluster used by MICe), it can be easily adapted if needed to other clusters that utilize the pbs queueing system.

Additional options relevant to the --queue=pbs choice are --time and --ppn. The default for --time is 2:00:00:00 (2 days) and the default number of processors per node (ppn) is 8. Both of these are utilized in the script that is constructed and sent to the pbs queueing system.