

# PyLauncher

Victor Eijkhout

Texas Advanced Computing Center

# **Motivation and basic usage**

# Why another launcher

- The shell-based launcher was static. This can lead to load unbalance
- Wish: restart
- Wish: dynamic task queue
- Wish: multicore jobs, MPI jobs
- Wish: customizability

# Availability

- Through the module system on TACC clusters
- Repository:  
<https://bitbucket.org/VictorEijkhout/pylauncher>

# Basic usage

- You need to write a little Python. Single core jobs:

```
#!/usr/bin/env python
```

```
import pylauncher
```

- Multi-core jobs:

```
pylauncher.ClassicLauncher("commandlines")
```

- Variable core:

```
pylauncher.ClassicLauncher("commandlines",cores=4)
```

```
pylauncher.ClassicLauncher("commandlines",cores="file")
```

- MPI

```
pylauncher.IbrunLauncher("commandlines",cores="file")
```

- MIC:

```
pylauncher.MICLauncher("commandlines",cores="file")
```

# Input files

```
#  
# example commandlines  
#  
echo "command 0"; sleep 21  
echo "command 1"; sleep 14  
# skip a few  
echo "command 9"; sleep 29  
  
echo "command 10"; sleep 16  
  
# core count attached  
5,echo "command 0"; sleep 21  
5,echo "command 1"; sleep 14  
3,echo "command 12"; sleep 13  
3,echo "command 13"; sleep 24  
3,echo "command 14"; sleep 28
```

- Each line is one job, note blank lines and comments, core count for “file” option

# Some implementation details

- Each pylauncher run leaves behind a directory “pylauncher\_tmpdir12345” where 12345 is jobid
- Each command line gets wrapped into a file “command1”, “command2”
- A successfully executed command leaves behind “expire1”, “expire2” &c

# Checkpoint / restart

- There is also a file “queuestate”
- If your job runs out of time or crashes:

```
pylauncher.ClassicLauncher("commandlines",cores=4,  
resume=12345)
```

- This will re-issue all not-finished commands from the aborted job



# MPI jobs

- Core count specified as before:

```
#!/usr/bin/env python
import pylauncher
pylauncher.MPILauncher("commandlines")
pylauncher.MPILauncher("commandlines",cores=4)
pylauncher.MPILauncher("commandlines",cores="file")
```

# Use in batch environment

- “module load pylauncher/2.0”
- Define `$TACC_PYLAUNCHER_DIR`, sets `$PYTHONPATH`
- See “examples” and “documentation” subdirectories.

# Unit tests

- The source is written for “nosetests”:  
all functions & classes that start with “test” are  
unit tests
- Currently 30 tests, some specific to TACC  
(meaning: SGE & SLURM)
- Takes about 1 minute to run, mostly because of  
“sleep” commands
- Some classes are only for testing: they generate  
dummy commands and command files

# Customizing the pylauncher

# Commandline classes

# CommandLineGenerator

- “commandline” = < Command , Corecount >
- “command” can also be “stall” (dynamic case)
- Arguments: list=[c1,c2,...] nmax=0,>0,None
  - Nmax is None (default): iterate over the list
  - Nmax>0: stop after that many
  - nmax==0: wait for finish() call
- Method: finish() stop the generator, necessary for dynamic case

# FileCommandlineGenerator

- Inherits from CommandlineGenerator
- Argument 1: filename
  - Each line is a Unix commandline
  - Line can optionally start with `< [0-9]+ ",," >`
  - Blank lines and comments with `"#"` are skipped
- Argument 2 (optional): core count

# DynamicCommandlineGenerator

- Inherits from CommandlineGenerator
- Method: `append(command[,cores])`



# Host management classes

# Node

- Description of a slot for a task
- Argument (opt): host=name, core=num, nodeid=num
- Methods: occupyWithTask(tid), release()
- Test: isfree()

# HostList

- Contains a list of host,core pairs
- Derived class SGEHostList
- Derived class SLURMHostList
- *Derive your own class for LL, PBS, &c*
- Function: TACCHostlist gives SGE or SLURM hostlist depending on hostname

# HostPool

- Maintains a pool of Node objects
- Arguments (opt): nhosts=num,  
hostlist=[h1,h2,...] or [h1;c1, h2;c2, ...]
- Argument (opt): commandprefixer= (see next)
- Method: requestNodes(n) (returns HostLocator; see next)
- Method: occupyNodes( locator, taskid )
- Method: releaseNodesByTask( taskid )

# prefixers

- Routine: takes a commandline and returns it with “ssh host” or so prefixed
- LocalPrefixer: identity
- SSHPrefixer(command,pool) returns:  
*cd curdir ; env the\_environment ; ssh pool command*
- IbrunPrefixer(command,pool) returns:  
*ibrun -o poolfirst -n poollength command*

# HostLocator

- Arguments: pool=..., offset=..., extent=...

# TACCHostPool

- Inherits from HostPool
- By default uses SSHPrefixer

# Task management classes



# Completion

- Argument (optional): taskid
- Method: attach(command)  
returns command with completion action attached
- Method: test()

# FileCompletion

- Inherits from Completion
- Completion based on creating “stamp” file
- Argument: taskid=nnn
- Argument (optional): stamproot=basename
- Argument (optional): stampdir=directory

# Task

- Argument: completion=
- Argument (optional): size=nnn (default=1)
- Argument (optional): taskid=nnn
- Method: hasCompleted()
- Method: startonnodes( hostlocator,.... )

# TaskQueue

- Method: enqueue( task )
- Method: startQueued( hostpool )
- Test: isEmpty()

# TaskGenerator

- Iterable, yields a Task or “stall”
- Argument 1: commandlinegenerator
- Argument (opt): completion = function that gives Completion objects

Tieing it all together

# LauncherJob

- Method: tick() submits queued jobs, returns:
  - “finished”: if generator stops, and all jobs finished
  - “continuing”: generator stopped, jobs still running
  - “stalling”: generator is stalled
  - “expired nnn”: reports on completed job
- Method: run() repeats tick until completed
- Argument (req'd): hostpool=
- Argument (req'd): taskgenerator=
- Argument (opt): prefixer, jobid, launcherdir

