caterva

Release 0.6.0

The Blosc Developers

Jul 14, 2021

CONTENTS

Python-caterva is a Python wrapper of Caterva, an open source C library specially designed to deal with large multi-dimensional, chunked, compressed datasets.

Getting Started

New to *python-caterva*? Check out the getting started guides. They contain an introduction to *python-caterva'* main concepts and an installation tutorial.

To the getting started guides

API Reference

The reference guide contains a detailed description of the *python-caterva* API. The reference describes how the functions work and which parameters can be used.

To the reference guide

Development

Saw a typo in the documentation? Want to improve existing functionalities? The contributing guidelines will guide you through the process of improving *python-caterva*.

To the development guide

Release Notes

Want to see what's new in the latest release? Check out the release notes to find out!

To the release notes

CHAPTER

ONE

GETTING STARTED

1.1 Installation

1.1.1 Pip

```
python -m pip install caterva
```

1.1.2 Source code

```
git clone --recurse-submodules https://github.com/Blosc/python-caterva
cd python-caterva
python -m pip install .
```

1.2 Tutorial

```
import caterva as cat

cat.__version__
```

```
'0.6.0'
```

1.2.1 Creating an array

```
c = cat.zeros((10000, 10000), itemsize=4, chunks=(1000, 1000), blocks=(100, 100))

c
```

```
<caterva.ndarray.NDArray at 0x7f0bc0552150>
```

1.2.2 Reading and writing data

```
import struct
import numpy as np

dtype = np.int32

c[0, :] = np.arange(10000, dtype=dtype)
c[:, 0] = np.arange(10000, dtype=dtype)
```

```
c[0, 0]
```

```
<caterva.ndarray.NDArray at 0x7f0bb00bf050>
```

```
np.array(c[0, 0]).view(dtype)
```

```
array(0, dtype=int32)
```

```
np.array(c[0, -1]).view(dtype)
```

```
array(9999, dtype=int32)
```

```
np.array(c[0, :]).view(dtype)
```

```
array([   0,    1,    2, ..., 9997, 9998, 9999], dtype=int32)
```

```
np.array(c[:, 0]).view(dtype)
```

```
array([   0,    1,    2, ..., 9997, 9998, 9999], dtype=int32)
```

```
np.array(c[:]).view(dtype)
```

```
array([[   0,    1,    2, ..., 9997, 9998, 9999],
       [   1,    0,    0, ...,    0,    0,    0],
       [   2,    0,    0, ...,    0,    0,    0],
       ...,
       [9997,    0,    0, ...,    0,    0,    0],
       [9998,    0,    0, ...,    0,    0,    0],
       [9999,    0,    0, ...,    0,    0,    0]], dtype=int32)
```

1.2.3 Persistent data

```
c1 = cat.full((1000, 1000), fill_value=b"pepe", chunks=(100, 100), blocks=(50, 50),
             urlpath="cat_tutorial.caterva")
```

```
c2 = cat.open("cat_tutorial.caterva")

c2.info
```

```
Type           : NDArray (Blosc)
Itemsize       : 4
Shape          : (1000, 1000)
Chunks         : (100, 100)
Blocks         : (50, 50)
Comp. codec    : LZ4
Comp. level    : 5
Comp. filters  : [SHUFFLE]
Comp. ratio    : 588.24
```

```
np.array(c2[0, 20:30]).view("S4")
```

```
array([b'pepe', b'pepe', b'pepe', b'pepe', b'pepe', b'pepe', b'pepe',
       b'pepe', b'pepe', b'pepe'], dtype='|S4')
```

```
import os
if os.path.exists("cat_tutorial.caterva"):
  cat.remove("cat_tutorial.caterva")
```

1.2.4 Compression params

```
b = np.arange(1000000).tobytes()

c1 = cat.from_buffer(b, shape=(1000, 1000), itemsize=8, chunks=(500, 10), blocks=(50,␣
↪10))

c1.info
```

```
Type           : NDArray (Blosc)
Itemsize       : 8
Shape          : (1000, 1000)
Chunks         : (500, 10)
Blocks         : (50, 10)
Comp. codec    : LZ4
Comp. level    : 5
Comp. filters  : [SHUFFLE]
Comp. ratio    : 6.64
```

```
c2 = c1.copy(chunks=(500, 10), blocks=(50, 10),
             codec=cat.Codec.ZSTD, clevel=9, filters=[cat.Filter.BITSHUFFLE])

c2.info
```

```
Type           : NDArray (Blosc)
Itemsize       : 8
Shape          : (1000, 1000)
Chunks         : (500, 10)
Blocks         : (50, 10)
Comp. codec    : ZSTD
Comp. level    : 9
Comp. filters  : [BITSHUFFLE]
Comp. ratio    : 20.83
```

1.2.5 Metalayers

```
from msgpack import packb, unpackb
```

```
meta = {
    "dtype": packb("i8"),
    "coords": packb([5.14, 23.])
}
```

```
c = cat.zeros((1000, 1000), 5, chunks=(100, 100), blocks=(50, 50), meta=meta)
```

```
len(c.meta)
```

```
3
```

```
c.meta.keys()
```

```
['caterva', 'dtype', 'coords']
```

```
for key in c.meta:
    print(f"{key} -> {unpackb(c.meta[key])}")
```

```
caterva -> [0, 2, [1000, 1000], [100, 100], [50, 50]]
dtype -> i8
coords -> [5.14, 23.0]
```

```
c.meta["coords"] = packb([0., 23.])
```

```
for key in c.meta:
    print(f"{key} -> {unpackb(c.meta[key])}")
```

```
caterva -> [0, 2, [1000, 1000], [100, 100], [50, 50]]
dtype -> i8
coords -> [0.0, 23.0]
```

1.2.6 Example of use

```
from PIL import Image
```

```
im = Image.open("../_static/blosc-logo_128.png")

im
```

```
meta = {"dtype": b"|u1"}

c = cat.asarray(np.array(im), chunks=(50, 50, 4), blocks=(10, 10, 4), meta=meta)

c.info
```

```
Type          : NDArray (Blosc)
Itemsize      : 1
Shape         : (70, 128, 4)
Chunks        : (50, 50, 4)
Blocks        : (10, 10, 4)
Comp. codec   : LZ4
Comp. level   : 5
Comp. filters : [SHUFFLE]
Comp. ratio   : 2.68
```

```
im2 = c[15:55, 10:35]  # Letter B

Image.fromarray(np.array(im2).view(c.meta["dtype"]))
```

CHAPTER

TWO

API REFERENCE

2.1 Global variables

`caterva.__version__`
The version of the caterva package.

class `caterva.Codec`(*value*)
Available codecs.

BLOSCLZ = 0

LZ4 = 1

LZ4HC = 2

ZLIB = 4

ZSTD = 5

class `caterva.Filter`(*value*)
Available filters.

BITSHUFFLE = 2

DELTA = 3

NOFILTER = 0

SHUFFLE = 1

TRUNC_PREC = 4

2.2 Constructors

2.2.1 Basics

empty(shape, itemsize, **kwargs)	Create an empty array.
copy(array, **kwargs)	Create a copy of an array.
from_buffer(buffer, shape, itemsize, **kwargs)	Create an array out of a buffer.
open(urlpath)	Open a new container from *urlpath*.
asarray(ndarray, **kwargs)	Convert the input to an array.

caterva.empty

`caterva.empty(shape, itemsize, **kwargs)`

Create an empty array.

Parameters

shape: tuple or list The shape for the final array.

itemsize: int The size, in bytes, of each element.

Returns

out: NDArray A *NDArray* is returned.

Other Parameters

kwargs: dict, optional Keyword arguments supported:

chunks: iterable object or None The chunk shape. If *None*, the array is stored using a non-compressed buffer. (Default *None*)

blocks: iterable object or None The block shape. If *None*, the array is stored using a non-compressed buffer. (Default *None*)

filename: str or None The name of the file to store data. If *None*, data is stored in-memory. (Default *None*)

memframe: bool If True, the array is backed by a frame in-memory. Else, by a super-chunk. (Default: *False*)

meta: dict or None A dictionary with different metalayers. One entry per metalayer:

key: bytes or str The name of the metalayer.

value: object The metalayer object that will be (de-)serialized using msgpack.

codec: `Codec` The name for the compressor codec. (Default: `Codec.LZ4`)

clevel: int (0 to 9) The compression level. 0 means no compression, and 9 maximum compression. (Default: 5)

filters: list The filter pipeline. (Default: [`Filter.SHUFFLE`])

filtersmeta: list The meta info for each filter in pipeline. (Default: [0])

nthreads: int The number of threads. (Default: 1)

usedict: bool If a dictionary should be used during compression. (Default: False)

caterva.copy

`caterva.copy(array, **kwargs)`

Create a copy of an array.

Parameters

array: NDArray The array to be copied.

Returns

out: NDArray A *NDArray* with a copy of the data.

Other Parameters

kwargs: dict, optional Keyword arguments that are supported by the `caterva.empty()` constructor.

caterva.from_buffer

`caterva.from_buffer`(*buffer*, *shape*, *itemsize*, ***kwargs*)

Create an array out of a buffer.

Parameters

buffer: bytes The buffer of the data to populate the container.

shape: tuple or list The shape for the final container.

itemsize: int The size, in bytes, of each element.

Returns

out: NDArray A *NDArray* is returned.

Other Parameters

kwargs: dict, optional Keyword arguments that are supported by the `caterva.empty()` constructor.

caterva.open

`caterva.open`(*urlpath*)

Open a new container from *urlpath*.

> **Warning:** Only one handler is supported per file.

Parameters

urlpath: str The file having a Blosc2 frame format with a Caterva metalayer on it.

Returns

out: NDArray A *NDArray* is returned.

caterva.asarray

`caterva.asarray`(*ndarray*, ***kwargs*)

Convert the input to an array.

Parameters

array: array_like An array supporting the python buffer protocol and the numpy array interface.

Returns

out: NDArray A Caterva array interpretation of *ndarray*. No copy is performed.

Other Parameters

kwargs: dict, optional Keyword arguments that are supported by the `caterva.empty()` constructor.

2.2.2 Utils

remove(urlpath)	Remove a caterva file.

caterva.remove

`caterva.`**`remove`**(*urlpath*)

Remove a caterva file.

Parameters

urlpath: String The array urlpath.

2.3 NDArray

The multidimensional data array class.

2.3.1 Attributes

itemsize	The itemsize of this container.
ndim	The number of dimensions of this container.
shape	The shape of this container.
chunks	The chunk shape of this container.
blocks	The block shape of this container.
meta	

caterva.NDArray.itemsize

`NDArray.`**`itemsize`**

The itemsize of this container.

caterva.NDArray.ndim

`NDArray.`**`ndim`**

The number of dimensions of this container.

caterva.NDArray.shape

`NDArray.`**`shape`**

The shape of this container.

caterva.NDArray.chunks

`NDArray.chunks`
The chunk shape of this container.

caterva.NDArray.blocks

`NDArray.blocks`
The block shape of this container.

caterva.NDArray.meta

property `NDArray.meta`

2.3.2 Methods

Slicing

__getitem__	Get a (multidimensional) slice as specified in key.
__setitem__	
slice	Get a (multidimensional) slice as specified in key.

caterva.NDArray.__getitem__

`NDArray.__getitem__`(*key*)
Get a (multidimensional) slice as specified in key.

Parameters

key: int, slice or sequence of slices The index for the slices to be updated. Note that step parameter is not honored yet in slices.

Returns

out: NDArray An array, stored in a non-compressed buffer, with the requested data.

caterva.NDArray.__setitem__

`NDArray.__setitem__`(*key*, *value*)

caterva.NDArray.slice

`NDArray.slice`(*key*, ***kwargs*)

Get a (multidimensional) slice as specified in key. Generalizes `__getitem__()`.

Parameters

key: int, slice or sequence of slices The index for the slices to be updated. Note that step parameter is not honored yet in slices.

Returns

out: NDArray An array with the requested data.

Other Parameters

kwargs: dict, optional Keyword arguments that are supported by the `caterva.empty()` constructor.

2.4 Metalayers

class `caterva.meta.Meta`(*ndarray*)

Class providing access to user meta on a `NDArray`. It will be available via the *.meta* property of an array.

2.4.1 Methods

`__getitem__`	Return the *item* metalayer.
`__setitem__`	Update the *key* metalayer with *value*.
`get`	Return the value for *key* if *key* is in the dictionary, else *default*.
`keys`	Return the metalayers keys
`__iter__`	Iter over the keys of the metalayers
`__contains__`	Check if the *key* metalayer exists or not

caterva.meta.Meta.__getitem__

`Meta.__getitem__`(*item*)

Return the *item* metalayer.

Parameters

item: str The name of the metalayer to return.

Returns

bytes The buffer containing the metalayer info (typically in msgpack format).

caterva.meta.Meta.__setitem__

`Meta.__setitem__(key, value)`

Update the *key* metalayer with *value*.

Parameters

key: str The name of the metalayer to update.

value: bytes The buffer containing the new content for the metalayer.

..warning: Note that the *length* of the metalayer cannot not change, else an exception will be raised.

caterva.meta.Meta.get

`Meta.get(key, default=None)`

Return the value for *key* if *key* is in the dictionary, else *default*. If *default* is not given, it defaults to `None`

caterva.meta.Meta.keys

`Meta.keys()`

Return the metalayers keys

caterva.meta.Meta.__iter__

`Meta.__iter__()`

Iter over the keys of the metalayers

caterva.meta.Meta.__contains__

`Meta.__contains__(key)`

Check if the *key* metalayer exists or not

CHAPTER

THREE

DEVELOPMENT

3.1 Contributing to python-caterva

python-caterva is a community maintained project. We want to make contributing to this project as easy and transparent as possible.

3.1.1 Asking for help

If you have a question about how to use python-caterva, please post your question on StackOverflow using the "caterva" tag.

3.1.2 Bug reports

We use GitHub issues to track public bugs. Please ensure your description is clear and has sufficient instructions to be able to reproduce the issue. The ideal report should contain the following:

1. Summarize the problem: Include details about your goal, describe expected and actual results and include any error messages.

2. Describe what you've tried: Show what you've tried, tell us what you found and why it didn't meet your needs.

3. Minimum reproducible example: Share the minimum amount of code needed to reproduce your issue. You can format the code nicely using markdown:

```
```python
import caterva as cat

...
```
```

4. Determine the environment: Indicates the python-caterva version and the operating system the code is running on.

3.1.3 Contributing to code

We actively welcome your code contributions. By contributing to python-caterva, you agree that your contributions will be licensed under the LICENSE file of the project.

Fork the repo

Make a fork of the python-caterva repository and clone it:

```
git clone https://github.com/<your-github-username>/python-caterva
```

Create your branch

Before you do any new work or submit a pull request, please open an issue on GitHub to report the bug or propose the feature you'd like to add.

Then create a new, separate branch for each piece of work you want to do.

Update docstrings

If you've changed APIs, update the involved docstrings using the doxygen format.

Run the test suite

If you have added code that needs to be tested, add the necessary tests and verify that all tests pass successfully.

3.2 Roadmap

This document lists the main goals for the upcoming python-caterva releases.

3.2.1 Features

- *Support for variable-length metalayers*. This would provide users a lot of flexibility to define their own metadata
- *Resize array dimensions*. This feature would allow Caterva to increase or decrease in size any dimension of the arrays.

3.2.2 Interoperability

- *Third-party integration*. Caterva need better integration with libraries like:
  - xarray (labeled arrays)
  - dask (computation)
  - napari (visualization)

CHAPTER

FOUR

RELEASE NOTES

4.1 Changes from 0.5.3 to 0.6.0

- Provide wheels in PyPi.
- Update caterva submodule to 0.5.0.

4.2 Changes from 0.5.1 to 0.5.3

- Fix dependencies installation issue.

4.3 Changes from 0.5.0 to 0.5.1

- Update *setup.py* and add *pyproject.toml*.

4.4 Changes from 0.4.2 to 0.5.0

- Big c-core refactoring improving the slicing performance.
- Implement *__setitem__* method for arrays to allow to update the values of the arrays.
- Use Blosc special-constructors to initialize the arrays.
- Improve the buffer and array protocols.
- Remove the data type support in order to simplify the library.

4.5 Changes from 0.4.1 to 0.4.2

- Add files in *MANIFEST.in*.

4.6 Changes from 0.4.0 to 0.4.1

- Fix invalid values for classifiers defined in *setup.py*.

4.7 Changes from 0.3.0 to 0.4.0

- Compile the package using scikit-build.
- Introduce a second level of multidimensional chunking.
- Complete API renaming.
- Support the buffer protocol and the numpy array protocol.
- Generalize the slicing.
- Make cat4py independent of numpy.

4.8 Changes from 0.2.3 to 0.3.0

- Set the development status to alpha.
- Add instructions about installing cat4py from pip.
- *getitem* and *setitem* are now special methods in *ext.Container*.
- Add new class from numpy arrays *NPArray*.
- Support for serializing/deserializing Containers to/from serialized frames (bytes).
- The *pshape* is calculated automatically if is *None*.
- Add a *.sframe* attribute for the serialized frame.
- Big refactor for more consistent inheritance among classes.
- The *from_numpy()* function always return a *NPArray* now.

4.9 Changes from 0.2.2 to 0.2.3

- Rename *MANINFEST.in* for *MANIFEST.in*.
- Fix the list of available cnames.

4.10 Changes from 0.2.1 to 0.2.2

- Added a *MANIFEST.in* for including all C-Blosc2 and Caterva sources in package.

4.11 Changes from 0.1.1 to 0.2.1

- Docstrings has been added. In addition, the documentation can be found at: https://cat4py.readthedocs.io.
- Add a *copy* parameter to *from_file()*.
- *complib* has been renamed to *cname* for compatibility with blosc-powered packages.
- The use of an itemsize different than a 2 power is allowed now.

INDEX