pyina Documentation

Release 0.2.1.dev0

Mike McKerns

Jan 22, 2019

Contents:

CHAPTER 1

pyina: MPI parallel map and cluster scheduling

1.1 About Pyina

The `pyina` package provides several basic tools to make MPI-based parallel computing more accessable to the end user. The goal of `pyina` is to allow the user to extend their own code to MPI-based parallel computing with minimal refactoring.

The central element of `pyina` is the parallel map algorithm. `pyina` currently provides two strategies for executing the parallel-map, where a strategy is the algorithm for distributing the work list of jobs across the availble nodes. These strategies can be used *"in-the-raw"* (i.e. directly) to provide the map algorithm to a user's own mpi-aware code. Further, in `pyina.mpi` `pyina` provides pipe and map implementations (known as *"easy map"*) that hide the MPI internals from the user. With the *"easy map"*, the user can launch their code in parallel batch mode – using standard python and without ever having to write a line of MPI code.

There are several ways that a user would typically launch their code in parallel – directly with `mpirun` or `mpiexec`, or through the use of a scheduler such as *torque* or *slurm*. `pyina` encapsulates several of these *"launchers"*, and provides a common interface to the different methods of launching a MPI job.

`pyina` is part of `pathos`, a python framework for heterogeneous computing. `pyina` is in active development, so any user feedback, bug reports, comments, or suggestions are highly appreciated. A list of known issues is maintained at http://trac.mystic.cacr.caltech.edu/project/pathos/query.html, with a public ticket list at https://github.com/uqfoundation/pyina/issues.

1.2 Major Features

`pyina` provides a highly configurable parallel map interface to running MPI jobs, with:

- a map interface that extends the python `map` standard
- the ability to submit batch jobs to a selection of schedulers
- the ability to customize node and process launch configurations
- the ability to launch parallel MPI jobs with standard python

- ease in selecting different strategies for processing a work list

1.3 Current Release

This documentation is for version `pyina-0.2.1.dev0`.

The latest released version of `pyina` is available at:

https://pypi.org/project/pyina

`pyina` is distributed under a 3-clause BSD license.

```
>>> import pyina
>>> print (pyina.license())
```

1.4 Development Version

You can get the latest development version with all the shiny new features at:

https://github.com/uqfoundation

If you have a new contribution, please submit a pull request.

1.5 Installation

`pyina` is packaged to install from source, so you must download the tarball, unzip, and run the installer:

```
[download]
$ tar -xvzf pyina-0.2.1.tar.gz
$ cd pyina-0.2.1
$ python setup py build
$ python setup py install
```

You will be warned of any missing dependencies and/or settings after you run the "build" step above. `pyina` depends on `dill`, `pox`, `pathos`, and `mpi4py`, so you should install them first. A version of MPI must also be installed. Launchers in `pyina` that submit to a scheduler will throw errors if the underlying scheduler is not available, however a scheduler is not required for `pyina` to execute.

Alternately, `pyina` can be installed with `pip` or `easy_install`:

```
$ pip install pyina
```

1.6 Requirements

`pyina` requires:

- `python`, **version >= 2.6** or **version >= 3.3**
- `numpy`, **version >= 1.0**
- `mpi4py`, **version >= 1.3**
- `dill`, **version >= 0.2.9**

- `pox`, **version >= 0.2.5**
- `pathos`, **version >= 0.2.3**

Optional requirements:

- `setuptools`, **version >= 0.6**
- `mystic`, **version >= 0.3.3**

1.7 More Information

Probably the best way to get started is to look at the documentation at http://pyina.rtfd.io. Also see `pyina.examples` and `pyina.tests` for a set of scripts that demonstrate the configuration and launching of mpi-based parallel jobs using the *"easy map"* interface. Also see `pyina.examples_other` for a set of scripts that test the more raw internals of `pyina`. You can run the tests with `python -m pyina.tests`. A script is included for querying, setting up, and tearing down an MPI environment, see `python -m pyina` for more information. The source code is generally well documented, so further questions may be resolved by inspecting the code itself. Please feel free to submit a ticket on github, or ask a question on stackoverflow (**@Mike McKerns**). If you would like to share how you use `pyina` in your work, please send an email (to **mmckerns at uqfoundation dot org**).

Important classes and functions are found here:

- `pyina.mpi` [the map API definition]
- `pyina.schedulers` [all available schedulers]
- `pyina.launchers` [all available launchers]

Mapping strategies are found here:

- `pyina.mpi_scatter` [the scatter-gather strategy]
- `pyina.mpi_pool` [the worker pool strategy]

`pyina` also provides a convience script that helps navigate the MPI environment. This script can be run from anywhere with:

```
$ mpi_world
```

If may also be convienent to set a shell alias for the launch of 'raw' mpi-python jobs. Set something like the following (for bash):

```
$ alias mpython1='mpiexec -np 1 `which python`'
$ alias mpython2='mpiexec -np 2 `which python`'
$ ...
```

1.8 Citation

If you use `pyina` to do research that leads to publication, we ask that you acknowledge use of `pyina` by citing the following in your publication:

```
M.M. McKerns, L. Strand, T. Sullivan, A. Fang, M.A.G. Aivazis,
"Building a framework for predictive science", Proceedings of
the 10th Python in Science Conference, 2011;
http://arxiv.org/pdf/1202.1056
```

(continues on next page)

(continued from previous page)

```
Michael McKerns and Michael Aivazis,
"pathos: a framework for heterogeneous computing", 2010- ;
http://trac.mystic.cacr.caltech.edu/project/pathos
```

Please see http://trac.mystic.cacr.caltech.edu/project/pathos or http://arxiv.org/pdf/1202.1056 for further information.

citation()
print citation

license()
print license

CHAPTER 2

pyina module documentation

2.1 ez_map module

The ez_map function is a helper to parallel_map to further simplify parallel programming. Primarily ez_map provides a standard interface for parallel_map, and facilitates running parallel jobs with serial python.

2.1.1 Usage

A call to ez_map will roughly follow this example:

```
>>> # get the parallel mapper
>>> from pyina.ez_map import ez_map
>>> # construct a target function
>>> def host(id):
...     import socket
...     return "Rank: %d -- %s" % (id, socket.gethostname())
...
>>> # launch the parallel map of the target function
>>> results = ez_map(host, range(100), nodes = 10)
>>> for result in results:
...     print(result)
```

2.1.2 Implementation

A parallel application is launched by using a helper script (e.g. *ezrun.py*) as an intermediary between the MPI implementation of the parallel map (e.g. *pyina.mpi_pool.parallel_map*) and the user's serial python.

The system call that submits the mpi job is blocking. Reasons are::

1. If the main program exits before the parallel job starts, any temp files used by ez_map will be lost.
2. User is supposed to want to use the return value of the map, so blocking at the result of map shouldn't be a big hinderance.

3. If we were to allow the call to be asynchronous, we would need to implement some kind of 'deferred' mechanism or job monitoring.

Argument movement for the argument list and the returned results are pickled, while the mapped function is either saved to and imported from a temporary file (e.g. *pyina.ez_map.ez_map*), or transferred through serialization (e.g. *pyina.ez_map.ez_map2*). Either implementation has it's own advantages and weaknesses, and one mapper may succeed in a case where the other may fail.

aprun_launcher (*kdict={}*)
prepare launch for parallel execution using aprun syntax: aprun -n(nodes) (python) (program) (progargs)

Notes

run non-python commands with: {'python':'', ...} fine-grained resource utilization with: {'nodes':'4 -N 1', ...}

aprun_tasks (*nodes*)
Helper function. compute aprun task_string from node string of pattern = N[:TYPE][:ppn=P] For example, aprun_tasks("3:core4:ppn=2") yields '3 -N 2'

ez_map (*func*, **arglist*, ***kwds*)
higher-level map interface for selected mapper and launcher

maps function 'func' across arguments 'arglist'. arguments and results are stored and sent as pickled strings, while function 'func' is inspected and written as a source file to be imported.

Further Input: nodes – the number of parallel nodes launcher – the launcher object scheduler – the scheduler object mapper – the mapper object timelimit – string representation of maximum run time (e.g. '00:02') queue – string name of selected queue (e.g. 'normal')

ez_map2 (*func*, **arglist*, ***kwds*)
higher-level map interface for selected mapper and launcher

maps function 'func' across arguments 'arglist'. arguments and results are stored and sent as pickled strings, the function 'func' is also stored and sent as pickled strings. This is different than 'ez_map', in that it does not use temporary files to store the mapped function.

Further Input: nodes – the number of parallel nodes launcher – the launcher object scheduler – the scheduler object mapper – the mapper object timelimit – string representation of maximum run time (e.g. '00:02') queue – string name of selected queue (e.g. 'normal')

launch (*command*)
launch mechanism for prepared launch command

moab_launcher (*kdict={}*)
prepare launch for moab submission using srun, mpirun, aprun, or serial syntax: echo "srun -n(nodes) (python) (program) (progargs)" | msub -l nodes=(nodes) -l walltime=(timelimit) -o (outfile) -e (errfile) -q (queue) syntax: echo "%(mpirun)s -np (nodes) (python) (program) (progargs)" | msub -l nodes=(nodes) -l walltime=(timelimit) -o (outfile) -e (errfile) -q (queue) syntax: echo "aprun -n (nodes) (python) (program) (progargs)" | msub -l nodes=(nodes) -l walltime=(timelimit) -o (outfile) -e (errfile) -q (queue) syntax: echo "(python) (program) (progargs)" | msub -l nodes=(nodes) -l walltime=(timelimit) -o (outfile) -e (errfile) -q (queue)

Notes

run non-python commands with: {'python':'', ...} fine-grained resource utilization with: {'nodes':'4:ppn=1,partition=xx', ...}

class moab_scheduler
Bases: object

moab scheduler – configured for mpirun, srun, aprun, or serial

__dict__ = mappingproxy({'__weakref__': <attribute '__weakref__' of 'moab_scheduler'

__module__ = 'pyina.schedulers'

__weakref__
list of weak references to the object (if defined)

aprun = 'moab_aprun'

mpirun = 'moab_mpirun'

serial = 'moab_serial'

srun = 'moab_srun'

mpirun_launcher(*kdict={}*)
prepare launch for parallel execution using mpirun syntax: mpiexec -np (nodes) (python) (program) (progargs)

Notes

run non-python commands with: {'python':'', ...}

mpirun_tasks(*nodes*)
Helper function. compute mpirun task_string from node string of pattern = N[:TYPE][:ppn=P] For example, mpirun_tasks("3:core4:ppn=2") yields 6

serial_launcher(*kdict={}*)
prepare launch for standard execution syntax: (python) (program) (progargs)

Notes

run non-python commands with: {'python':'', ...}

srun_launcher(*kdict={}*)
prepare launch for parallel execution using srun syntax: srun -n(nodes) (python) (program) (progargs)

Notes

run non-python commands with: {'python':'', ...} fine-grained resource utilization with: {'nodes':'4 -N1', ...}

srun_tasks(*nodes*)
Helper function. compute srun task_string from node string of pattern = N[:ppn=P][,partition=X] For example, srun_tasks("3:ppn=2,partition=foo") yields '3 -N2'

torque_launcher(*kdict={}*)
prepare launch for torque submission using mpiexec, srun, aprun, or serial syntax: echo "mpiexec -np (nodes) (python) (program) (progargs)" | qsub -l nodes=(nodes) -l walltime=(timelimit) -o (outfile) -e (errfile) -q (queue) syntax: echo "srun -n(nodes) (python) (program) (progargs)" | qsub -l nodes=(nodes) -l walltime=(timelimit) -o (outfile) -e (errfile) -q (queue) syntax: echo "aprun -n (nodes) (python) (program) (progargs)" | qsub -l nodes=(nodes) -l walltime=(timelimit) -o (outfile) -e (errfile) -q (queue) syntax: echo "(python) (program) (progargs)" | qsub -l nodes=(nodes) -l walltime=(timelimit) -o (outfile) -e (errfile) -q (queue)

Notes

run non-python commands with: {'python':'', ...} fine-grained resource utilization with: {'nodes':'4:nodetype:ppn=1', ...}

class torque_scheduler

Bases: object

torque scheduler – configured for mpirun, srun, aprun, or serial

__dict__ = mappingproxy({'__weakref__': <attribute '__weakref__' of 'torque_scheduler

__module__ = 'pyina.schedulers'

__weakref__

list of weak references to the object (if defined)

aprun = 'torque_aprun'

mpirun = 'torque_mpirun'

serial = 'torque_serial'

srun = 'torque_srun'

2.2 launchers module

This module contains prepared launchers for parallel execution, including bindings to some common combinations of launchers and schedulers.

Base classes: SerialMapper - base class for pipe-based mapping with python ParallelMapper - base class for pipe-based mapping with mpi4py

Parallel launchers: Mpi - Slurm - Alps -

Pre-built combinations of the above launchers and schedulers: TorqueMpi, TorqueSlurm, MoabMpi, MoabSlurm

Pre-configured maps using the 'scatter-gather' strategy: MpiScatter, SlurmScatter, AlpsScatter, TorqueMpiScatter, TorqueSlurmScatter, MoabMpiScatter, MoabSlurmScatter

Pre-configured maps using the 'worker pool' strategy: MpiPool, SlurmPool, AlpsPool, TorqueMpiPool, TorqueSlurmPool, MoabMpiPool, MoabSlurmPool

2.2.1 Usage

A typical call to a pyina mpi map will roughly follow this example:

```
>>> # instantiate and configure a scheduler
>>> from pyina.schedulers import Torque
>>> config = {'nodes'='32:ppn=4', 'queue':'dedicated', 'timelimit':'11:59'}
>>> torque = Torque(**config)
>>>
>>> # instantiate and configure a worker pool
>>> from pyina.launchers import Mpi
>>> pool = Mpi(scheduler=torque)
>>>
>>> # do a blocking map on the chosen function
>>> results = pool.map(pow, [1,2,3,4], [5,6,7,8])
```

Several common configurations are available as pre-configured maps. The following is identical to the above example:

```
>>> # instantiate and configure a pre-configured worker pool
>>> from pyina.launchers import TorqueMpiPool
>>> config = {'nodes'='32:ppn=4', 'queue':'dedicated', 'timelimit':'11:59'}
>>> pool = TorqueMpiPool(**config)
>>>
>>> # do a blocking map on the chosen function
>>> results = pool.map(pow, [1,2,3,4], [5,6,7,8])
```

Notes

This set of parallel maps leverage the mpi4py module, and thus has many of the limitations associated with that module. The function f and the sequences in args must be serializable. The maps provided here...

<<< FIXME >>

functionality when run from a script, however are somewhat limited when used in the python interpreter. Both imported and interactively-defined functions in the interpreter session may fail due to the pool failing to find the source code for the target function. For a work-around, try:

<<< END FIXME >>>

class SerialMapper(**args*, ***kwds*)

Bases: *pyina.mpi.Mapper*

Mapper base class for pipe-based mapping with python.

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will default to 1. If source is not given, will attempt to minimially use TemporaryFiles. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If timeout is not given, will default to scheduler's timelimit or INF.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will default to 1. If source is not given, will attempt to minimially use TemporaryFiles. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If timeout is not given, will default to scheduler's timelimit or INF.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

__repr__()
Return repr(self).

_launcher(*kdict={}*)
prepare launch command for pipe-based execution

equivalent to: (python) (program) (progargs)

Notes

run non-python commands with: {'python':'', ...}

map(*func*, **args*, ***kwds*)
The function 'func', it's arguments, and the results of the map are all stored and shipped across communicators as pickled strings.

Optional Keyword Arguments:

- onall = if True, include master as a worker [default: True]

NOTE: 'onall' defaults to True for both the scatter-gather and the worker pool strategies. A worker pool with onall=True may have added difficulty in pickling functions, due to asynchronous message passing with itself.

Additional keyword arguments are passed to 'func' along with 'args'. prepare launch command for pipe-based execution

equivalent to: (python) (program) (progargs)

Notes

run non-python commands with: {'python':'', ...}

class ParallelMapper(**args*, ***kwds*)
Bases: *pyina.mpi.Mapper*

Mapper base class for pipe-based mapping with mpi4py.

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

_ParallelMapper__get_nodes()
get the number of nodes in the pool

_ParallelMapper__nodes = None

_ParallelMapper__set_nodes(*nodes*)
set the number of nodes in the pool

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

__repr__()
Return repr(self).

_launcher(*kdict={}*)
prepare launch command for pipe-based execution

equivalent to: (python) (program) (progargs)

Notes

run non-python commands with: {'python':'', ...}

map(*func*, **args*, ***kwds*)
The function 'func', it's arguments, and the results of the map are all stored and shipped across communicators as pickled strings.

Optional Keyword Arguments:

- onall = if True, include master as a worker [default: True]

NOTE: 'onall' defaults to True for both the scatter-gather and the worker pool strategies. A worker pool with onall=True may have added difficulty in pickling functions, due to asynchronous message passing with itself.

Additional keyword arguments are passed to 'func' along with 'args'. prepare launch command for pipe-based execution

equivalent to: (python) (program) (progargs)

Notes

run non-python commands with: {'python':'', ...}

njobs(*nodes*)
convert node_string intended for scheduler to int number of nodes

compute int from node string. For example, parallel.njobs("4") yields 4

nodes
get the number of nodes in the pool

class Mpi(**args*, ***kwds*)
Bases: *pyina.launchers.ParallelMapper*

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

_launcher(*kdict={}*)
prepare launch command for parallel execution using mpirun

equivalent to: mpiexec -np (nodes) (python) (program) (progargs)

Notes

run non-python commands with: {'python':'', ... }

map(*func*, **args*, ***kwds*)
The function 'func', it's arguments, and the results of the map are all stored and shipped across communicators as pickled strings.

Optional Keyword Arguments:

- onall = if True, include master as a worker [default: True]

NOTE: 'onall' defaults to True for both the scatter-gather and the worker pool strategies. A worker pool with onall=True may have added difficulty in pickling functions, due to asynchronous message passing with itself.

Additional keyword arguments are passed to 'func' along with 'args'. prepare launch command for pipe-based execution

equivalent to: (python) (program) (progargs)

Notes

run non-python commands with: {'python':'', ... } prepare launch command for parallel execution using mpirun

equivalent to: mpiexec -np (nodes) (python) (program) (progargs)

Notes

run non-python commands with: {'python':'', ... }

njobs(*nodes*)
convert node_string intended for scheduler to mpirun node_string

compute mpirun task_string from node string of pattern = N[:TYPE][:ppn=P] For example, mpirun.njobs("3:core4:ppn=2") yields 6

class Slurm(**args*, ***kwds*)
Bases: *pyina.launchers.ParallelMapper*

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

_launcher(*kdict={}*)
prepare launch for parallel execution using srun

equivalent to: srun -n(nodes) (python) (program) (progargs)

Notes

run non-python commands with: {'python':'', ... } fine-grained resource utilization with: {'nodes':'4 -N1', ... }

map(*func*, **args*, ***kwds*)
The function 'func', it's arguments, and the results of the map are all stored and shipped across communicators as pickled strings.

Optional Keyword Arguments:

- onall = if True, include master as a worker [default: True]

NOTE: 'onall' defaults to True for both the scatter-gather and the worker pool strategies. A worker pool with onall=True may have added difficulty in pickling functions, due to asynchronous message passing with itself.

Additional keyword arguments are passed to 'func' along with 'args'. prepare launch command for pipe-based execution

equivalent to: (python) (program) (progargs)

Notes

run non-python commands with: {'python':'', ... } prepare launch for parallel execution using srun

equivalent to: srun -n(nodes) (python) (program) (progargs)

Notes

run non-python commands with: {'python':'', ...} fine-grained resource utilization with: {'nodes':'4 -N1', ...}

njobs(*nodes*)
convert node_string intended for scheduler to srun node_string

compute srun task_string from node string of pattern = N[:ppn=P][,partition=X] For example, srun.njobs("3:ppn=2,partition=foo") yields '3 -N2'

class Alps(**args*, ***kwds*)
Bases: *pyina.launchers.ParallelMapper*

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

_launcher(*kdict={}*)
prepare launch for parallel execution using aprun

equivalent to: aprun -n (nodes) (python) (program) (progargs)

Notes

run non-python commands with: {'python':'', ...} fine-grained resource utilization with: {'nodes':'4 -N 1', ...}

map(*func*, **args*, ***kwds*)
The function 'func', it's arguments, and the results of the map are all stored and shipped across communicators as pickled strings.

Optional Keyword Arguments:

- onall = if True, include master as a worker [default: True]

NOTE: 'onall' defaults to True for both the scatter-gather and the worker pool strategies. A worker pool with onall=True may have added difficulty in pickling functions, due to asynchronous message passing with itself.

Additional keyword arguments are passed to 'func' along with 'args'. prepare launch command for pipe-based execution

equivalent to: (python) (program) (progargs)

Notes

run non-python commands with: {'python':'', ...} prepare launch for parallel execution using aprun

equivalent to: aprun -n (nodes) (python) (program) (progargs)

Notes

run non-python commands with: {'python':'', ... } fine-grained resource utilization with: {'nodes':'4 -N 1', ... }

njobs(*nodes*)
convert node_string intended for scheduler to aprun node_string

compute aprun task_string from node string of pattern = N[:TYPE][:ppn=P] For example, aprun.njobs("3:core4:ppn=2") yields '3 -N 2'

class MpiPool(**args*, ***kwds*)
Bases: *pyina.launchers.Mpi*

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

class MpiScatter(**args*, ***kwds*)
Bases: *pyina.launchers.Mpi*

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

class SlurmPool(**args*, ***kwds*)
Bases: *pyina.launchers.Slurm*

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

class SlurmScatter(**args*, ***kwds*)
Bases: *pyina.launchers.Slurm*

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

class AlpsPool(**args*, ***kwds*)
Bases: *pyina.launchers.Alps*

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or

$WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

class AlpsScatter(**args*, ***kwds*)
Bases: *pyina.launchers.Alps*

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

class TorqueMpi(**args*, ***kwds*)
Bases: *pyina.launchers.Mpi*

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

class TorqueSlurm(**args*, ***kwds*)
Bases: *pyina.launchers.Slurm*

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

class MoabMpi(**args*, ***kwds*)

Bases: *pyina.launchers.Mpi*

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

class MoabSlurm(**args*, ***kwds*)

Bases: *pyina.launchers.Slurm*

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

class TorqueMpiPool(**args*, ***kwds*)

Bases: *pyina.launchers.TorqueMpi*

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

class TorqueMpiScatter(**args*, ***kwds*)

Bases: *pyina.launchers.TorqueMpi*

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

class TorqueSlurmPool(**args*, ***kwds*)

Bases: *pyina.launchers.TorqueSlurm*

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

class TorqueSlurmScatter(**args*, ***kwds*)
Bases: *pyina.launchers.TorqueSlurm*

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

class MoabMpiPool(**args*, ***kwds*)
Bases: *pyina.launchers.MoabMpi*

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

class MoabMpiScatter(**args*, ***kwds*)
Bases: *pyina.launchers.MoabMpi*

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or

$WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

class MoabSlurmPool(**args*, ***kwds*)

Bases: *pyina.launchers.MoabSlurm*

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

class MoabSlurmScatter(**args*, ***kwds*)

Bases: *pyina.launchers.MoabSlurm*

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will try to grab the number of nodes from the associated scheduler, and failing will count the local cpus. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If pickle is not given, will attempt to minimially use TemporaryFiles.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.launchers'

2.3 mappers module

tiny function wrapper to make ez_map interface for mappers more standard

provides: mapper_str = mapper() interface

(for a the raw map function, use parallel_map directly)

worker_pool()
use the 'worker pool' strategy; hence one job is allocated to each worker, and the next new work item is provided when a node completes its work

scatter_gather()
use the 'scatter-gather' strategy; hence split the workload as equally as possible across all available workers in a single pass

2.4 mpi module

This module contains the base of map and pipe interfaces to the mpi4py module.

Pipe methods provided: ???

Map methods provided: map - blocking and ordered worker pool [returns: list]

Base classes: Mapper - base class for pipe-based mapping

2.4.1 Usage

A typical call to a pyina mpi map will roughly follow this example:

```
>>> # instantiate and configure a scheduler
>>> from pyina.schedulers import Torque
>>> config = {'nodes'='32:ppn=4', 'queue':'dedicated', 'timelimit':'11:59'}
>>> torque = Torque(**config)
>>>
>>> # instantiate and configure a worker pool
>>> from pyina.launchers import Mpi
>>> pool = Mpi(scheduler=torque)
>>>
>>> # do a blocking map on the chosen function
>>> print pool.map(pow, [1,2,3,4], [5,6,7,8])
```

Several common configurations are available as pre-configured maps. The following is identical to the above example:

```
>>> # instantiate and configure a pre-configured worker pool
>>> from pyina.launchers import TorqueMpiPool
>>> config = {'nodes'='32:ppn=4', 'queue':'dedicated', 'timelimit':'11:59'}
>>> pool = TorqueMpiPool(**config)
>>>
>>> # do a blocking map on the chosen function
>>> print pool.map(pow, [1,2,3,4], [5,6,7,8])
```

Notes

See pyina.launchers and pyina.schedulers for more launchers and schedulers.

_save (*boolean*)
if True, save temporary files after pickling; useful for debugging

_debug (*boolean*)
if True, print debuging info and save temporary files after pickling

class Mapper(**args*, ***kwds*)

Bases: pathos.abstract_launcher.AbstractWorkerPool

Mapper base class for pipe-based mapping.

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will default to 1. If source is not given, will attempt to minimially use TemporaryFiles. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If timeout is not given, will default to scheduler's timelimit or INF.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

_Mapper__launch(*command*)
launch mechanism for prepared launch command

_Mapper__settings()
apply default settings, then update with given settings

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers ncpus - number of worker processors servers - list of worker servers scheduler - the associated scheduler workdir - associated $WORKDIR for scratch calculations/files

Other class members: scatter - True, if uses 'scatter-gather' (instead of 'worker-pool') source - False, if minimal use of TemporaryFiles is desired timeout - number of seconds to wait for return value from scheduler

NOTE: if number of nodes is not given, will default to 1. If source is not given, will attempt to minimially use TemporaryFiles. If workdir is not given, will default to scheduler's workdir or $WORKDIR. If scheduler is not given, will default to only run on the current node. If timeout is not given, will default to scheduler's timelimit or INF.

For more details, see the docstrings for the "map" method, or the man page for the associated launcher (e.g mpirun, mpiexec).

__module__ = 'pyina.mpi'

__repr__()
Return repr(self).

_cleanup(**args*)
clean-up any additional tempfiles - path to pickled function output (e.g. 'my_results') - path to pickled function source (e.g. 'my_func.py or 'my_func.pik') - path to pickled function inputs (e.g. 'my_args.arg')

_launcher(*kdict={}*)
prepare launch command based on current settings

equivalent to: NotImplemented

_modularize(*func*)
pickle.dump function to tempfile

_modulenamemangle(*modfilename*)
mangle modulename string for use by mapper

_pickleargs(*args*, *kwds*)
pickle.dump args and kwds to tempfile

_save_in(**args*)
save input tempfiles - path to pickled function source (e.g. 'my_func.py or 'my_func.pik') - path to pickled function inputs (e.g. 'my_args.arg')

_save_out(**args*)
save output tempfiles - path to pickled function output (e.g. 'my_results')

map(*func*, **args*, ***kwds*)
The function 'func', it's arguments, and the results of the map are all stored and shipped across communicators as pickled strings.

Optional Keyword Arguments:

- onall = if True, include master as a worker [default: True]

NOTE: 'onall' defaults to True for both the scatter-gather and the worker pool strategies. A worker pool with onall=True may have added difficulty in pickling functions, due to asynchronous message passing with itself.

Additional keyword arguments are passed to 'func' along with 'args'.

settings
apply default settings, then update with given settings

2.5 mpi_pool module

MPool
alias of `multiprocess.pool.Pool`

__index(**inputs*)
build an index iterator for the given inputs

__queue(**inputs*)
iterator that groups inputs by index (i.e. [(x[0], a[0]),(x[1], a[1])])

_debug(*boolean*)
print debug statements

lookup(*inputs*, **index*)
get tuple of inputs corresponding to the given index

parallel_map(*func*, **seq*, ***kwds*)
the worker pool strategy for mpi

2.6 mpi_scatter module

__index(**inputs*)
build an index iterator for the given inputs

__queue(**inputs*)
iterator that groups inputs by index (i.e. [(x[0], a[0]),(x[1], a[1])])

balance_workload(*nproc*, *popsize*, **index*, ***kwds*)
divide popsize elements on 'nproc' chunks

nproc: int number of nodes popsize: int number of jobs index: int rank of node(s) to calculate for (using slice notation) skip: int rank of node upon which to not calculate (i.e. the master)

returns (begin, end) index vectors

get_workload(*index*, *nproc*, *popsize*, *skip=None*)
returns the workload that this processor is responsible for

index: int rank of node to calculate for nproc: int number of nodes popsize: int number of jobs skip: int rank of node upon which to not calculate (i.e. the master)

returns (begin, end) index

lookup(*inputs*, **index*)
get tuple of inputs corresponding to the given index

parallel_map(*func*, **seq*, ***kwds*)
the scatter-gather strategy for mpi

2.7 schedulers module

This module contains bindings to some common schedulers.

Base classes: Scheduler - base class for cpu cluster scheduling

Schedulers: Torque - Moab - Lsf -

2.7.1 Usage

A typical call to a pyina mpi map will roughly follow this example:

```
>>> # instantiate and configure a scheduler
>>> from pyina.schedulers import Torque
>>> config = {'nodes'='32:ppn=4', 'queue':'dedicated', 'timelimit':'11:59'}
>>> torque = Torque(**config)
>>>
>>> # instantiate and configure a worker pool
>>> from pyina.mpi import Mpi
>>> pool = Mpi(scheduler=torque)
>>>
>>> # do a blocking map on the chosen function
>>> results = pool.map(pow, [1,2,3,4], [5,6,7,8])
```

Notes

The schedulers provided here are built through pipes and not direct bindings, and are currently somewhat limited on inspecting the status of a submitted job and killing a submitted job. Currently, the use of pre-built scheduler job files are also not supported.

class Scheduler(**args*, ***kwds*)
Bases: object

Scheduler base class for cpu cluster scheduling.

Important class members: nodes - number (and potentially description) of workers queue - name of the scheduler queue [default: 'normal'] timelimit - upper limit of clocktime for each scheduled job workdir - associated $WORKDIR for scratch calculations/files

Other class members: jobfile - name of the 'job' file pyina.mpi builds for the scheduler outfile - name of the 'output' file the scheduler will write to errfile - name of the 'error' file the scheduler will write to

NOTE: The format for timelimit is typically 'HH:MM' or 'HH:MM:SS', while the format for nodes is typically 'n' or some variant of 'n:ppn=m' where 'n' is number of nodes and 'm' is processors per node. For more details, see the docstrings for the "sumbit" method, or the man page for the associated scheduler.

_Scheduler__init(**args*, ***kwds*)
default filter for __init__ inputs

_Scheduler__launch(*command*)
launch mechanism for prepared launch command

_Scheduler__nodes = 1

_Scheduler__settings()
fetch the settings for the map (from defaults and self.__dict__)

__dict__ = mappingproxy({'__weakref__': <attribute '__weakref__' of 'Scheduler' objec

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers queue - name of the scheduler queue [default: 'normal'] timelimit - upper limit of clocktime for each scheduled job workdir - associated $WORKDIR for scratch calculations/files

Other class members: jobfile - name of the 'job' file pyina.mpi builds for the scheduler outfile - name of the 'output' file the scheduler will write to errfile - name of the 'error' file the scheduler will write to

NOTE: The format for timelimit is typically 'HH:MM' or 'HH:MM:SS', while the format for nodes is typically 'n' or some variant of 'n:ppn=m' where 'n' is number of nodes and 'm' is processors per node. For more details, see the docstrings for the "sumbit" method, or the man page for the associated scheduler.

__module__ = 'pyina.schedulers'

__repr__()
Return repr(self).

__weakref__
list of weak references to the object (if defined)

_cleanup()
clean-up scheduler files (jobfile, outfile, and errfile)

_prepare()
prepare the scheduler files (jobfile, outfile, and errfile)

_submit(*command*, *kdict={}*)
prepare the given command for the scheduler

equivalent to: (command)

fetch(*outfile*, *subproc=None*)
fetch result from the results file

settings
fetch the settings for the map (from defaults and self.__dict__)

submit(*command*)
submit the given command to the scheduler

equivalent to: (command)

class Torque(**args*, ***kwds*)
Bases: *pyina.schedulers.Scheduler*

Scheduler that leverages the torque scheduler.

Important class members: nodes - number (and potentially description) of workers queue - name of the scheduler queue [default: 'normal'] timelimit - upper limit of clocktime for each scheduled job workdir - associated $WORKDIR for scratch calculations/files

Other class members: jobfile - name of the 'job' file pyina.mpi builds for the scheduler outfile - name of the 'output' file the scheduler will write to errfile - name of the 'error' file the scheduler will write to

NOTE: The format for timelimit is typically 'HH:MM' or 'HH:MM:SS', while the format for nodes is typically 'n' or some variant of 'n:ppn=m' where 'n' is number of nodes and 'm' is processors per node. For more details, see the docstrings for the "sumbit" method, or the man page for the associated scheduler.

__module__ = 'pyina.schedulers'

_submit(*command*, *kdict={}*)
prepare the given command for submission with qsub

equivalent to: echo "(command)" | qsub -l nodes=(nodes) -l walltime=(timelimit) -o (outfile) -e (errfile) -q (queue)

Notes

run non-python commands with: {'python':'', ...} fine-grained resource utilization with: {'nodes':'4:nodetype:ppn=1', ...}

submit(*command*)
submit the given command with qsub

equivalent to: echo "(command)" | qsub -l nodes=(nodes) -l walltime=(timelimit) -o (outfile) -e (errfile) -q (queue)

Notes

run non-python commands with: {'python':'', ...} fine-grained resource utilization with: {'nodes':'4:nodetype:ppn=1', ...}

class Moab(**args*, ***kwds*)
Bases: *pyina.schedulers.Scheduler*

Scheduler that leverages the moab scheduler.

Important class members: nodes - number (and potentially description) of workers queue - name of the scheduler queue [default: 'normal'] timelimit - upper limit of clocktime for each scheduled job workdir - associated $WORKDIR for scratch calculations/files

Other class members: jobfile - name of the 'job' file pyina.mpi builds for the scheduler outfile - name of the 'output' file the scheduler will write to errfile - name of the 'error' file the scheduler will write to

NOTE: The format for timelimit is typically 'HH:MM' or 'HH:MM:SS', while the format for nodes is typically 'n' or some variant of 'n:ppn=m' where 'n' is number of nodes and 'm' is processors per node. For more details, see the docstrings for the "sumbit" method, or the man page for the associated scheduler.

__module__ = 'pyina.schedulers'

_submit(*command*, *kdict={}*)
prepare the given command for submission with msub ‘ equivalent to: echo “(command)” | msub -l nodes=(nodes) -l walltime=(timelimit) -o (outfile) -e (errfile) -q (queue)

Notes

run non-python commands with: {‘python’:’‘, ...} fine-grained resource utilization with: {‘nodes’:‘4:ppn=1,partition=xx’, ...}

submit(*command*)
submit the given command with msub ‘ equivalent to: echo “(command)” | msub -l nodes=(nodes) -l walltime=(timelimit) -o (outfile) -e (errfile) -q (queue)

Notes

run non-python commands with: {‘python’:’‘, ...} fine-grained resource utilization with: {‘nodes’:‘4:ppn=1,partition=xx’, ...}

class Lsf(**args*, ***kwds*)
Bases: *pyina.schedulers.Scheduler*

Scheduler that leverages the lsf scheduler.

Important class members: nodes - number (and potentially description) of workers queue - name of the scheduler queue [default: ‘normal’] timelimit - upper limit of clocktime for each scheduled job workdir - associated $WORKDIR for scratch calculations/files

Other class members: jobfile - name of the ‘job’ file pyina.mpi builds for the scheduler outfile - name of the ‘output’ file the scheduler will write to errfile - name of the ‘error’ file the scheduler will write to

NOTE: The format for timelimit is typically ‘HH:MM’ or ‘HH:MM:SS’, while the format for nodes is typically ‘n’ or some variant of ‘n:ppn=m’ where ‘n’ is number of nodes and ‘m’ is processors per node. For more details, see the docstrings for the “sumbit” method, or the man page for the associated scheduler.

__init__(**args*, ***kwds*)

Important class members: nodes - number (and potentially description) of workers queue - name of the scheduler queue [default: ‘normal’] timelimit - upper limit of clocktime for each scheduled job workdir - associated $WORKDIR for scratch calculations/files

Other class members: jobfile - name of the ‘job’ file pyina.mpi builds for the scheduler outfile - name of the ‘output’ file the scheduler will write to errfile - name of the ‘error’ file the scheduler will write to

NOTE: The format for timelimit is typically ‘HH:MM’ or ‘HH:MM:SS’, while the format for nodes is typically ‘n’ or some variant of ‘n:ppn=m’ where ‘n’ is number of nodes and ‘m’ is processors per node. For more details, see the docstrings for the “sumbit” method, or the man page for the associated scheduler.

__module__ = 'pyina.schedulers'

_submit(*command*, *kdict={}*)
prepare the given command for submission with bsub

equivalent to: bsub -K -W (timelimit) -n (nodes) -o (outfile) -e (errfile) -q (queue) -J (progname) “(command)”

Notes

if mpich='mx', uses "-a mpich_mx mpich_mx_wrapper" instead of given launcher if mpich='gm', uses "-a mpich_gm gmmpirun_wrapper" instead of given launcher run non-python commands with: {'python':'', ...}

submit (*command*)
submit the given command with bsub

equivalent to: bsub -K -W (timelimit) -n (nodes) -o (outfile) -e (errfile) -q (queue) -J (progname) "(command)"

Notes

if mpich='mx', uses "-a mpich_mx mpich_mx_wrapper" instead of given launcher if mpich='gm', uses "-a mpich_gm gmmpirun_wrapper" instead of given launcher run non-python commands with: {'python':'', ...}

2.8 tools module

Various mpi python tools

Main function exported are::

- ensure_mpi: make sure the script is called by mpi-enabled python
- get_workload: get the workload the processor is responsible for

balance_workload (*nproc*, *popsize*, **index*, ***kwds*)
divide popsize elements on 'nproc' chunks

nproc: int number of nodes popsize: int number of jobs index: int rank of node(s) to calculate for (using slice notation) skip: int rank of node upon which to not calculate (i.e. the master)

returns (begin, end) index vectors

ensure_mpi (*size=1*, *doc=None*)
ensure that mpi-enabled python is being called with the appropriate size

inputs:

- size: minimum required size of the MPI world [default = 1]
- doc: error string to throw if size restriction is violated

get_workload (*index*, *nproc*, *popsize*, *skip=None*)
returns the workload that this processor is responsible for

index: int rank of node to calculate for nproc: int number of nodes popsize: int number of jobs skip: int rank of node upon which to not calculate (i.e. the master)

returns (begin, end) index

isoformat (*seconds*)
generate an isoformat timestring for the given time in seconds

isoseconds (*time*)
calculate number of seconds from a given isoformat timestring

lookup (*inputs*, **index*)
get tuple of inputs corresponding to the given index

mpiprint (*string=''*, *end='\n'*, *rank=0*, *comm=None*)
print the given string to the given rank

which_mpirun (*mpich=None*, *fullpath=False*)
try to autodetect an available mpi launcher

if mpich=True only look for mpich, if False only look for openmpi

which_python (*lazy=False*, *fullpath=True*)
get an invocation for this python on the execution path

which_strategy (*scatter=True*, *lazy=False*, *fullpath=True*)
try to autodetect an available strategy (scatter or pool)

CHAPTER 3

pyina scripts documentation

3.1 ezpool script

helper script for `pyina.mpi` maps using the *'worker pool'* strategy

Notes

this uses the same code as `ezscatter`, but with `pyina.mpi_pool`.

Warning: this is a helper script for `pyina.mpi.Mapper` – don't use it directly.

3.2 ezscatter script

helper script for `pyina.mpi` maps using the *'scatter gather'* strategy

Notes

this uses the same code as `ezpool`, but with `pyina.mpi_scatter`.

Warning: this is a helper script for `pyina.mpi.Mapper` – don't use it directly.

3.3 mpi_world script

setup/query/kill the MPI environment

Notes

Commandline options are:

- `--help` [prints this message]
- `--workers nodes` [set mpi world (nodes is a list of worker nodes)]
- `--fetch N` [get rank and names of 'N' worker nodes]
- `--kill` [tear down mpi world]

`'mpd &'` must be run before setting the worker nodes.

Examples:

```
$ mpi_world --workers n00 n01 n02 n03
seting up mpi...

$ mpi_world --fetch 4
Rank: 0 -- n00.borel.local
Rank: 1 -- n01.borel.local
Rank: 3 -- n03.borel.local
Rank: 2 -- n02.borel.local
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index

Symbols

A

B

C

E

F

G

I

L

M

N

P

S

T

W