xarray$_{mongodb}$Documentation

Release 0.2.2.dev3+g5a2021c.d20211109

Amphora, Inc.

2021-11-09

CONTENTS

xarray_mongodb allows storing xarray objects on MongoDB. Its design is heavily influenced by GridFS.

CHAPTER

ONE

CURRENT FEATURES

- Synchronous operations with PyMongo
- asyncio support with Motor
- Units annotations with Pint
- Delayed put/get of xarray objects backed by dask. Only metadata and numpy-backed variables (e.g. indices) are written and read back at the time of graph definition.
- Support for dask distributed. Note that the full init parameters of the MongoDB client are sent over the network; this includes access credentials. One needs to make sure that network communications between dask client and scheduler and between scheduler and workers are secure.
- Data is stored on the database in a format that is agnostic to Python; this allows writing clients in different languages.

CHAPTER

TWO

UPCOMING FEATURES

- Sparse arrays with Sparse

CHAPTER

THREE

LIMITATIONS

- The Motor Tornado driver is not supported due to lack of developer interest - submissions are welcome.
- At the moment of writing, Dask and Pint are not supported at the same time due to limitations in the Pint and xarray packages.
- `attrs` are limited to the data types natively accepted by PyMongo
- Non-string xarray dimensions and variable names are not supported

CHAPTER

FOUR

QUICK START

```
>>> import pymongo
>>> import xarray
>>> import xarray_mongodb

>>> db = pymongo.MongoClient()['mydb']
>>> xdb = xarray_mongodb.XarrayMongoDB(db)
>>> a = xarray.DataArray([1, 2], dims=['x'], coords={'x': ['x1', 'x2']})
>>> _id, _ = xdb.put(a)
>>> xdb.get(_id)

<xarray.DataArray (x: 2)>
array([1, 2])
Coordinates:
  * x        (x) <U2 'x1' 'x2'
```

Dask support:

```
>>> _id, future = xdb.put(a.chunk(1))  # store metadata and numpy variables
>>> future.compute()  # store dask variables
>>> b = xdb.get(_id)  # retrieve metadata and numpy variables
>>> b

<xarray.DataArray (x: 2)>
dask.array<shape=(2,), dtype=int64, chunksize=(1,)>
Coordinates:
  * x        (x) <U2 'x1' 'x2'

>>> b.compute()  # retrieve dask variables

<xarray.DataArray (x: 2)>
array([1, 2])
Coordinates:
  * x        (x) <U2 'x1' 'x2'
```

CHAPTER

FIVE

INDEX

5.1 Installation

5.1.1 Required dependencies

- Python 3.7 or later
- MongoDB 3.6 or later
- xarray
- dask
- toolz
- PyMongo

5.1.2 Optional dependencies

- Motor for asyncio support
- Pint
- Sparse *(support not yet implemented)*

5.1.3 Minimum dependency versions

xarray_mongodb adopts a rolling policy regarding the minimum supported versions of its dependencies:

- **Python:** 42 months (NEP-29)
- **numpy:** 24 months (NEP-29)
- **pandas:** 12 months
- **pint and sparse:** very latest available versions only, until the technology based on NEP-18 will have matured. This extends to all other libraries as well when one wants to use pint or sparse.
- **all other libraries:** 6 months

You can see the actual minimum supported and tested versions:

- For using pint and/or sparse
- For everything else

5.1.4 Sphinx documentation

To build the Sphinx documentation:

1. Source conda environment
2. Move to the root directory of this project
3. Execute:

```
conda env create -n xarray_mongodb_docs --file ci/requirements-docs.yml
conda activate xarray_mongodb_docs
export PYTHONPATH=$PWD
sphinx-build -n -j auto -b html -d build/doctrees doc build/html
```

5.1.5 Testing

To run the test suite:

1. Start MongoDB on localhost (no password)
2. Source conda environment
3. Move to the root directory of this project
4. Execute:

```
conda env create -n xarray_mongodb_py37 --file ci/requirements-py37.yml
conda activate xarray_mongodb_py37
export PYTHONPATH=$PWD
py.test
```

Replace py37 with any of the environments available in the `ci` directory.

5.2 What's New

5.2.1 v0.3.0 (unreleased)

5.2.2 v0.2.2 (2021-11-05)

- Added Support for python 3.9
- Dropped support for python 3.6
- Added support for MongoDB 5.0
- Dropped support for MongoDB 3.6
- Use Sphinx 3 for documentation

5.2.3 v0.2.1 (2021-01-13)

- Support for dask 2020.12
- CI tests for MongoDB 4.4 and Python 3.9
- Use Sphinx 3 for documentation

5.2.4 v0.2.0 (2020-07-03)

Database structure changes

- Removed `units` field from the `xarray.chunks` collection. Pint must always wrap around dask, and not the other way around.
- `DataArray.attrs` was previously lost upon storage; it is now saved in the top-level `attrs` dict. (#10).
- The `attrs` dict is now omitted if empty.
- Added `attrs` dict under every element of coords and data_vars (omitted if empty).
- Embed small variables into the metadata document. Added optional `data` key to each variable on the database. Added new parameter `embed_threshold_bytes` to control how aggressive embedding should be (see *API Reference*).

Packaging changes

- xarray_mongodb now adopts a rolling *Minimum dependency versions* policy based on NEP-29.

  Increased minimum dependency versions:

Package	old	new
xarray	0.10.4	0.13
numpy	1.13	1.15
dask	1.1	1.2
pandas	0.21	0.24

- Added support for Python 3.8
- Added support for Motor on Windows (requires Motor >=2.1)
- toolz is now automatically installed by `pip install xarray_mongodb`
- Now using setuptools-scm for versioning
- Now using GitHub actions for CI
- Run all CI tests on MongoDB 3.6 and 4.2

Other changes

- Fixed error when a package importing xarray_mongodb runs `mypy --strict`
- Automatically cast scalar numpy objects (e.g. float64) wrapped by pint.Quantity to scalar ndarrays upon insertion

5.2.5 v0.1.0 (2019-03-13)

Initial release.

5.3 API Reference

class xarray_mongodb.**XarrayMongoDB**(*database*, *collection='xarray'*, ***, *chunk_size_bytes=261120*, *embed_threshold_bytes=261120*, *ureg=None*)

Synchronous driver for MongoDB to read/write xarray objects

Parameters

- **database** – pymongo.database.Database
- **collection** (*str*) – prefix of the collections to store the xarray data. Two collections will actually be created, <collection>.meta and <collection>.chunks.
- **chunk_size_bytes** (*int*) – Size of the payload in a document in the chunks collection. Not to be confused with dask chunks. dask chunks that are larger than chunk_size_bytes will be transparently split across multiple MongoDB documents.
- **embed_threshold_bytes** (*int*) – Cumulative size of variable buffers that will be embedded into the metadata documents in <collection>.meta. Buffers that exceed the threshold (starting from the largest) will be stored into the chunks documents in <collection>.chunks.

  Note:

  – Embedded variables ignore the `load` parameter of *get()*

  – dask variables are never embedded, regardless of size

  – set `embed_threshold_bytes=0` to force all buffers to be saved to <collection>.chunks, with the only exception of size zero non-dask variables

  – size zero non-dask variables are always embedded

- **ureg** (*pint.registry.UnitRegistry*) – pint registry to allow putting and getting arrays with units. If omitted, it defaults to the global registry defined with pint.set_application_registry(). If the global registry was never set, it defaults to a standard registry built with `defaults_en.txt`.

get(*_id*, *load=None*)

Read an xarray object back from MongoDB

:param ObjectId _id: MongoDB object ID, as returned by *put()*

Parameters **load** – Determines which variables to load immediately and which instead delay loading with dask. Must be one of:

None (default) Match whatever was stored with put(), including chunk sizes

True Immediately load all variables into memory. dask chunk information, if any, will be discarded.

False Only load indices in memory; delay the loading of everything else with dask.

collection of str variable names that must be immediately loaded into memory. Regardless of this, indices are always loaded. Non-existing variables are ignored. When retrieving a DataArray, you can target the data with the special hardcoded variable name `__DataArray__`.

Note: Embedded variables (see `embed_threshold_bytes`) are always loaded regardless of this flag.

Returns `xarray.DataArray` or `xarray.Dataset`, depending on what was stored with *put()*

Raises ***DocumentNotFoundError*** – _id not found in the MongoDB 'meta' collection, or one or more chunks are missing in the 'chunks' collection. This error typically happens when:

- documents were deleted from the database
- the Delayed returned by put() was never computed
- one or more chunks of the dask variables failed to compute at any point during the graph resolution

If chunks loading is delayed with dask (see 'load' parameter), this exception may be raised at compute() time.

It is possible to invoke *get()* before *put()* is computed, as long as:

- The `pass` parameter is valued None, False, or does not list any variables that were backed by dask during *put()*
- the output of *get()* is computed after the output of *put()* is computed

Warning: The dask graph (if any) underlying the returned xarray object contains full access credentials to the MongoDB server. This commands caution if one pickles it and stores it on disk, or if he sends it over the network e.g. through dask distributed.

put(*x*)

Write an xarray object to MongoDB. Variables that are backed by dask are not computed; instead their insertion in the database is delayed. All other variables are immediately inserted.

This method automatically creates an index on the 'chunks' collection if there isn't one yet.

Parameters **x** – `xarray.DataArray` or `xarray.Dataset`

Returns

Tuple of:

- MongoDB _id of the inserted object
- dask delayed object, or None if there are no variables using dask. It must be explicitly computed in order to fully store the Dataset/DataArray on the database.

> **Warning:** The dask future contains access full credentials to the MongoDB server. This commands caution if one pickles it and stores it on disk, or if he sends it over the network e.g. through dask distributed.

class xarray_mongodb.**XarrayMongoDBAsyncIO**(*database*, *collection='xarray'*, ***, *chunk_size_bytes=261120*, *embed_threshold_bytes=261120*, *ureg=None*)

asyncio driver for MongoDB to read/write xarray objects

Parameters

- **database** – motor.motor_asyncio.AsyncIOMotorDatabase
- **collection** (*str*) – See *XarrayMongoDB*
- **chunk_size_bytes** (*int*) – See *XarrayMongoDB*
- **embed_threshold_bytes** (*int*) – See *XarrayMongoDB*
- **ureg** (*pint.registry.UnitRegistry*) – See *XarrayMongoDB*

async get(*_id*, *load=None*)

Asynchronous variant of *xarray_mongodb.XarrayMongoDB.get()*

async put(*x*)

Asynchronous variant of *xarray_mongodb.XarrayMongoDB.put()*

exception xarray_mongodb.**DocumentNotFoundError**

One or more documents not found in MongoDB

5.4 Database Reference

xarray_mongodb stores data on MongoDB in a format that is agnostic to Python; this allows writing clients in different languages.

Like with GridFS, data is split across two collections, <prefix>.meta and <prefix>.chunks. By default, these are xarray.meta and xarray.chunks.

> **Note:** At the moment of writing, support for sparse arrays has not been implemented yet.

5.4.1 xarray.meta

The <prefix>.meta collection contains one document per xarray.Dataset or xarray.DataArray object, formatted as follows:

```
{
    '_id': bson.ObjectId(...),
    'attrs': bson.SON(...)  (optional),
    'chunkSize': 261120,
    'coords': bson.SON(...),
    'data_vars': bson.SON(...),
    'name': '<str>' (optional),
}
```

Where:

- `_id` is the unique ID of the xarray object
- `attrs`, `coords`, and `data_vars` are `bson.SON` objects with the same order as the dictionaries in the xarray object (note how dicts preserve insertion order starting from Python 3.6).
- `attrs` are the `Dataset.attrs` or `DataArray.attrs`, in native MongoDB format. Python object types that are not recognized by PyMongo are not supported. Omit when no attrs are available.
- chunkSize is the number of bytes stored at most in each document in the `<prefix>.chunks` collection. This is not to be confused with dask chunk size; for each dask chunk there are one or more MongoDB documents in the `<prefix>.chunks` collection (see later).
- `name` is the `DataArray.name`; omit for unnamed arrays and Datasets.
- `coords` and `data_vars` contain one key/value pair for every `xarray.Variable`, where the key is the variable name and the value is a dict defined as follows:

```
{
    'chunks': [[2, 2], [2, 2]],
    'dims': ['x'],
    'dtype': '<i8',
    'shape': [4, 4],
    'type': <'ndarray'|'COO'>,
    'attrs': bson.SON(...) (optional),
    'units': <str> (optional),

    # For ndarray only; omit in case of sparse.COO
    'data': <bytes> (optional),

    # For sparse.COO only; omit in case of ndarray
    'fill_value': <bytes> (optional),
    'sparse_data': <bytes> (optional),
    'sparse_coords': <bytes> (optional),
    'nnz': <int> (optional),
}
```

  - `chunks` are the dask chunk sizes at the moment of storing the array, or None if the variable was not backed by dask at the moment of storing the object.
  - `dims` are the names of the variable dimensions
  - `dtype` is the dtype of the numpy/dask variable, always in string format
  - `shape` is the overall shape of the numpy/dask array
  - `type` is the backend array type; `ndarray` for dense objects and `COO` for `sparse.COO` objects.
  - `attrs` are the variable attributes, if any
  - `units` is the string representation of `pint.Unit`, e.g. `kg * m /s ** 2`. The exact meaning of each symbol is deliberately omitted here and remitted to pint (or whatever other engine is used to handle units of measures). Omit for unit-less objects.
  - `data` contains the raw numpy buffer of the variable in the metadata document. It is meant to be used for small variables only. The buffer is in row-major (C) order and little endian encoding. If `data` is defined, `type` must be set to ndarray, `chunks` must always be None, and there must not be any documents for the variable in the `<prefix>.chunks` collection.
  - `fill_value` is the default value of a sparse array. It is a bytes buffer in little endian encoding of as many bytes as implied by dtype. This format allows encoding dtypes that are not native to MongoDB, e.g. complex numbers. Never present when type=ndarray.

  - `sparse_data`, `sparse_coords` and `nnz` store embedded sparse arrays. See *sparse_arrays*.

`xarray.DataArray` objects are identifiable by having exactly one variable in `data_vars`, conventionally named `__DataArray__`. Note how `DataArray.attrs` are the same as the attributes of its data variable; in xarray_mongodb they are only stored in the top-level `attrs` key (there is never a `data_vars.__DataArray__.attrs` key).

Note: When dealing with dask variables, `shape` and/or `chunks` may contain NaN instead of integer sizes when the variable size is unknown at the moment of graph definition. Also, `dtype`, `type`, and `fill_value` may potentially be wrong in the `meta` document and may be overridden by the `chunks` documents (see below).

5.4.2 xarray.chunks

The `<prefix>.chunks` collection contains the numpy data underlying the array. There is a N:1 relationship between the chunks and the meta documents.

Each document is formatted as follows:

```
{
    '_id': bson.ObjectId(...),
    'meta_id': bson.ObjectId(...),
    'name': 'variable name',
    'chunk': [0, 0],
    'dtype': '<i8',
    'shape': [1, 2]},
    'n': 0,
    'type': <'ndarray'|'COO'>,

    # For ndarray only; omit in case of sparse.COO
    'data': <bytes>,

    # For sparse.COO only; omit in case of ndarray
    'sparse_data': <bytes>,
    'sparse_coords': <bytes>',
    'nnz': <int>,
    'fill_value': <bytes>,
}
```

Where:

- `meta_id` is the Object Id of the `<prefix>.meta` collection
- `name` is the variable name, matching one defined in `<prefix>.meta`
- `chunk` is the dask chunk ID, or None for variables that were not backed by dask at the moment of storing the object
- `dtype` is the numpy dtype. It may be mismatched with, and overrides, the one defined in the `meta` collection.
- `shape` is the size of the current chunk. Unlike the `shape` and `chunks` variables defined in `<prefix>.meta`, it is never NaN.
- `n` is the sequential document counter for the current variable and chunk (see below)
- `type` is the raw array type; `ndarray` for dense arrays; `COO` for sparse ones. It may be mismatched with, and overrides, the one defined in the `meta` collection.
- `data` is the raw numpy buffer, in row-major (C) order and little endian encoding.

Since numpy arrays and dask chunks can be larger than the maximum size a MongoDB document can hold (typically 16MB), each numpy array or dask chunk may be split across multiple documents, much like it happens in GridFS. If the number of bytes in `data` would be larger than `chunkSize`, then it is split across multiple documents, with n=0, n=1, … etc. The split happens after converting the numpy array into a raw bytes buffer, and may result in having numpy points split across different documents if `chunkSize` is not an exact multiple of the `dtype` size.

Note: It is possible for all variables to be embedded into the metadata. In such a case, there won't be any documents in the chunks collection.

5.4.3 Sparse arrays

Sparse arrays (constructed using the Python class `sparse.COO`) differ from dense arrays as follows:

- In `xarray.meta`,
  - The `type` field has value `COO`
  - Extra field `fill_value` contains the value for all cells that are not explicitly listed. It is a raw binary blob in little endian encoding containing exactly one element of the indicated dtype.
- In `xarray.chunks`,
  - The `type` field has value `COO`
  - Extra field `fill_value` contains the value for all cells that are not explicitly listed
  - Extra field `nnz` is a non-negative integer (possibly zero) counting the number of cells that differ from `fill_value`.
  - There is no `data` field.
  - The `sparse_data` field contains sparse values. It is a binary blob representing a one-dimensional numpy array of the indicated dtype with as many elements as `nnz`.
  - The field `sparse_coords` is a binary blob representing a two-dimensional numpy array, with as many rows as the number of dimensions (see `shape`) and as many columns as `nnz`. It always contains unsigned integers in little endian format, regardless of the declared dtype. The word length is:
    - If max(shape) < 256, 1 byte
    - If 256 <= max(shape) < 2**16, 2 bytes
    - If 2**16 <= max(shape) < 2**32, 4 bytes
    - Otherwise, 8 bytes

    Each column of `sparse_coords` indicates the coordinates of the matching value in `sparse_data`.

See next section for examples.

When the total of the `sparse_data` and `sparse_coords` bytes exceeds `chunkSize`, then the information is split across multiple documents, as follows:

1. Documents containing slices of `sparse_data`; in all but the last one, `sparse_coords` is a bytes object of size 0
2. Documents containing slices of `sparse_coords`; in all but the first one, `sparse_data` is a bytes object of size 0

Note: When nnz=0, both data and coords are bytes objects of size 0.

5.4.4 Examples

xarray object:

```
xarray.Dataset(
    {"x": [[0, 1.1,   0],
           [0,   0, 2.2]]
    }
)
```

chunks document (dense):

```
{
    '_id': bson.ObjectId(...),
    'meta_id': bson.ObjectId(...),
    'name': 'x',
    'chunk': [0, 0],
    'dtype': '<f8',
    'shape': [2, 3],
    'n': 0,
    'type': 'ndarray',
    'data': # 48 bytes buffer that contains [0, 1.1, 0, 0, 0, 2.2]
}
```

chunks document (sparse):

```
{
    '_id': bson.ObjectId(...),
    'meta_id': bson.ObjectId(...),
    'name': 'x',
    'chunk': [0, 0],
    'dtype': '<f8',
    'shape': [2, 3]},
    'n': 0,
    'type': 'COO',
    'nnz': 2,
    'fill_value': b'\x00\x00\x00\x00\x00\x00\x00\x00',
    'sparse_data': # 16 bytes buffer that contains [1.1, 2.2]
    'sparse_coords': # 4 bytes buffer that contains [[0, 1,
                     #                               [1, 2]]
}
```

5.4.5 Indexing

Documents in `<prefix>.chunks` are identifiable by a unique functional key `(meta_id, name, chunk, n)`. The driver automatically creates a non-unique index `(meta_id, name, chunk)` on the collection. Indexing `n` is unnecessary as all the segments for a chunk are always read back together.

5.4.6 Missing data

<prefix>.chunks may miss some or all of the documents needed to reconstruct the xarray object. This typically happens when:

- the user invokes put(), but then does not compute the returned future
- some or all of the dask chunks fail to compute because of a fault at any point upstream in the dask graph
- there is a fault in MongoDB, e.g. the database becomes unreachable between the moment put() is invoked and the moment the future is computed, or if the disk becomes full.

The document in <prefix>.meta allows defining the (meta_id, name, chunk) search key for all objects in <prefix>.chunks and identify any missing documents. When a chunk is split across multiple documents, one can figure out if the retrieved documents (n=0, n=1, ...) are the complete set:

- for dense arrays (type=ndarray), the number of bytes in data must be the same as the productory of shape multiplied by dtype.size.
- for sparse arrays(type=COO), the number of bytes in data plus coords must be the same as nnz * (dtype.size + len(shape) * coords.dtype.size) where coords.dtype.size is either 1, 2, 4 or 8 depending on max(shape) (see above).

5.5 Developer notes: Pint and Sparse

Note: This page is for people contributing patches to the xarray_mongodb library itself.

If you just want to use Pint or Sparse, just make sure you satisfy the dependencies (see *Installation*) and feed the data through! Also read the documentation of the ureg parameter when initialising *XarrayMongoDB*.

For how pint and sparse objects are stored on the database, see *Database Reference*.

5.5.1 What is NEP18, and how it impacts xarray_mongodb

Several "numpy-like" libraries support a duck-type interface, specified in NEP18, so that both numpy and other NEP18-compatible libraries can transparently wrap around them.

xarray_mongodb does not, itself, use NEP18. However, it does explicitly support several data types that are possible thanks to NEP18. Namely,

- A xarray.Variable can directly wrap:
  - a numpy.ndarray, or
  - a pint.Quantity, or
  - a sparse.COO, or
  - a dask.array.Array.

  The wrapped object is accessible through the .data property.

  Note: xarray.IndexVariable wraps a pandas.Index, but the .data property converts it on the fly to a numpy.ndarray.

- A pint.Quantity can directly wrap:

  - a numpy.ndarray, or
  - a sparse.COO, or
  - a dask.array.Array.

Note: Vanilla pint can also wrap int, float, decimal.Decimal, but they are automatically transformed to numpy.ndarray as soon as xarray wraps around the Quantity.

The wrapped object is accessible through the `.magnitude` property.

- A dask.array.Array can directly wrap:
  - a numpy.ndarray, or
  - a sparse.COO.

  The wrapped object cannot be accessed until the dask graph is computed; however the object meta-data is visible without computing through the `._meta` property.

  Note: dask wrapping pint, while theoretically possible due to how NEP18 works, is not supported.

- A sparse.COO is always backed by two numpy.ndarray objects, `.data` and `.coords`.

5.5.2 Worst case

The most complicated use case that xarray_mongodb has to deal with is

1. a xarray.Variable, which wraps around
2. a pint.Quantity, which wraps around
3. a dask.array.Array, which wraps around
4. a sparse.COO, which is built on top of
5. two numpy.ndarray.

The order is always the one described above. Simpler use cases may remove any of the intermediate layers; at the top there's always has a xarray.Variable and at the bottom the data is always stored by numpy.ndarray.

Note: At the moment of writing, the example below doesn't work; see pint#878.

```
>>> import dask.array as da
>>> import numpy as np
>>> import pint
>>> import sparse
>>> import xarray
>>> ureg = pint.UnitRegistry()
>>> a = xarray.DataArray(
...     ureg.Quantity(
...         da.from_array(
...             sparse.COO.from_numpy(
...                 np.array([0, 0, 1.1])
...             )
```

(continues on next page)

(continued from previous page)

```
...         ), "kg"
...     )
... )
>>> a
<xarray.DataArray (dim_0: 3)>
dask.array<array, shape=(3,), dtype=float64, chunksize=(3,), chunktype=pint.Quantity>
Dimensions without coordinates: dim_0
>>> a.data
<Quantity(<dask.array<array, shape=(3,), dtype=float64, chunksize=(3,),
          chunktype=COO>>, 'kilogram')>
>>> a.data.magnitude
<dask.array<array, shape=(3,), dtype=float64, chunksize=(3,), chunktype=COO>
>>> a.data.units
<Unit('kilogram')>
>>> a.data.magnitude._meta
<COO: shape=(0,), dtype=float64, nnz=0, fill_value=0.0>
>>> a.data.magnitude.compute()
<COO: shape=(3,), dtype=float64, nnz=1, fill_value=0.0>
>>> a.data.magnitude.compute().data
array([1.1])
>>> a.data.magnitude.compute().coords
array([[2]])
```

5.5.3 Legacy support

There is a set of minimum required versions when pint and sparse are not involved, and a different set of much more recent ones when they are.

See also: *Minimum dependency versions*.

CHAPTER

SIX

LICENSE

xarray_mongodb is developed by Amphora and is available under the open source Apache License

The database storage specifications are patent-free and in the public domain. Anybody can write an alternative implementation; compatibility with the Python module is not enforced by law, but strongly encouraged.

INDEX