mr4mp

Release 2.7.1

Andrei Lapets

Jun 09, 2023

CONTENTS

Thin MapReduce-like layer that wraps the Python multiprocessing library.

CHAPTER

ONE

PURPOSE

This package provides a streamlined interface for the built-in Python multiprocessing library. The interface makes it possible to parallelize in a succinct way (sometimes using only one line of code) a data workflow that can be expressed in a MapReduce-like form. More background information about this package's design and implementation, as well a detailed use case, can be found in a related article.

CHAPTER

TWO

INSTALLATION AND USAGE

This library is available as a package on PyPI:

```
python -m pip install mr4mp
```

The library can be imported in the usual way:

```
import mr4mp
```

2.1 Examples

In addition to the use case in a related article and the example below, smaller examples for each of the methods can be found in the documentation.

2.1.1 Word-Document Index

Assume there exists a collection of documents and that each document contains a collection of 7-character "words". This example demonstrates how a dictionary that associates each word to the collection of documents in which that word appears can be built by leveraging multiprocessing and the MapReduce paradigm. Suppose the function definitions below are found within a module `example.py`:

```
from random import choice
from string import ascii_lowercase
from uuid import uuid4

def word():
    """Generate a random 7-character 'word'."""
    return ''.join(choice(ascii_lowercase) for _ in range(7))

def doc():
    """Generate a random 25-word 'document' and its identifier."""
    return ([word() for _ in range(25)], uuid4())

def docs():
    """Generate list of 50 random 'documents'."""
    return [doc() for _ in range(50)]

def word_to_doc_id_dict(document):
    """Build a dictionary mapping the 'words' in a 'document' to its identifier."""
```

(continues on next page)

(continued from previous page)

```
    (words, identifier) = document
    return {w: {identifier} for w in words}

def merge_dicts(d, e):
    """Merge two dictionaries ``d`` and ``e``."""
    return {w: (d.get(w, set()) | e.get(w, set())) for w in d.keys() | e.keys()}
```

The code below (also included in `example.py`) constructs a dictionary that maps each individual word to the set of document identifiers in which that word appears. The code does so by incrementally building up larger and larger dictionaries (starting from one dictionary per document via the `word_to_doc_id_dict` function and merging them via the `merge_dicts` function), all while using the maximum number of processes supported by the system:

```
if __name__ == '__main__':
    import mr4mp
    from timeit import default_timer

    start = default_timer()
    p = mr4mp.pool()
    p.mapreduce(word_to_doc_id_dict, merge_dicts, docs())
    p.close()
    print(
        "Finished in " + str(default_timer()-start) + "s " +
        "using " + str(len(p)) + " process(es)."
    )
```

Note that any code invoking library methods must be protected inside an `if __name__ == '__main__':` block to ensure that the multiprocessing library methods can safely load the module without causing side effects. Executing the module might yield the output below:

```
python example.py
Finished in 0.664681524217187s using 2 process(es).
```

Suppose that it is explicitly indicated (by adding `processes=1` to the invocation of `pool`) that only one process can be used:

```
p = mr4mp.pool(processes=1)
```

After the above modification, executing the module might yield the output below:

```
python example.py
Finished in 2.23329004518571s using 1 process(es).
```

CHAPTER

THREE

DEVELOPMENT

All installation and development dependencies are fully specified in `pyproject.toml`. The `project.optional-dependencies` object is used to specify optional requirements for various development tasks. This makes it possible to specify additional options (such as `docs`, `lint`, and so on) when performing installation using pip:

```
python -m pip install .[docs,lint]
```

3.1 Documentation

The documentation can be generated automatically from the source files using Sphinx:

```
python -m pip install .[docs]
cd docs
sphinx-apidoc -f -E --templatedir=_templates -o _source .. && make html
```

3.2 Testing and Conventions

All unit tests are executed and their coverage is measured when using pytest (see the `pyproject.toml` file for configuration details):

```
python -m pip install .[test]
python -m pytest
```

Some unit tests are included in the module itself and can be executed using doctest:

```
python src/mr4mp/mr4mp.py -v
```

Style conventions are enforced using Pylint:

```
python -m pip install .[lint]
python -m pylint src/mr4mp test/test_mr4mp.py
```

3.3 Contributions

In order to contribute to the source code, open an issue or submit a pull request on the GitHub page for this library.

3.4 Versioning

Beginning with version 0.1.0, the version number format for this library and the changes to the library associated with version number increments conform with Semantic Versioning 2.0.0.

3.5 Publishing

This library can be published as a package on PyPI by a package maintainer. First, install the dependencies required for packaging and publishing:

```
python -m pip install .[publish]
```

Ensure that the correct version number appears in `pyproject.toml`, and that any links in this README document to the Read the Docs documentation of this package (or its dependencies) have appropriate version numbers. Also ensure that the Read the Docs project for this library has an automation rule that activates and sets as the default all tagged versions. Create and push a tag for this version (replacing `?.?.?` with the version number):

```
git tag ?.?.?
git push origin ?.?.?
```

Remove any old build/distribution files. Then, package the source into a distribution archive:

```
rm -rf build dist src/*.egg-info
python -m build --sdist --wheel .
```

Finally, upload the package distribution archive to PyPI:

```
python -m twine upload dist/*
```

3.5.1 mr4mp module

Thin MapReduce-like layer that wraps the Python multiprocessing library.

class mr4mp.mr4mp.**pool**(*processes=None*, *stages=None*, *progress=None*, *close=False*)

Bases: object

Class for a MapReduce-for-multiprocessing resource pool that can be used to run MapReduce-like workflows across multiple processes.

Parameters

- **processes** (Optional[int]) – Number of processes to allocate and to employ in executing workflows.
- **stages** (Optional[int]) – Number of stages (progress updates are provided once per stage).
- **progress** (Optional[Callable[[Iterable], Iterable]]) – Function that wraps an iterable (can be used to also report progress).

- **close** (Optional[bool]) – Flag indicating whether this instance should be closed after one workflow.

```
>>> from operator import inv, add
>>> with pool() as pool_:
...     results = pool_.mapreduce(m=inv, r=add, xs=range(3))
...     results
-6
```

__enter__()
Placeholder to enable use of `with` construct.

__exit__(**exc_details*)
Close this instance; exceptions are not suppressed.

mapreduce(*m*, *r*, *xs*, *stages=None*, *progress=None*, *close=None*)
Perform the map operation `m` and the reduce operation `r` over the supplied inputs `xs` (optionally in stages on subsequences of the data) and then release resources if directed to do so.

Parameters

- **m** (Callable[..., Any]) – Operation to be applied to each element in the input iterable.
- **r** (Callable[..., Any]) – Operation that can combine two outputs from itself, the map operation, or a mix.
- **xs** (Iterable) – Input to process using the map and reduce operations.
- **stages** (Optional[int]) – Number of stages (progress updates are provided once per stage).
- **progress** (Optional[Callable[[Iterable], Iterable]]) – Function that wraps an iterable (can be used to also report progress).
- **close** (Optional[bool]) – Flag indicating whether this instance should be closed after one workflow.

The `stages`, `progress`, and `close` parameter values each revert by default to those of this *pool* instance if they are not explicitly supplied. Supplying a value for any one of these parameters when invoking this method overrides this instance's value for that parameter *only during that invocation of the method* (this instance's value does not change).

```
>>> from operator import inv, add
>>> with pool() as pool_:
...     pool_.mapreduce(m=inv, r=add, xs=range(3))
-6
```

mapconcat(*m*, *xs*, *stages=None*, *progress=None*, *close=None*)
Perform the map operation `m` over the elements in the iterable `xs` (optionally in stages on subsequences of the data) and then release resources if directed to do so.

Parameters

- **m** (Callable[..., Sequence]) – Operation to be applied to each element in the input iterable.
- **xs** (Iterable) – Input to process using the map operation.

- **stages** (Optional[int]) – Number of stages (progress updates are provided once per stage).
- **progress** (Optional[Callable[[Iterable], Iterable]]) – Function that wraps an iterable (can be used to also report progress).
- **close** (Optional[bool]) – Flag indicating whether this instance should be closed after one workflow.

In contrast to the *pool.mapreduce* method, the map operation `m` *must return a* `Sequence`, as the results of this operation are combined using `operator.concat`.

The `stages`, `progress`, and `close` parameter values each revert by default to those of this *pool* instance if they are not explicitly supplied. Supplying a value for any one of these parameters when invoking this method overrides this instance's value for that parameter *only during that invocation of the method* (this instance's value does not change).

```
>>> with pool() as pool_:
...     pool_.mapconcat(m=tuple, xs=[[1], [2], [3]])
(1, 2, 3)
```

close()

Prevent any additional work from being added to this instance and release resources associated with this instance.

```
>>> from operator import inv
>>> pool_ = pool()
>>> pool_.close()
>>> pool_.mapconcat(m=inv, xs=range(3))
Traceback (most recent call last):
  ...
ValueError: Pool not running
```

closed()

Return a boolean indicating whether this instance has been closed.

```
>>> pool_ = pool()
>>> pool_.close()
>>> pool_.closed()
True
```

Return type bool

terminate()

Terminate the underlying `multiprocessing Pool` instance (associated resources will eventually be released, or they will be released when the instance is closed).

cpu_count()

Return number of available CPUs.

```
>>> with pool() as pool_:
...     isinstance(pool_.cpu_count(), int)
True
```

Return type int

__len__()
Return number of processes supplied as a configuration parameter when this instance was created.

```
>>> with pool(1) as pool_:
...     len(pool_)
1
```

Return type int

mr4mp.mr4mp.**mapreduce**(*m*, *r*, *xs*, *processes=None*, *stages=None*, *progress=None*)
One-shot function for performing a workflow (no explicit object management or resource allocation is required on the user's part).

Parameters

- **m** (Callable[..., Any]) – Operation to be applied to each element in the input iterable.
- **r** (Callable[..., Any]) – Operation that can combine two outputs from itself, the map operation, or a mix.
- **xs** (Iterable) – Input to process using the map and reduce operations.
- **processes** (Optional[int]) – Number of processes to allocate and to employ in executing the workflow.
- **stages** (Optional[int]) – Number of stages (progress updates are provided once per stage).
- **progress** (Optional[Callable[[Iterable], Iterable]]) – Function that wraps an iterable (can be used to also report progress).

```
>>> from operator import inv, add
>>> mapreduce(m=inv, r=add, xs=range(3))
-6
```

mr4mp.mr4mp.**mapconcat**(*m*, *xs*, *processes=None*, *stages=None*, *progress=None*)
One-shot function for applying an operation across an iterable and assembling the results back into a list (no explicit object management or resource allocation is required on the user's part).

Parameters

- **m** (Callable[..., Sequence]) – Operation to be applied to each element in the input iterable.
- **xs** (Iterable) – Input to process using the map and reduce operations.
- **processes** (Optional[int]) – Number of processes to allocate and to employ in executing the workflow.
- **stages** (Optional[int]) – Number of stages (progress updates are provided once per stage).
- **progress** (Optional[Callable[[Iterable], Iterable]]) – Function that wraps an iterable (can be used to also report progress).

In contrast to the *mapreduce* function, the map operation `m` *must return a* `Sequence`, as the results of this operation are combined using `operator.concat`.

```
>>> mapconcat(m=list, xs=[[1], [2], [3]])
[1, 2, 3]
```

PYTHON MODULE INDEX

INDEX

Symbols

C

M

P

T