pytest-mpi Documentation

Release 0.4

James Tocknell

Jun 20, 2020

Contents:

pytest-mpi provides a number of things to assist with using pytest with MPI-using code, specifically:

- Displaying of the current MPI configuration (e.g. the MPI version, the number of processes)
- Sharing temporary files/folders across the MPI processes
- Markers which allow for skipping or xfailing tests based on whether the tests are being run under MPI

Further features will be added in the future, and contribution of features is very much welcomed.

CHAPTER 1

Usage

The important thing to remember is that *pytest-mpi* assists with running tests when *pytest* is run under MPI, rather than launching *pytest* under MPI. To actually run the tests under MPI, you will want to run something like:

```
$ mpirun -n 2 python -m pytest --with-mpi
```

Note that by default the MPI tests are not run—this makes it easy to run the non-MPI parts of a test suite without having to worry about installing MPI and mpi4py.

An simple test using the *mpi* marker managed by *pytest-mpi* is:

```
import pytest
@pytest.mark.mpi
def test_size():
    from mpi4py import MPI
    comm = MPI.COMM_WORLD
    assert comm.size > 0
```

This test will be automatically be skipped unless *–with-mpi* is used. We can also specify a minimum number of processes required to run the test:

```
import pytest
@pytest.mark.mpi(min_size=2)
def test_size():
    from mpi4py import MPI
    comm = MPI.COMM_WORLD
    assert comm.size >= 2
```

There are also *mpi_skip*, for when a test should not be run under MPI (e.g. it causes a lockup or segmentation fault), and *mpi_xfail*, for when a test should succeed when run normally, but fail when run under MPI.

CHAPTER 2

Markers

`pytest.mark.`**`mpi`**`(`*min_size=None*`)`

Mark that this test must be run under MPI.

Parameters **`min_size`** (*`int`*) – Specify that this test requires at least *min_size* processes to run. If there are insufficient processes, skip this test.

For example:

```
import pytest

@pytest.mark.mpi(minsize=4)
def test_mpi_feature():
    ...
```

`pytest.mark.`**`mpi_skip`**`()`

Mark that this test should be skipped when run under MPI.

`pytest.mark.`**`mpi_xfail`**`()`

Mark that this test should be xfailed when run under MPI.

CHAPTER 3

Fixtures

`pytest_mpi.`**`mpi_file_name`**`(`*tmpdir*, *request*`)`

Provides a temporary file name which can be used under MPI from all MPI processes.

This function avoids the need to ensure that only one process handles the naming of temporary files.

`pytest_mpi.`**`mpi_tmp_path`**`(`*tmp_path*`)`

Wraps *pytest.tmp_path* so that it can be used under MPI from all MPI processes.

This function avoids the need to ensure that only one process handles the naming of temporary folders.

`pytest_mpi.`**`mpi_tmpdir`**`(`*tmpdir*`)`

Wraps *pytest.tmpdir* so that it can be used under MPI from all MPI processes.

This function avoids the need to ensure that only one process handles the naming of temporary folders.

CHAPTER 4

Changelog

4.1 0.3

- Fixed pylint failures
- Added testing of examples in documentation
- Added proper tests
- Fix bugs found via tests

4.2 0.2

- Add proper documentation of features
- Display more MPI related information on test run
- Add *mpi_skip* and *mpi_xfail* markers
- Add *mpi_tmpdir* and *mpi_tmp_path*

4.3 0.1.1

- Fix plugin as the pytest command line parsing logic needs to be outside main plugin class

4.4 0.1

Initial version

CHAPTER 5

Contributing to pytest-mpi

We welcome contributions to pytest-mpi, subject to our code of conduct whether it is improvements to the documentation or examples, bug reports or code improvements.

5.1 Reporting Bugs

Bugs should be reported to https://github.com/aragilar/pytest-mpi. Please include what version of Python this occurs on, as well as which operating system. Information about your h5py and HDF5 configuration is also helpful.

5.2 Patches and Pull Requests

The main repository is https://github.com/aragilar/pytest-mpi, please make pull requests against that repository, and the branch that pull requests should be made on is master (backporting fixes will be done separately if necessary).

5.3 Running the tests

pytest-mpi uses tox to run its tests. See https://tox.readthedocs.io/en/latest/ for more information about tox, but the simplest method is to run:

```
tox
```

in the top level of the git repository.

5.4 Making a release

Current minimal working method (this doesn't produce a release commit, deal with DOIs needing to be preregistered, not automated, not signed etc.):

1. Checkout the latest commit on the `master` branch on the main repository locally. Ensure the work directory is clean (`git purge`/`git clean -xfd`).
2. Tag this commit with an annotated tag, with the format being `v*.*.*` (`git tag -a v*.*.*`; I should sign these. . .). The tag should mention the changes in this release.
3. Push tag to github.
4. Create a release on github using the web interface, copying the content of the tag.
5. Build sdist and wheel (`python setup.py sdist bdist_wheel`), and upload to PyPI (`twine upload dist/*`).

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Index