Reading NWB files on S3 with Zarr-HDF5

Ben Dichter
June 25, 2020
CatalystNeuro

Background

During the course of building data exploration tools for Neuropixel data, it has become clear that we need a way to efficiently read segments of these files in the cloud. We want to provide an interface for users to conveniently read NWB files hosted on S3 (whether via DANDI or in a private S3 space) so that data exploration tools can efficiently read small pieces of larger data files. It is possible to use h5py and fusefs to read the file directly from S3, similar to how this file is read normally when local. This approach does work, and is straightforward to implement within the current pynwb API, but is very suboptimal. Opening a file in PyNWB requires many small read commands, and in the context of S3, this means sending requests over a network. The result is that even opening a modestly sized NWB file can be prohibitively long for a simple data exploration application.

One possible solution is to use an alternative backend, such as Zarr. Zarr is a great backend, but it has two major problems for our application. The first problem is that there are data relationship primitives present in the HDF5 standard that are used by NWB and are not in the Zarr standard. This makes converting data from HDF5 to Zarr difficult. The second problem is that there is no API for Zarr in MATLAB, so a user would potentially have to convert from Zarr back to HDF5. Any imperfections in this round-trip (which are likely given problem 1) will cause issues here.

In this document, we explore an alternative solution: save the file with HDF5 and read that HDF5 file using the Zarr API (Zarr-HDF5). This would allow us to keep the advantages from both approaches: the data would remain in the exact same format, and we could have a more efficient data reading interface. In this approach, the metadata associated with the HDF5 file is stored locally in small JSON files, which minimizes the number of network calls. Preliminary steps for this process were provided by the NetCDF development group here, but this technology is still very new and not fully developed. We have assessed this technology for our needs and made enhancements where required to ensure that this works for our desired applications.

First, I will discuss the obstacles of reading NWB files in this way, which motivated targeted development of the Zarr-HDF5 toolset to overcome these issues. Then, I will discuss performance analysis comparing h5py with Zarr. Finally, I will discuss the necessary steps to integrate the new HDF5-Zarr interface with PyNWB and NWBWidgets.

Methods

Zarr and HDF5 have similar capabilities, but not all data types present in HDF5 are also present in Zarr. Our first obstacle was that some of the data types present in the NWB standard and used by Neuropixel NWB files were not supported by Zarr-HDF5.

Vectors that contain variable-length strings

This type is used in NWB for text-based columns of a DynamicTable. In the NWB files for Neuropixel datasets, the manually assigned brain area acronyms that were associated with each electrode are stored using variable length strings. The previously developed Zarr-HDF5 reader was not able to read this data type. To accomodate this data type, we developed a reader for variable-length strings and incorporated it into Zarr-HDF5.

Object references as attributes of datasets

This data type is commonly used in NWB's vector index, which allows NWB to efficiently store ragged arrays. In the NWB files for Neuropixel datasets, the dataset representing spike times uses this data type to assign spike times to specific units. Without this data type, we would not be able to determine which spike times belong to which unit.

We attended a Zarr meeting and discussed this issue with the development team. Their response was that Zarr intentionally decided to not support this type of data, because it has the possibility of introducing circular references.

Our solution here was to rewrite these objects as a string where the value of that string is the path of the link. The advantage of this approach is that resolving this reference involves the exact same syntax as before:

```
file[object_reference] => file['path/to/referenced/object']
```

However this approach does require that the user or API knows that this string represents a reference, not a normal string.

Continuous datasets

HDF5 supports datasets stored in a continuous mode, as well as datasets stored in a chunked mode. Zarr-HDF5 only supports datasets in chunked mode. Our first approach was to tell Zarr that the continuous HDF5 datasets were just one very large chunk. This did allow us to read the data values from the dataset, but caused a problem with reading small subregions of the dataset. When reading a segment of a dataset, Zarr reads all chunks in entirety that contain some of that data, so when reading a small segment of a 1-chunk dataset, Zarr reads the entire dataset. For Neuropixel data, this can take hours. Our solution was to create chunks for Zarr,

essentially pretending that the dataset is chunked. We tested the read speed across different chunk sizes to find the optimal demarcation size for these chunks.

Object references in compound datasets

Compound datasets with object references as one of their datatype components are used in NWB, but they are not supported in Zarr. IN HDF5Zarr, object references are read as uint8. They will be represented as object dtypes, for compound datasets as well. However, to be able to interpret when the object dtype is an object reference, extra data regarding the dtype is stored in zarr store, that will be also present in the exported json file as well.

Result

h5py Read Speed vs. Zarr Read Speed

In order to compare between the Zarr and h5py data reading approaches, we compared the read speed of each of the entire datasets in a single Neuropixel NWB file. For h5py, the read operation is in two steps: Get Object (i.e. `dset = file['path_to_dset']`) which gathers information about how the data is stored on the disk and Read Data (`data = dset[:]`) which reads the data values stored in the dataset into memory. h5py must perform these operations in sequence in order to read any dataset. The total time is indicated as Read Time. For Zarr, the Get Object command is very fast, because the meta-data associated with the dataset is local and parseable very quickly, so the Read Time is dominated by the Read Data step.

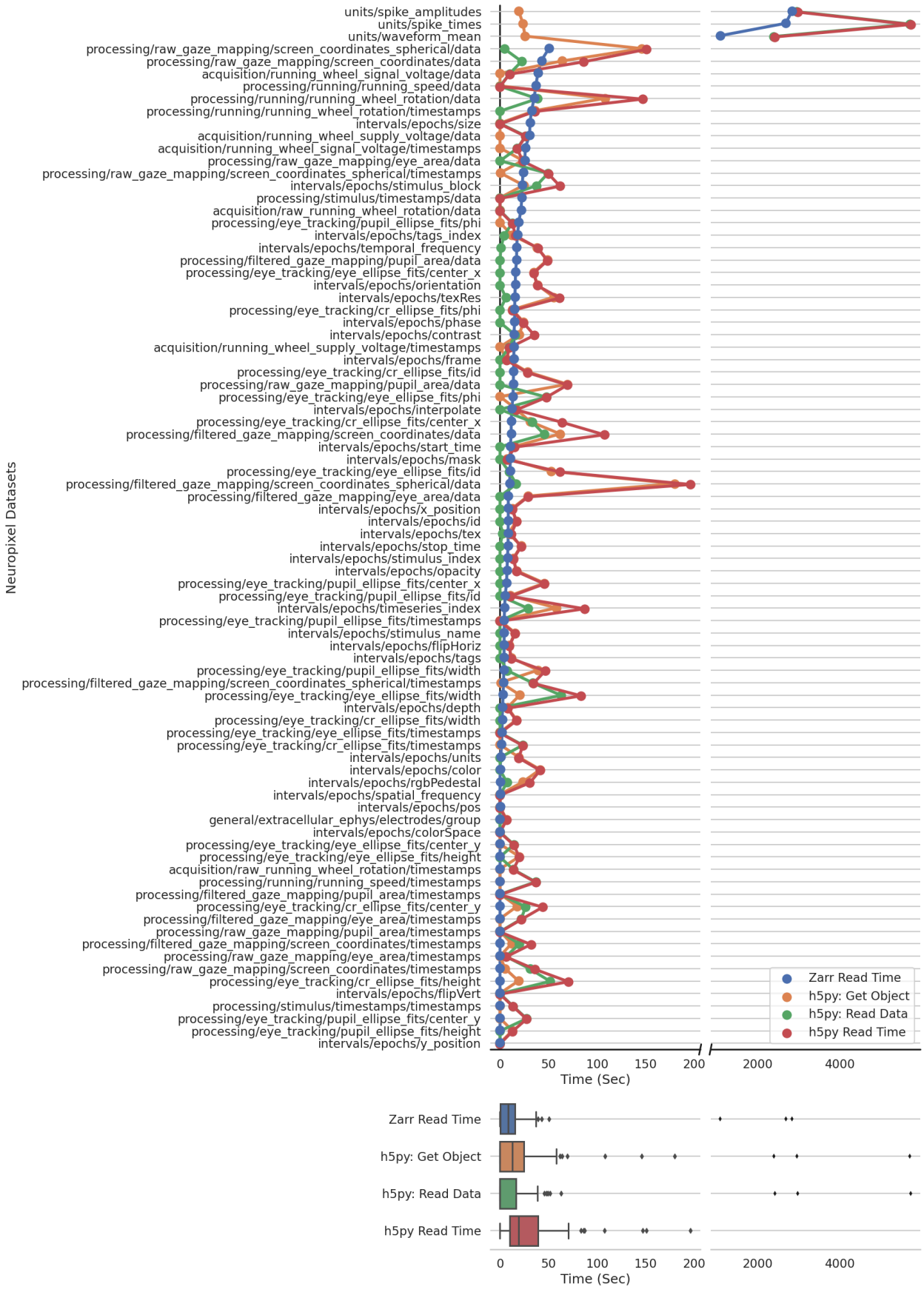
Neuropixel Datasets
units/spike_amplitudes
units/spike_times
units/waveform_mean
processing/raw_gaze_mapping/screen_coordinates_spherical/data
processing/raw_gaze_mapping/screen_coordinates/data
acquisition/running_wheel_signal_voltage/data
processing/running/running_speed/data
processing/running/running_wheel_rotation/data
processing/running/running_wheel_rotation/timestamps
intervals/epochs/size
acquisition/running_wheel_supply_voltage/data
acquisition/running_wheel_signal_voltage/timestamps
processing/raw_gaze_mapping/eye_area/data
processing/raw_gaze_mapping/screen_coordinates_spherical/timestamps
intervals/epochs/stimulus_block
processing/stimulus/timestamps/data
acquisition/raw_running_wheel_rotation/data
processing/eye_tracking/pupil_ellipse_fits/phi
intervals/epochs/tags_index
intervals/epochs/temporal_frequency
processing/filtered_gaze_mapping/pupil_area/data
processing/eye_tracking/eye_ellipse_fits/center_x
intervals/epochs/orientation
intervals/epochs/texRes
processing/eye_tracking/cr_ellipse_fits/phi
intervals/epochs/phase
intervals/epochs/contrast
acquisition/running_wheel_supply_voltage/timestamps
intervals/epochs/frame
processing/eye_tracking/cr_ellipse_fits/id
processing/raw_gaze_mapping/pupil_area/data
processing/eye_tracking/eye_ellipse_fits/phi
intervals/epochs/interpolate
processing/eye_tracking/cr_ellipse_fits/center_x
processing/filtered_gaze_mapping/screen_coordinates/data
intervals/epochs/start_time
intervals/epochs/mask
processing/eye_tracking/eye_ellipse_fits/id
processing/filtered_gaze_mapping/screen_coordinates_spherical/data
processing/filtered_gaze_mapping/eye_area/data
intervals/epochs/x_position
intervals/epochs/id
intervals/epochs/tex
intervals/epochs/stop_time
intervals/epochs/stimulus_index
intervals/epochs/opacity
processing/eye_tracking/pupil_ellipse_fits/center_x
processing/eye_tracking/pupil_ellipse_fits/id
intervals/epochs/timeseries_index
processing/eye_tracking/pupil_ellipse_fits/timestamps
intervals/epochs/stimulus_name
intervals/epochs/flipHoriz
intervals/epochs/tags
processing/eye_tracking/pupil_ellipse_fits/width
processing/filtered_gaze_mapping/screen_coordinates_spherical/timestamps
processing/eye_tracking/eye_ellipse_fits/width
intervals/epochs/depth
processing/eye_tracking/cr_ellipse_fits/width
processing/eye_tracking/eye_ellipse_fits/timestamps
processing/eye_tracking/cr_ellipse_fits/timestamps
intervals/epochs/units
intervals/epochs/color
intervals/epochs/rgbPedestal
intervals/epochs/spatial_frequency
intervals/epochs/pos
general/extracellular_ephys/electrodes/group
intervals/epochs/colorSpace
processing/eye_tracking/eye_ellipse_fits/center_y
processing/eye_tracking/eye_ellipse_fits/height
acquisition/raw_running_wheel_rotation/timestamps
processing/running/running_speed/timestamps
processing/filtered_gaze_mapping/pupil_area/timestamps
processing/eye_tracking/cr_ellipse_fits/center_y
processing/filtered_gaze_mapping/eye_area/timestamps
processing/raw_gaze_mapping/pupil_area/timestamps
processing/filtered_gaze_mapping/screen_coordinates/timestamps
processing/raw_gaze_mapping/eye_area/timestamps
processing/raw_gaze_mapping/screen_coordinates/timestamps
processing/eye_tracking/cr_ellipse_fits/height
intervals/epochs/flipVert
processing/stimulus/timestamps/timestamps
processing/eye_tracking/pupil_ellipse_fits/center_y
processing/eye_tracking/pupil_ellipse_fits/height
intervals/epochs/y_position
Zarr Read Time
h5py: Get Object
h5py: Read Data
h5py Read Time
0
50
100
150
200
2000
4000
Time (Sec)
Zarr Read Time
h5py: Get Object
h5py: Read Data
h5py Read Time
0
50
100
150
200
2000
4000
Time (Sec)

Figure 1. Comparison of data read times across Neuropixel dataset. Read times are shown for each of the datasets, ordered by Zarr Read Time. The critical comparison is blue: Zarr Read Time vs. red: h5py Read Time.

In analysis and visualization it is often necessary to read small portions of large datasets. For instance, when visualizing a small 4-second time window of an hour-long session or analyzing the spike times of a single unit among hundreds, it would be wasteful to read the entire dataset. To facilitate reading only small pieces of large datasets, large continuous HDF5 Datasets were recorded in the Zarrstore structure as if they were chunked. Several chunk sizes were explored to determine the optimal chunk size.

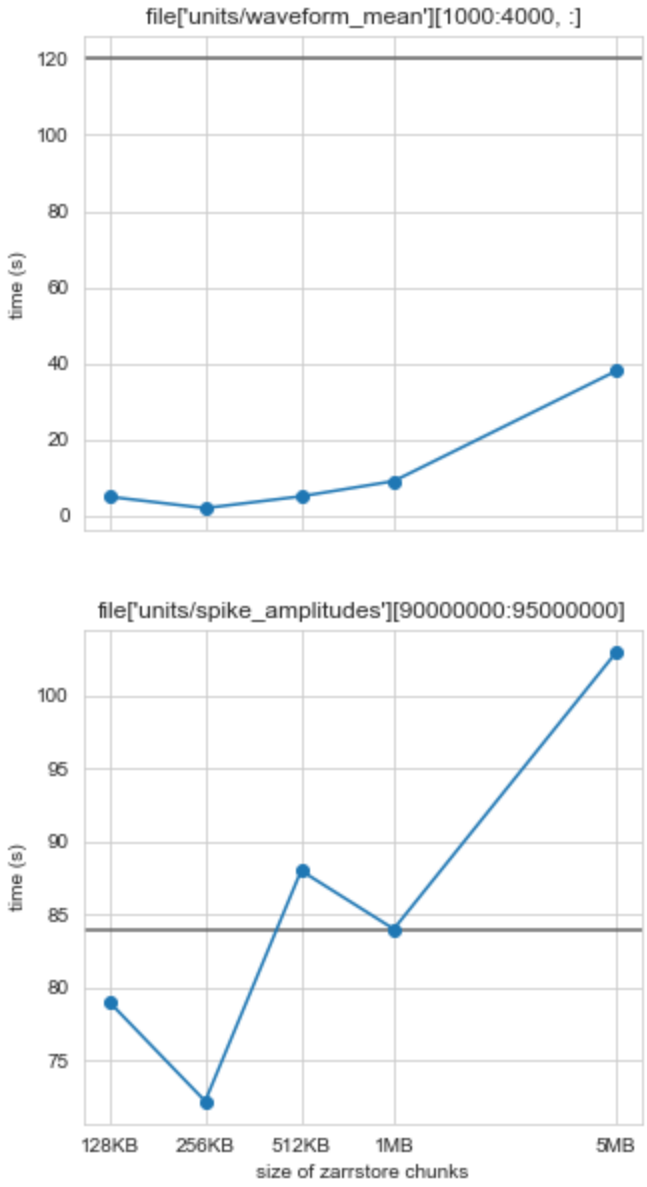

Figure 2: Slice read analysis. The read time of two data read operations is compared for different sizes of chunks in the zarrstore (blue) against the read time for h5py (grey).

Comparison of reading data segments shows that 256KB chunks are a reasonable choice for chunking in the zarrstore. It also demonstrates that, using this approach, reading small segments of data is much faster than reading the entire dataset (which is in the 1,000s of seconds for both of these datasets).

Conclusion

We have demonstrated that Zarr can be successfully used to read NWB files saved using the HDF5 format, and that it has the potential of addressing the performance needs to enable convenient exploration of Neuropixel NWB files stored in the cloud.

We have integrated this approach into PyNWB so that NWBWidgets can read data through Zarr. This allows us to efficiently read data files stored on S3, and only read the section of the files that we need.

Appendix

Open access codebase: https://github.com/catalystneuro/HDF5Zarr

HDF5Zarr can be used to read a local HDF5 file where the datasets are actually read using the Zarr library. Download example dataset from https://girder.dandiarchive.org/api/v1/item/5eda859399f25d97bd27985d/download

```
import zarr
from hdf5zarr import HDF5Zarr

file_name = 'sub-699733573_ses-715093703.nwb'
store = zarr.DirectoryStore('storezarr')
hdf5_zarr = HDF5Zarr(filename = file_name, store=store, store_mode='w',
max_chunksize=2*2**20)
zgroup = hdf5_zarr.consolidate_metadata(metadata_key = '.zmetadata')
```

Without indicating a specific zarr store, zarr uses the default `zarr.MemoryStore`. Alternatively, pass a zarr store such as:

```
store = zarr.DirectoryStore('storezarr')
hdf5_zarr = HDF5Zarr(file_name, store = store, store_mode = 'w')
```

Examine structure of file using Zarr tools:

```
# print dataset names
zgroup.tree()
# read
arr = zgroup['units/spike_times']
val = arr[0:1000]
```

Once you have a zgroup object, this object can be read by PyNWB using

```
from hdf5zarr import NWBZARRHDF5IO
io = NWBZARRHDF5IO(mode='r+', file=zgroup)
```

Export metadata from zarr store to a single json file

```
import json
metadata_file = 'metadata'
with open(metadata_file, 'w') as f:
    json.dump(zgroup.store.meta_store, f)
```

Open NWB file on remote S3 store. requires a local metadata_file, constructed in previous steps.

```
import s3fs
from hdf5zarr import NWBZARRHDF5IO


fs = s3fs.S3FileSystem(anon=True)

f =
fs.open('dandiarchive/girder-assetstore/4f/5a/4f5a24f7608041e495c85329dba318b
7', 'rb')

# import metadata from a json file
with open(metadata_file, 'r') as f:
    store = json.load(f)

hdf5_zarr = HDF5Zarr(f, store = store, store_mode = 'r')
zgroup = hdf5_zarr.zgroup
io = NWBZARRHDF5IO(mode='r', file=zgroup, load_namespaces=True)
```

Here is the entire workflow for opening a file remotely:

```
import zarr
import s3fs
from hdf5zarr import HDF5Zarr, NWBZARRHDF5IO

file_name = 'sub-699733573_ses-715093703.nwb'
store = zarr.DirectoryStore('storezarr')
hdf5_zarr = HDF5Zarr(filename = file_name, store=store, store_mode='w',
max_chunksize=2*2**20)
zgroup = hdf5_zarr.consolidate_metadata(metadata_key = '.zmetadata')

fs = s3fs.S3FileSystem(anon=True)

f =
fs.open('dandiarchive/girder-assetstore/4f/5a/4f5a24f7608041e495c85329dba318b
7', 'rb')
hdf5_zarr = HDF5Zarr(f, store = store, store_mode = 'r')
zgroup = hdf5_zarr.zgroup
io = NWBZARRHDF5IO(mode='r', file=zgroup, load_namespaces=True)
nwb = io.read()
```