

Zarr vs. HDF5

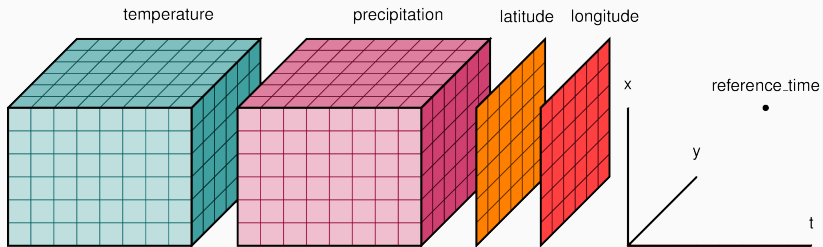
Joe Jevnik

November 4th, 2019

PyData NYC 2019

Core Concepts

Multidimensional Data



Multidimensional Data

00	01	02
10	11	12
20	21	22
30	31	32
40	41	42
50	51	52

..	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	..
----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----

Row Order

00	01	02
10	11	12
20	21	22
30	31	32
40	41	42
50	51	52

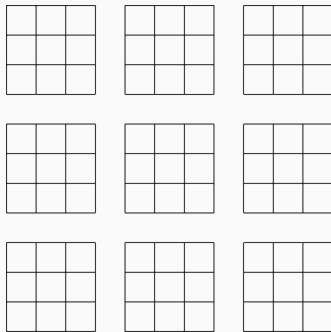
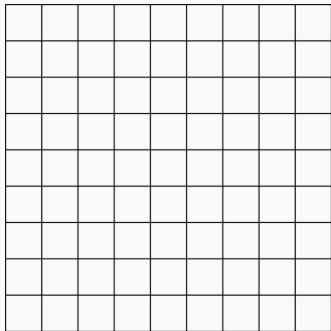
..	?	?	00	01	02	10	11	12	20	21	22	30	31	32	40	41	42	50	51	52	?	?	..
----	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	----

Column Order

00	01	02
10	11	12
20	21	22
30	31	32
40	41	42
50	51	52

..	?	?	00	10	20	30	40	50	01	11	21	31	41	51	02	12	22	32	42	52	?	?	..
----	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	----

Chunks



compression/random access trade off

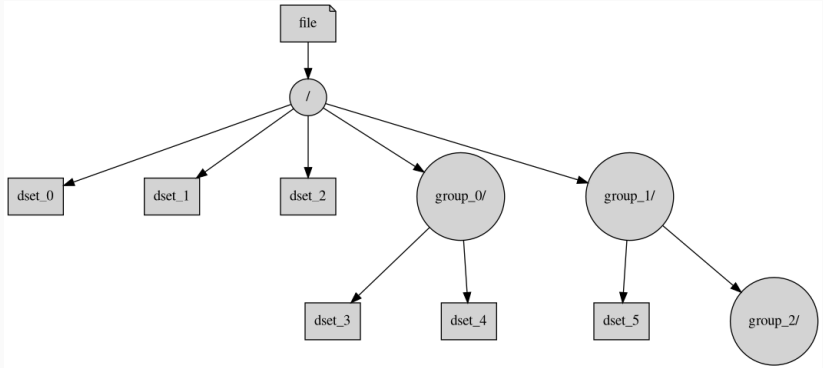
compression/random access trade off
throughput/latency access trade off

compression/random access trade off
throughput/latency access trade off
reduce io

compression/random access trade off
throughput/latency access trade off
reduce io
facilitate caching

- compression/random access trade off
- throughput/latency access trade off
- reduce io
- facilitate caching
- allow extending the shape of the dataset

Hierarchy



Definition (Dataset)

a multidimensional array

leaves of a Zarr or HDF5 tree

Definition (Dataset)

a multidimensional array

leaves of a Zarr or HDF5 tree

Definition (Group)

collection of **datasets** or **groups**

Definition (Dataset)

a multidimensional array

leaves of a Zarr or HDF5 tree

Definition (Group)

collection of **datasets** or **groups**

Definition (Node)

either a **dataset** or **group**

Definition (Attributes)

key-value data

Definition (Attributes)

key-value data

property of each **node**

Python Interface

nested dictionaries

nested dictionaries

leaves are *array-like*

nested dictionaries

leaves are *array-like*

supports numpy-style indexing

nested dictionaries

leaves are *array-like*

supports numpy-style indexing

NOTE: describes h5py, not pytables

```
>>> import zarr
>>> f = zarr.open('file.zarr')
>>> f
<zarr.hierarchy.Group '/'>
```



```
>>> import zarr
>>> f = zarr.open('file.zarr')
>>> f
<zarr.hierarchy.Group '/'>

>>> f['dset'] = np.arange(20 * 5).reshape(20, 5)
>>> f['dset']
>>> <zarr.core.Array '/dset' (20, 5) int64>
```

```
>>> import zarr
>>> f = zarr.open('file.zarr')
>>> f
<zarr.hierarchy.Group '/'>

>>> f['dset'] = np.arange(20 * 5).reshape(20, 5)
>>> f['dset']
>>> <zarr.core.Array '/dset' (20, 5) int64>

>>> f['dset'][10, 3]
53
>>> f['dset'][:]
array(...))
```

```
>>> import h5py
>>> f = h5py.File('file.h5')
>>> f
<HDF5 file "file.h5" (mode r+)>
```

```
>>> import h5py
>>> f = h5py.File('file.h5')
>>> f
<HDF5 file "file.h5" (mode r+)>

>>> f['dset'] = np.arange(20 * 5).reshape(20, 5)
>>> f['dset']
<HDF5 dataset "dset": shape (20, 5), type "<i8">
```

```
>>> import h5py
>>> f = h5py.File('file.h5')
>>> f
<HDF5 file "file.h5" (mode r+)>

>>> f['dset'] = np.arange(20 * 5).reshape(20, 5)
>>> f['dset']
<HDF5 dataset "dset": shape (20, 5), type "<i8">

>>> f['dset'][10, 3]
53
>>> f['dset'][:]
array([...])
```

`Node.attrs` to get access to attributes as dict-like object

`Node.attrs` to get access to attributes as dict-like object

`Group.create_dataset` to set chunk shape and compression

`Node.attrs` to get access to attributes as dict-like object

`Group.create_dataset` to set chunk shape and compression

`Group.create_group` to create sub-groups

`Node.attrs` to get access to attributes as dict-like object

`Group.create_dataset` to set chunk shape and compression

`Group.create_group` to create sub-groups

`Dataset.read_direct` to read into existing buffers

Making a Decision

- over 20 years old

- over 20 years old
- excellent cross language support

- over 20 years old
- excellent cross language support
- lots of existing software

- over 20 years old
- excellent cross language support
- lots of existing software
- written in (very clean) C

- over 20 years old
- excellent cross language support
- lots of existing software
- written in (very clean) C
- can be made thread safe, not thread optimal

- over 20 years old
- excellent cross language support
- lots of existing software
- written in (very clean) C
- can be made thread safe, not thread optimal
- extensible in C

- first release in 2015, 1.0 on May 17, 2016

- first release in 2015, 1.0 on May 17, 2016
- written in Python, Python oriented

- first release in 2015, 1.0 on May 17, 2016
- written in Python, Python oriented
- has specification which could be reimplemented

- first release in 2015, 1.0 on May 17, 2016
- written in Python, Python oriented
- has specification which could be reimplemented
- multithreading support

- first release in 2015, 1.0 on May 17, 2016
- written in Python, Python oriented
- has specification which could be reimplemented
- multithreading support
- extensible in Python

filters and compressors

filters and compressors

storage backends

filters and compressors

storage backends

which extensions come as part of the library itself?

filters and compressors

storage backends

which extensions come as part of the library itself?

how to extend the libraries for non-standard use cases?

filters and compressors

storage backends

which extensions come as part of the library itself?

how to extend the libraries for non-standard use cases?

ease of distributing extensions

Filters

Definition (Filter)

a function that sits between the raw data and the storage

Definition (Filter)

a function that sits between the raw data and the storage
converts data on read and write

Definition (Filter)

a function that sits between the raw data and the storage
converts data on read and write
act on one chunk at a time

Definition (Filter)

a function that sits between the raw data and the storage
converts data on read and write
act on one chunk at a time
composable

Definition (Filter)

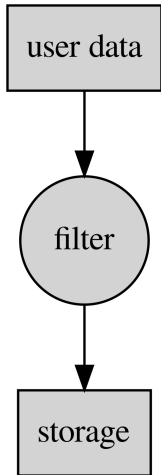
a function that sits between the raw data and the storage
converts data on read and write
act on one chunk at a time
composable
compressors

Definition (Filter)

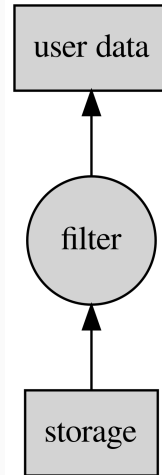
a function that sits between the raw data and the storage
converts data on read and write
act on one chunk at a time
composable
compressors
checksumming

Filters

write



read

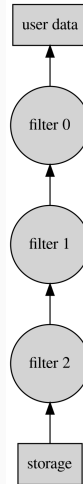


Filter Pipelines

write



read



Default Filters in HDF5

- lzf
- gzip
- szip (patent issues)
- scale-offset (lossy)
- shuffle
- fletcher32 (checksum)

Default Filters in Zarr

- `blosc`
- `lz4`
- `zstd`
- `zlib`
- `gzip`
- `bz2`
- `lzma`
- `delta`
- `fixed_scale_offset`
- `quantize` (lossy)
- `pack_bits` (boolean packing)
- `categorical` (`str` \rightarrow `int`)
- `json`, `msgpack`, `pickle`
- `vlen_string`

Writing an HDF5 Filter

```
static size_t
f(unsigned int flags,
   size_t cd_nelmts, const unsigned cd_values[],
   size_t nbytes, size_t *buf_size, void **buf) {

    if (!(flags & H5Z_FLAG_REVERSE)) {
        /* encode data */
    }
    else {
        /* decode data */
    }

}
```

Writing an HDF5 Filter

```
static size_t
f(unsigned int flags,
   size_t cd_nelmts, const unsigned cd_values[],
   size_t nbytes, size_t *buf_size, void **buf) {

    if (!(flags & H5Z_FLAG_REVERSE)) {
        /* encode data */
    }
    else {
        /* decode data */
    }

    return /* number of bytes in new buffer
           or 0 on failure */;
}
```


Writing an HDF5 Filter Cont.

```
static htri_t  
can_apply(hid_t dcpl_id, hid_t type_id, hid_t space_id) {  
    /* return -1 for error, 0 for false, or 1 for true */  
}
```

Writing an HDF5 Filter Cont.

```
const int my_filter_id = /* ... */;
static H5Z_class2_t filter = {
    .version = H5Z_CLASS_T_VERS,
    .id = my_filter_id,
    .encoder_present = 1,
    .decoder_present = 1,
    .name = "my_filter",
    .can_apply = can_apply,
    .set_local = NULL,
    .filter = f,
};

int my_filter_register() { return H5Zregister(&filter); }
```

Using a Custom HDF5 Filter

```
import ctypes
dll = ctypes.CDLL('filter.so')
assert dll.my_filter_register() == 0
filter_id = ctypes.c_int.in_dll(
    dll, 'my_filter_id',
).value
```

Using a Custom HDF5 Filter

```
import ctypes
dll = ctypes.CDLL('filter.so')
assert dll.my_filter_register() == 0
filter_id = ctypes.c_int.in_dll(
    dll, 'my_filter_id',
).value

f = h5py.File('a.h5')
dset = f.create_dataset(
    'dset',
    compression=filter_id,
    dtype='i8',
    shape=(100,),
)
```

h5py statically links to hdf5 by default

Difficulties with Custom HDF5 Filters

h5py statically links to hdf5 by default

h5py doesn't re-export H5Zregister

Difficulties with Custom HDF5 Filters

h5py statically links to hdf5 by default

h5py doesn't re-export H5Zregister

hard to thread parameters

Difficulties with Custom HDF5 Filters

h5py statically links to hdf5 by default

h5py doesn't re-export H5Zregister

hard to thread parameters

hard to manage state

Difficulties with Custom HDF5 Filters

h5py statically links to hdf5 by default

h5py doesn't re-export H5Zregister

hard to thread parameters

hard to manage state

must use malloc/free

Difficulties with Custom HDF5 Filters

h5py statically links to hdf5 by default

h5py doesn't re-export H5Zregister

hard to thread parameters

hard to manage state

must use malloc/free

can't use custom pipelines in h5py (I think)

Difficulties with Custom HDF5 Filters

h5py statically links to hdf5 by default

h5py doesn't re-export H5Zregister

hard to thread parameters

hard to manage state

must use malloc/free

can't use custom pipelines in h5py (I think)

doesn't trigger until flush

filter ids use central authority

Good Points with Custom HDF5 Filters

filter ids use central authority
can be used in different languages

Good Points with Custom HDF5 Filters

filter ids use central authority
can be used in different languages
plugin directory to find extension

Writing a Zarr Filter

```
import numcodecs
from numcodecs.abc import Codec

class MyFilter(Codec):
    codec_id = 'my_filter'

    def encode(self, buf):
        ...

    def decode(self, buf, out=None):
        ...

numcodecs.register_codec(MyFilter)
```

Using a Custom Zarr Filter

```
import numpy as np
import zarr

from my_filter import MyFilter

f = zarr.open('a.zarr', 'w')
dset = f.create_dataset(
    'dset',
    compressor=MyFilter(),
    #filters=[..., MyFilter(), ...],
    dtype='i8',
    shape=(100,),
)
```


Problems with Custom Zarr Filters

no central authority, names can collide

Problems with Custom Zarr Filters

no central authority, names can collide

require Python code or an alternative implementation

Problems with Custom Zarr Filters

no central authority, names can collide
require Python code or an alternative implementation
probably want to be written in native code anyways

Good Points with Custom Zarr Filters

written in Python

Good Points with Custom Zarr Filters

written in Python

easy distribution (Python package)

Good Points with Custom Zarr Filters

written in Python

easy distribution (Python package)

easy to pass parameters to filter

Good Points with Custom Zarr Filters

written in Python

easy distribution (Python package)

easy to pass parameters to filter

easy to manage state

Good Points with Custom Zarr Filters

written in Python

easy distribution (Python package)

easy to pass parameters to filter

easy to manage state

split API is more clear

Good Points with Custom Zarr Filters

- written in Python
- easy distribution (Python package)
- easy to pass parameters to filter
- easy to manage state
- split API is more clear
- easily works in filter pipelines

Delta of Deltas Example

```
https://github.com/llllllllllll/zarr-vs-hdf5-talk/blob/  
master/examples/delta\_of\_delta\_filter
```

Storage

abstract storage of structures

abstract storage of structures
no single file type

Storage Protocol

abstract storage of structures

no single file type

configurable

abstract storage of structures

no single file type

configurable

two kinds of data

abstract storage of structures

no single file type

configurable

two kinds of data

user data

Storage Protocol

abstract storage of structures

no single file type

configurable

two kinds of data

user data

metadata (tree structure, extra driver info)

virtual contiguous memory space

HDF5 Storage Model

virtual contiguous memory space
allocator-like

HDF5 Storage Model

virtual contiguous memory space

allocator-like

no information about semantic data

HDF5 Storage Model

virtual contiguous memory space

allocator-like

no information about semantic data

library handles caching/locking

HDF5 Storage Model

virtual contiguous memory space

allocator-like

no information about semantic data

library handles caching/locking

single threaded

based on dict from str to bytes

based on dict from str to bytes

Python mapping protocol

Zarr Storage Model

based on dict from str to bytes

Python mapping protocol

key contains semantic information

Zarr Storage Model

based on dict from str to bytes

Python mapping protocol

key contains semantic information

group/dset/0.3

Zarr Storage Model

based on dict from str to bytes

Python mapping protocol

key contains semantic information

group/dset/0.3

can manage locking, not required

Zarr Storage Model

based on dict from str to bytes

Python mapping protocol

key contains semantic information

group/dset/0.3

can manage locking, not required

composable

Default Storage Backends in HDF5

- sec2 (posix API)

Default Storage Backends in HDF5

- `sec2` (posix API)
- `windows` (Windows file API)

Default Storage Backends in HDF5

- `sec2` (posix API)
- `windows` (Windows file API)
- `stdio` (buffered, C `stdio.h`)

Default Storage Backends in HDF5

- `sec2` (posix API)
- `windows` (Windows file API)
- `stdio` (buffered, C `stdio.h`)
- `core` (in memory)

Default Storage Backends in HDF5

- `sec2` (posix API)
- `windows` (Windows file API)
- `stdio` (buffered, C `stdio.h`)
- `core` (in memory)
- `family` (directory of blocks)

Default Storage Backends in HDF5

- `sec2` (posix API)
- `windows` (Windows file API)
- `stdio` (buffered, C `stdio.h`)
- `core` (in memory)
- `family` (directory of blocks)
- `fileobj` (h5py only)

Default Storage Backends in Zarr

- `MemoryStore`
- `DirectoryStore`
- `TempStore`
- `NestedDirectoryStore`
- `ZipStore` (single file)
- `DBMStore`
- `LMDBStore`
- `SQLiteStore`
- `MongoDBStore`
- `RedisStore`
- `ABSStore` (Azure Blob Storage)
- `LRUStoreCache`
- `ConsolidatedMetadataStore`
- `S3Map` (third party)

Writing a Custom HDF5 File Driver

```
struct H5FD_class_t;
```

Writing a Custom HDF5 File Driver

```
struct H5FD_class_t;  
  
herr_t (*write)(H5FD_t* file, H5FD_mem_t type,  
                hid_t dxpl, haddr_t addr,  
                size_t size, const void* buffer);
```

Writing a Custom HDF5 File Driver

```
struct H5FD_class_t;

herr_t (*write)(H5FD_t* file, H5FD_mem_t type,
                hid_t dxpl, haddr_t addr,
                size_t size, const void* buffer);

herr_t (*read)(H5FD_t* file, H5FD_mem_t type,
               hid_t dxpl, haddr_t addr,
               size_t size, void* buffer);
```

Writing a Custom HDF5 File Driver

```
struct H5FD_class_t;

herr_t (*write)(H5FD_t* file, H5FD_mem_t type,
                hid_t dxpl, haddr_t addr,
                size_t size, const void* buffer);

herr_t (*read)(H5FD_t* file, H5FD_mem_t type,
               hid_t dxpl, haddr_t addr,
               size_t size, void* buffer);

/* 28 more member functions and  
    static member variables */
```

Using a Custom HDF5 File Driver

```
import h5py

from my_driver_as_extension_module import set_fapl

h5py.register_driver('my_driver', set_fapl)
```


Using a Custom HDF5 File Driver

```
import h5py

from my_driver_as_extension_module import set_fapl

h5py.register_driver('my_driver', set_fapl)

file = h5py.File(
    'path/to/pass/to/driver/open'
    'w',
    driver='my_driver',
    **extra_driver_kwargs,
)
```

Problems with Custom HDF5 File Drivers

huge API

Problems with Custom HDF5 File Drivers

huge API

no semantic information

Problems with Custom HDF5 File Drivers

huge API

no semantic information

hard to optimize

Problems with Custom HDF5 File Drivers

huge API

no semantic information

hard to optimize

not composable

Problems with Custom HDF5 File Drivers

huge API

no semantic information

hard to optimize

not composable

need to rely on HDF5 to do a lot of things for you

Problems with Custom HDF5 File Drivers

huge API

no semantic information

hard to optimize

not composable

need to rely on HDF5 to do a lot of things for you

small ABI incompatibility between 1.8 and 1.10

Problems with Custom HDF5 File Drivers

huge API

no semantic information

hard to optimize

not composable

need to rely on HDF5 to do a lot of things for you

small ABI incompatibility between 1.8 and 1.10

single threaded

Problems with Custom HDF5 File Drivers

huge API

no semantic information

hard to optimize

not composable

need to rely on HDF5 to do a lot of things for you

small ABI incompatibility between 1.8 and 1.10

single threaded

not many people have published file drivers

Good Points with Custom HDF5 File Drivers

`set_fapl` makes it easy to thread arguments

Good Points with Custom HDF5 File Drivers

`set_fapl` makes it easy to thread arguments
reasonable state management

Good Points with Custom HDF5 File Drivers

`set_fapl` makes it easy to thread arguments
reasonable state management
source code of built in drivers is very clear

Good Points with Custom HDF5 File Drivers

`set_fapl` makes it easy to thread arguments
reasonable state management
source code of built in drivers is very clear
can be built to minimize copies

Good Points with Custom HDF5 File Drivers

`set_fapl` makes it easy to thread arguments

reasonable state management

source code of built in drivers is very clear

can be built to minimize copies

can flatten or cut up the data to use different drivers

Good Points with Custom HDF5 File Drivers

`set_fapl` makes it easy to thread arguments

reasonable state management

source code of built in drivers is very clear

can be built to minimize copies

can flatten or cut up the data to use different drivers

64 bit memory space always

Writing a Custom Zarr Storage Backend

```
from collections.abc import MutableMapping

class MyStorageBackend(MutableMapping):
    def __getitem__(self, key):
        ...

    def __setitem__(self, key, data):
        ...

    def __delitem__(self, key):
        ...

    # ... __iter__, __len__
```


Problems with Custom Zarr Storage Backends

always need to copy on read

Good Points with Custom Zarr Storage Backends

composable

Good Points with Custom Zarr Storage Backends

composable

user defined caching, consolidation, etc.

Good Points with Custom Zarr Storage Backends

composable

user defined caching, consolidation, etc.

semantic information available

Good Points with Custom Zarr Storage Backends

composable

user defined caching, consolidation, etc.

semantic information available

written in Python

Conclusion

Zarr

- lots of modern features
- actively developed
- responsive and helpful devs
- Python focused/specific*
- easier to prototype extensions
- but probably don't need them

HDF5

- stable mature code
- cross language support
- very single threaded
- harder to write extensions
- and may need them

Thank You

- <https://github.com/llllllllllll> (10 lowercase L's)
- Example delta of deltas filter for both HDF5 and Zarr:
https://github.com/llllllllllll/zarr-vs-hdf5-talk/blob/master/examples/delta_of_delta_filter
- S3 file driver for HDF5: <https://github.com/h5s3/h5s3>
- @__qualname__