

Pylauncher Lab

Victor Eijkhout

Throughput Computing Workshop, September 27

1 Setting up the environment for the tutorial

- Make sure you have python available:
`module load python`
- Make sure the pylauncher is available:
 - Normally you would do:
`module load pylauncher`
 - but for this lab we'll use the latest version of the launcher
`tar fxz ~/train00/pylauncher.tar.gz`
`cd pylauncher`
- You can run the examples by submitting batch jobs, but here we will do everything interactively. Request one compute node with
`/usr/bin/srun -n 16 -p normal \`
`--reservation="TACC-Training-2013-09-27" -A PROJECTNUMBER \`
`-t 0:30:00 --pty /bin/bash -l`
(for two nodes, change the 16 to 32, et cetera) where PROJECTNUMBER is
 - 20130927DataIntensive if you signed up through TACC, or
 - TG-TRA120009 if you signed up through XSEDE.
- You should now be in the pylauncher directory. Do:
`export PYTHONPATH=`pwd`: $PYTHONPATH`
(this is normally not necessary if you load the module) and go to the tutorial directory:
`cd tutorial`

2 A simple launcher example

- Run the python file `random_commandline.py` to generate a file with commandlines. Inspect the resulting file.

- **Compile the `random_wait` program:**
`cc -O -o random_wait random_wait.c`
- **Run the example:**
`python run_random.py`
 after a minute or two you get summary output.
- You should now have a directory with a name `pylauncher_tmpdir123456`. The number is your job number. An interactive session with `srun` is actually a type of batch job, so you get the number of that job.
 - Take a look in that directory. Do you see that for each task number there are three files?
 - Do `ls -s`. You'll see that files with a name `expire123` have zero size.
 - Take a look at one of the `exec123` files. On their last line is the command that they should execute, followed by a `touch` command that is the source of the `expire00` file.
 - Finally, there are the `out99` files, which contain the standard output (and `stderr`) of the corresponding `exec` file.
- Since we're running interactively, maybe you want some output on what the launcher is doing. Add `debug="job"` as parameter to the launcher call.
- Try running the example again. Take a close look at the error message. The PyLauncher wants a unique working directory for each run, and since interactively we are still in the same run you need to remove the working directory of the previous python call with `rm -rf pylauncher_tmpdir1234567`. Now you can run the example again.
- The output scrolls by rather fast. Use Control-S to stop it and Control-Q to resume. When it's stopped, can you understand what it's reporting?

3 Further examples

- If you use `PYL_ID` in a commandline, it gets replaced dynamically by the task number.
- Edit the `random_commandlines.py` file to use `PYL_ID` instead of a random number.
- Run the `run_random.py`.
- Go into the workdirectory that was generated. Take a look at a few `out99` files. Can you confirm that everything worked the way it should?