

HDF5eis: A storage and input/output solution for big multidimensional time series data from environmental sensors

Malcolm C. A. White¹, Zhendong Zhang¹, Tong Bai¹, Hongrui Qiu¹, Hilary Chang¹, and Nori Nakata¹

ABSTRACT

Modern high-performance computing (HPC) tasks overwhelm conventional geophysical data formats. We describe a new data schema called HDF5eis (read H-D-F-size) for handling big multidimensional time series data from environmental sensors in HPC applications and implement a freely available Python application programming interface (API) for building and processing HDF5eis files. HDF5eis augments the popular Hierarchical Data Format 5 with a minimal set of additional conventions that facilitate fast and flexible data input and output protocols for regularly sampled (in time) data with any number of dimensions. HDF5eis supports arbitrary ancillary data (e.g., metadata) storage in columnar format or as UTF-8 encoded byte streams alongside time series data. Our HDF5eis API enables simple and efficient access to big data sets distributed across a potentially large number of small heterogeneous files through a single point of access. HDF5eis outperforms conventional seismic data formats by up to two orders of magnitude in terms of random read access times. We contribute HDF5eis as an operational tool and an experimental draft proposal that will help establish the next generation of data standards in the earth sciences.

INTRODUCTION

Seismology is a data-driven science, and the recent emergence within the earthquake seismology community of so-called large-N recording paradigms, such as distributed acoustic sensing (DAS) (e.g., Lindsey et al., 2019) and nodal geophone arrays (e.g., Lin et al., 2013; Ben-Zion et al., 2015), has simultaneously created new opportunities for discovery and inundated researchers with a glut of data. At the same time, exploration seismologists are increasingly interested in continuous recordings from the same dense recording technology for applications such as monitoring microseismicity during hydraulic fracturing treatments of unconventional hydrocarbon reservoirs (e.g., Li and van der Baan, 2021). New recording paradigms challenge the traditional distinctions between the conventions for acquiring, exchanging, archiving, and processing seismic data historically associated with either earthquake (passive source) or exploration (active source) seismology. Whereas earthquake seismologists are accustomed to data recorded by a relatively small number of sensors (small N) over a long period of time (large T), exploration seismologists are accustomed to the exact opposite: data recorded by a large number of sensors (large N) over a short period of time (small T). With continuous recordings from dense acquisition systems becoming increasingly commonplace, large-N, large-T data are posed to become the status quo in the era of "Big Data Seismology" (Arrowsmith et al., 2022).

Many of the specialized data formats at the core of seismological research today, such as (mini)SEED (Ahern et al., 2014), SAC (Goldstein et al., 2003; Goldstein and Snoke, 2005), SEG-Y (Barry et al., 1975), and SEG-D (Barry et al., 1975; Allen et al., 1994), originated in the 1970s and the 1980s following the advent of digital computing. Despite the modifications and revisions that have been made to the original format definitions, modern high-performance computing (HPC) tasks strain these conventional formats far beyond their intended uses. Data input/output (I/O) operations on conventional data formats for data-hungry processing procedures, such as training deep learning models (e.g., Ross et al., 2018; Zhu and Beroza, 2018; Mousavi et al., 2020) or the massive cross-correlation calculations necessary for seismic interferometry (e.g., Wapenaar et al., 2010a, 2010b) and matched-filter processing (e.g., Gibbons and Ringdal, 2006; Shelly et al., 2016; Ross et al., 2019; Shelly, 2020), can quickly

¹Massachusetts Institute of Technology, Department of Earth, Atmospheric and Planetary Sciences, Cambridge, Massachusetts, USA. E-mail: malcolmw@ mit.edu (corresponding author); zhendong.zhang@kaust.edu.sa; tbai@mines.edu; hongruiq@mit.edu; hilarych@mit.edu; nnakata@mit.edu. © 2023 Society of Exploration Geophysicists. All rights reserved.

Peer-reviewed code related to this article can be found at this paper's Supplemental Materials link.

Manuscript received by the Editor 8 July 2022; revised manuscript received 19 December 2022; published ahead of production 23 January 2023; published online 12 April 2023.

overwhelm computational infrastructure. Although each of these conventional formats offers critical functionality for particular tasks (e.g., acquiring real-time data in the field and archiving data with standardized metadata), seismologists need new formats if they wish to reduce the time needed to extract physical insights from large data sets.

This progression toward bigger data and more intensive computing is part of a much broader trend in science and society, which means that seismologists are fortunately not facing this technological development alone. Data scientists labor continuously to develop tools for effectively handling the increasing volume and variety of data. Such tools must be optimized to handle large volumes of data and flexible to handle a wide variety of data. However, optimization and flexibility often are opposed to one another, and an appropriate balance between them must be found (Figure 1). The Hierarchical Data Format 5 (HDF5) (The HDF Group, 1997-2022) is a prominent solution striking this balance for scientific data, although its applications are by no means restricted to scientific data. The seismological community has already leveraged the HDF5 in specialized formats such as IRIS PASS-CAL's PH5 format (Hess et al., 2017) and the Adaptable Seismic Data Format (ASDF) (Krischer et al., 2016), as has the broader earth science community in formats such as netCDF (Unidata, 2016) and NASA Earth Observatory System's Hierarchical Data Format - Earth Observing System (HDF-EOS) (Habermann, 2015). However, PH5, netCDF, and HDF-EOS are primarily intended for exchanging and archiving data. Although the ASDF is intended for HPC applications, design decisions associated with its definition (a detailed description of these is deferred to the "Motivation" section) limit its effectiveness in the analysis of multidimensional large-N, large-T data.

In this paper, we present a new data schema called HDF5eis (read H-D-F-size) that is intended to mitigate some of the limitations of the formats alluded to previously. HDF5eis is an HDF5based solution for handling big multidimensional time series data from environmental sensors in HPC applications. Our goal is to develop a workhorse for I/O intensive computing in desktop and HPC cluster environments with an application programming interface (API) simple enough for novice programmers to learn quickly. We present HDF5eis, not as a final, one-size-fits-all solution, but rather an experimental draft proposal intended to generate discus-



Figure 1. The Yin and Yang of data formatting: to gain widespread use, a data form must be sufficiently optimized for a particular set of tasks while simultaneously remaining flexible. These two objectives often are opposed to one another and must be held in proper balance.

sion of how existing technology might be leveraged by new data format standards in the earth sciences community. HDF5eis is mainly geared toward seismic data; however, we aim to keep it flexible enough to accommodate generic time series data from any environmental sensor. HDF5eis refines and improves upon some of the concepts introduced primarily by the ASDF.

This paper is structured as follows: after elaborating on our motivations for developing HDF5eis, we specify our design specifications, the schema definition, and our implementation of a Python API for building and processing HDF5eis files. Then, we describe a hypothetical experiment that demonstrates the usefulness of HDF5eis and compare its performance against conventional seismic formats using simulated data from a hypothetical survey comprising multiple acquisition systems. We conclude with a discussion of the merits and demerits of HDF5eis.

MOTIVATION

Although it may be clear that a key feature setting HDF5eis apart from conventional seismic data formats is that it leverages the HDF5 architecture, one may well wonder what sets it apart from the more comparable PH5 and ASDF, which also leverage the HDF5. Indeed HDF5eis is inspired by the ASDF in many ways and shares many similarities with it. Because PH5 is primarily designed for archiving and exchanging standardized metadata and highly structured array data recorded during temporary deployments — in fact, data often are converted from PH5 to SEG-Y for analysis — we focus the remainder of this section on what differentiates HDF5eis from the ASDF. Next, we outline key lessons learned from the ASDF and how we improve upon those aspects.

First and most importantly, the atomic storage unit for regularly sampled time series data in the ASDF is a 1D data array - each 1D array is considered a discrete data object representing a single-channel time series. Although many contiguous data channels can be stored in a single ASDF file, each one is stored as a separate data object. This severely limits the efficiency of certain access patterns to multidimensional array data. Consider, for example, reading a short segment of data recorded by a DAS fiber with 1000 channels. Because the ASDF stores each data channel in an independent 1D array, contiguous channels are liable to be stored on distant disk sectors, which increases the amount of time spent seeking the relevant data to fulfill multichannel data queries. However, if those data were stored contiguously on disk in a single 2D array, I/O primitives could be used much more efficiently when reading data. The HDF5's "hyperslab selection" functionality provides a seamless mechanism for efficiently accessing contiguous data volumes in such a scenario. A secondary side effect of atomizing data at such a fine level of granularity is that ASDF files bloat with excessive metadata maintained by the HDF5, which hinders performance. This is only a problem, however, when tens of thousands of data segments are stored in a single file. Multidimensional arrays are supported by the ASDF as so-called "auxiliary data", but these are — as the name implies — secondary data structures rather than the emphasis of the format. HDF5eis makes multidimensional arrays the atomic storage unit around which the schema is defined.

Second, to retrieve data from an ASDF file, one must exhaustively traverse every data object in the file and parse the associated string identifiers to check for query matches. This is fine in principle but becomes prohibitively slow in practice, particularly when using the Python API to access data in a file with highly fragmented data (e.g., a large number of channels or discontinuous/gappy data). This problem is exacerbated by the highly granular structure of the ASDF files. HDF5eis mitigates this inefficiency by (1) enabling the grouping of coordinate data channels and (2) maintaining a tabular index of file contents, which enables efficient regular-expression parsing to rapidly fulfill queries. We elaborate on this indexing feature subsequently.

Finally, the ASDF operates on each file independently. Data can thus be either (1) stored in a single large file — this option provides a convenient access pattern, but the large files can become unwieldy — or (2) stored in many relatively small files — this option yields manageable file sizes but increases the user's bookkeeping burden. However, HDF5eis leverages HDF5's hierarchical design and external linking functionality to enable external links to an arbitrary number of files from a single master file. This master file serves as a convenient, single point of access to data distributed across a potentially large number of files. This helps maintain manageable file sizes for large data sets without sacrificing the convenience of having a single point of access. Furthermore, a single data file can be linked to multiple master files, enabling users to conveniently create multiple virtual data sets with different subsets of data without having to copy any of them.

One salient novelty of the ASDF that we retain and extend is the notion of integrating existing formats for storing ancillary data alongside primitive time series data. The ASDF implements this feature by defining storage structures for standard formats based on the Extended Markup Language (XML), namely, QuakeML (Schorlemmer et al., 2011), StationXML, and Seis-Prov (Krischer et al., 2015) for storing earthquake catalog data, network metadata, and data provenance, respectively. However, these formats are particular to the earthquake seismology domain and are not amenable to the broad spectrum of the alternative data domains in the earth sciences. We extend this feature to enable the storage of data in a columnar format and any UTF-8 encoded byte stream. Supporting columnar data provides a natural interface to the "DataFrame" structures that are the cornerstone of powerful data science software such as pandas for Python, DataFrames.jl for Julia, and data.frame in R, which are well suited for operating on relational data and support optimized data queries, data transformations, and statistical analyses. Supporting arbitrary UTF-8 encoded data implies that any text-based format, such as simple text README data and derivatives of XML or JSON, can be integrated into HDF5eis files.

DESIGN SPECIFICATIONS

In service of our primary objective — i.e., to efficiently handle big multidimensional time series data from environmental sensors in HPC applications — we design HDF5eis to

- 1) store primitive time series data with arbitrary dimensionality,
- store arbitrary ancillary data in a columnar format or as UTF-8 encoded byte streams,
- provide a single point of access to diverse data distributed across many files, and
- simultaneously leverage existing technology and minimize external dependencies.

It should be noted that the base HDF5 API satisfies our design specifications. However, by imposing a minimal set of additional conventions, we can significantly increase its efficiency for a particular class of tasks. By this, we do not mean that we will make HDF5 itself more efficient. Rather, we aim to define a minimal set of additional structures that will make the HDF5's efficiency more accessible to earth scientists. Defining and imposing a set of conventions on the HDF5 enables the implementation of universal I/O protocols and reusable, shareable code that will help earth scientists shift their focus from the technicalities of data management to the scientific research problems to which the data pertain.

By leveraging the HDF5 software stack, HDF5eis freely inherits

- various lossless, in-flight compression filters (e.g., GZIP, LZF, and SZIP);
- an interface for defining custom compression filters (e.g., STEIM and STEIM2);
- advanced data indexing and memory caching algorithms for accelerating I/O operations;
- seamless functionality for integrating structured array data with arbitrary dimensionality; and
- hierarchical data structures that permit symbolic links to external data objects.

SCHEMA DEFINITION

Having outlined our motivations and design specifications, we now move to the HDF5eis schema definition. We wish to make a distinction at this point between the schema *definition* and *implementation*. The schema definition is a set of rules defining which data structures are permissible, whereas its implementation is a set of software for tangibly manifesting and manipulating those structures. This section outlines the HDF5eis schema definition. All computational variables in the following description take string values unless otherwise noted.

At the highest level, data in each HDF5eis file are separated into three main HDF5 Group objects (groups): (1) /timeseries, (2) /products, and (3) /metadata (Figure 2). The root group possesses an HDF5 Attribute object (attribute) called __VERSION which records the schema version of the file for control against future schema revisions. The structure and function of each group are elaborated on in a top-down fashion next.

The /timeseries group

Primitive time series data are stored in a nested hierarchy beneath the /timeseries group. The HDF5 Data set object (data set) at the terminus of each branch in the hierarchy stores a single, continuous block of multidimensional time series data with fixed sampling intervals along each dimension. These data sets can have any number of dimensions; however, the time axis must be oriented along the last axis. Each data set is addressed by a unique identifier formatted similar to /timeseries/{tag}/__{start_time}Z__{end_time}Z in which {tag} is a user-specified "tag" intended to differentiate collections of data and {start_time} and {end_time} are the coordinated universal time (UTC) times corresponding to the first and last samples in the data block, respectively. The {start_time} and {end_time} fields are formatted similar to %Y%m%dT%H:%M:% S. %n (for string format specifications, see Table 1). Each data set also is assigned an attribute with key sampling_rate and value equal to the fixed temporal sampling rate of the time series as a floating point number in units of samples per second (s^{-1}) .

The /timeseries group requires a nested data table under the key __TS_INDEX that maintains an index of all other data sets in the /timeseries group. Specifications for storing tabular data are

White et al.



Figure 2. Schematic overview of example HDF5eis file. (a) The basic hierarchical layout of file contents. (b) A more detailed pictorial representation of file contents. Gray and blue containers represent the groups and data sets, respectively. Green rectangles represent the attributes.

discussed in the next subsection. For each data set in the /timeseries group, there must be a corresponding row in the __TS_INDEX table with tag, start_time, end_time, sampling_rate, and npts columns. The tag, start_time, end_time, and sampling_rate columns record the variables by the same name in the preceding paragraph. The npts column must be the integer number of time samples comprising the data set. This __TS_INDEX table can be quickly parsed to find the data sets needed to fulfill a user query. No other tables are permitted in the /timeseries group.

The /products and /metadata groups

The /products and /metadata groups are fundamentally different from the /timeseries group in that the former provide storage and I/O for tabular data comprising columnar data with potentially mixed data types and UTF-8 encoded byte streams, whereas the latter provides storage and I/O for structured multidimensional arrays of time series data with uniform type. The difference between the /products and /metadata groups is merely an organizational convention: /products is intended for data products that are derived from or otherwise related to the data in /timeseries (e.g., earthquake catalog data), whereas /metadata is intended for static metadata associated with /timeseries data (e.g., recording geometry, instrument responses, and data provenance).

Tabular data

Tabular data are organized in groups (one per table) nested within the /products and /metadata groups (with the exception of the /timeseries/__TS__INDEX table) and addressed by /{parent}/{key} in which {parent} is either /products or /metadata, as appropriate, and {key} is a user-specified "key" value. To differentiate tabular data from UTF-8 encoded byte streams, groups corresponding to tabular data are each assigned an attribute with TYPE and value TABLE. Data sets corresponding to UTF-8 kev encoded byte streams, on the other hand, are each assigned a similar attribute with the value UTF-8. Each column within a table is stored in an independent data set addressed by /{parent}/{key}/{column}, in which {parent} and {key} are as previously described and {column} is the column name. Each column may have a different data type, but data within a single column must be of homogeneous type, and all columns within a single table must have the same length. Valid data types for columnar data are numeric types (integers and floats), character strings, and timestamp or "datetime" values.

Because the HDF5 backend does not support a native datetime data type, special care must be taken when storing tabular data with

columns of datetime values. HDF5eis stores datetime data as 64 bit integers representing the number of nanoseconds since 1 January 1970 00:00:00 (UTC). Every data set nested within a group for which the __TYPE Attribute is equal to TABLE requires an __IS _UTC_DATETIME64 attribute with a boolean value True or False. This flag indicates how to properly interpret (and potentially convert) the data at read time.

Columnar data comprising character strings must similarly be handled carefully. Just as every data set nested within a group for which the ___TYPE attribute is equal to TABLE requires an __IS_UTC_DATETIME64 attribute, so too do they require a similar __IS_UTF-8 attribute. Character strings are stored as raw UTF-8 encoded bytes with a sufficient number of bytes per string to represent the longest string in the column. Setting the __IS_UTF-8 attribute to True indicates that these bytes should be appropriately decoded at read time.

UTF-8 encoded data

UTF-8 encoded data can be stored in a columnar format when they constitute part of a data table (e.g., alphanumeric station IDs in a network geometry metadata table) or they can be stored as individual byte streams of arbitrary length. Each byte stream is stored in a separate data set, which is addressed by /{parent}/{key} in which /{parent} and {key} are as described for tabular data. Also as described for tabular data, each such data set must have a TYPE attribute with the value UTF-8. The rationale for permitting arbitrary UTF-8 encoded data is to enable existing formats to be leveraged. For example, the StationXML, QuakeML (Schorlemmer et al., 2011), and SEIS-PROV (Krischer et al., 2015) standards can be used to keep records of station metadata, seismic events, and data provenance, respectively, as is done in the ASDF. Alternatively, any other text-based format, even a simple text README, can be stored as a UTF-8 encoded byte stream.

Groups corresponding to tabular data and data sets corresponding to UTF-8 encoded byte streams each possess a _____FORMAT attribute that records the format of the stored data. As examples, setting

_____FORMAT to CSS3.1-origin could be used to indicate that a table records seismic event origin data using the Center for Seismological Studies CSS3.1 database schema, or STATIONXML might be used to indicate that a UTF-8 encoded byte stream can be parsed using the STATIONXML protocol.

API IMPLEMENTATION

We implement a publicly available Python API to build and process HDF5eis files. Although HDF5eis's schema definition depends only

Table 1. String format specifications for naming data sets.

Formatter	Meaning	Range	Example
%Y	Four digit year	0–9999	2022
%m	Zero-padded, two digit month	1–12	03
%d	Zero-padded, two digit day of the month	1–31	27
%H	Zero-padded, two digit hour of the day	0–23	14
%M	Zero-padded, two digit minute of the hour	0–59	19
%S	Zero-padded, two digit second of the minute	0–59	01
%n	Zero-padded, nine digit nanosecond of the second	0-999999999	030900120

on the HDF5, the Python API that we implement introduces a few additional dependencies. We use the h5py API to interact directly with the underlying HDF5 files. In addition to h5py, we use the pandas package (McKinney, 2010) to manipulate tabular data in memory. In particular, we manipulate the __TS_INDEX table using pandas to rapidly fulfill time series queries — specifying the tag and start and end time of the desired samples is sufficient to determine precisely which data set(s) contain them and at which offsets they reside within their respective data set(s). Our API also serves tabular data from HDF5eis files as pandas.DataFrame objects. Finally, we use the numpy library (Harris et al., 2020) to manipulate time series data in memory and perform various other basic operations. These three dependencies (h5py, pandas, and numpy) are mature, stable packages that are unlikely to introduce breaking changes in the near future. Our API is most easily learned through the tutorials that are included in the distribution.

Because HDF5eis is designed to maximize flexibility, the mapping between existing formats and HDF5eis is nonunique. Multiple valid HDF5eis structures could be implemented to represent data in an existing format, and the optimal file structure will depend on the intended use case. The nonuniqueness of this mapping from conventional data formats to HDF5eis inhibits us from implementing a simple program for converting data. HDF5eis users, at least those creating HDF5eis files, need to have some understanding of the internal HDF5eis structures to convert data from existing formats. Although it is a fairly straightforward exercise, users must write a script to convert their data to HDF5eis. To facilitate this process and demonstrate additional nuances of the API not covered in this paper, we include a few tutorials in the distribution.

EXAMPLE USE CASE

To illustrate the advantages of implementing a multidimensional array data structure and basic features of our API, we now consider a hypothetical experiment monitoring microseismic activity during a hydraulic fracturing treatment. In this section, we describe the experimental design and basic structure for handling the data using HDF5eis. In the "Comparison against alternative formats" section, we simulate a small subset of the data that would be recorded during such an experiment and compare the performance of various data formats for handling the data.

The data acquisition system for our hypothetical experiment comprises four separate components: (1) a downhole DAS fiber, (2) a downhole distributed temperature sensing (DTS) fiber, (3) a 1D array of downhole geophones, and (4) a 2D array of surface geophones.

Table 2. Summary of data acquisition parameters in a hypothetical 30 day experiment monitoring microseismicity during hydraulic fracturing treatment.

Component	Number of channels	Sampling rate (s^{-1})	Data size
DAS	1024	4000	38.62 TB
DTS	1024	60^{-1}	168.8 MB
Downhole geophones	$24 \times 3 = 72$	500	347.6 GB
Surface geophones	$16 \times 16 \times 3 = 768$	500	3.621 TB
All	2888	NA	42.58 TB

Data are acquired from all four of these components continuously over a 30 day period. The DAS and DTS fibers record 1024 channels of data. The DAS data are sampled at a rate of 4000 s⁻¹, whereas DTS data are sampled at a rate of 1 min⁻¹. The downhole geophone array comprises 24 3C sensors sampled at a rate of 500 s⁻¹. The 2D array of surface geophones comprises a 16 × 16 array of 3C sensors also sampled at a rate of 500 s⁻¹. Data recorded in this configuration and stored with 32 bit floats will require 43 TB of storage (Table 2). Organizing storage for such a data set is a significant technical challenge, but this challenge can be handled efficiently using HDF5eis as follows.

DAS data are amenable to being stored on disk using 2D arrays with shape (nchannels and npts) in which nchannels is the number of channels being recorded and npts is the number of temporal samples. Allocating one file for every 5 min of uncompressed DAS data requires 8640 files consuming approximately 4.6 GB each, which is manageable on any modern computer with enough storage. It also is important to consider the "chunk layout" and compression algorithm when creating HDF5eis files, as they will significantly impact storage and I/O performance, but both of these aspects are optional and a discussion of them is beyond the scope of this thought experiment. DAS data can be associated with the tag DAS.

DTS data also are amenable to being stored on disk using 2D arrays with shape (nchannels and npts). Because DTS data are sampled infrequently, they amount to less than 1 GB of data and can easily be stored in a single file. DTS data can be associated with the tag DTS.

Downhole geophone data are amenable to being stored on disk using 3D arrays with shape (nsensors, ncomponents, and npts) in which nsensors = 24 is the number of recording sensors and ncomponents = 3 is the number of components per sensor. Allocating one file for every 6 h of downhole geophone data requires 120 files consuming approximately 2.9 GB of storage each. Downhole geophone data can be associated with tag geophones/downhole.

Surface geophone data are amenable to being stored on disk using 4D arrays with shape (nrows, ncolumns, ncomponents, and npts) in which nrows = ncolumns = 16 are the number of sensor rows and columns, respectively. Allocating one file for every 30 min of surface geophone data requires 1440 files consuming approximately 2.6 GB of storage each. Surface geophone data can be associated with the tag geophones/surface.

All 10,201 files created in the preceding configuration can be linked to and subsequently accessed through a single master file using our Python API. Array geometry also can be stored in tabular format under the /metadata group. The user no longer needs to maintain a record of the location of each individual file, even if they are distributed across multiple disks on a network file system. Furthermore, the user can request data spanning multiple files without any knowledge of which samples come from which files; the necessary bookkeeping is done by the API. The structure of the resultant master file may look as shown in Figure 3.

Not only do such data structures simplify bookkeeping for such large data sets but also they enable efficient I/O for many common access patterns via hyperslab selection (also known as array slicing). Data subsets specified by start and end indices and stride length along each storage dimension can be extracted in a single call to the HDF5 library. For instance, each of the following subsets of data can be extracted in a single API call that leverages the HDF5 hyperslab selection functionality for efficient I/O (Figure 4):

- 1) DAS data from all channels.
- 2) DTS data from channels 256 to 511.
- 3) Vertical-component data from downhole geophones 8 to 11.
- 3C data from every second geophone in row 12 of the surface array.

These storage requirements are based on the assumption that data are uncompressed; however, one may wish to compress the data. HDF5eis inherits and exposes various compression algorithms, which can be configured to suit one's particular needs. For example, the GZIP compression algorithm can be used with compression level 9 (maximum compression) for long-term archival or data exchange purposes. Entire files can be retrieved and decompressed for processing as needed. Alternatively, one may compress the data with a more moderate compression level and decompress it in an on-the-fly manner. In this scenario, small chunks of compressed data can be transmitted from a network file system to a processing node where it can be decompressed for processing. This only requires decompressing the requested chunks of data (not the entire file) and can be optimized to balance demands on storage, network, and compute resources.

COMPARISON AGAINST ALTERNATIVE FORMATS

In this section, we describe results from two experiments designed to assess the relative performance of HDF5eis against SEG-Y, miniSEED, and the ASDF. SEG-Y is chosen as a representative of exploration seismic data formats. MiniSEED is chosen as a representative of earthquake seismic data formats. The ASDF is chosen as a comparable format to HDF5eis because it also leverages the HDF5. The code for these experiments is included as a Jupyter Notebook in the supplemental S1 for readers who wish to validate these results. The segyio (Equinor ASA, 2022), obspy (Beyreuther et al., 2010), and pyasdf (Krischer et al., 2016) Python packages are used to read and write SEG-Y, miniSEED, and the ASDF data, respectively.

The first experiment is designed to simulate random reads from the DAS data set described in the previous section. We generate random sample values to simulate real data recorded by 1024 channels with a 4000 s⁻¹ sampling rate over a 30 min period. Data are stored in an uncompressed 32 bit floating point representation for all formats. HDF5eis data are stored in a single file per 5 min period as described in the previous section. SEG-Y data also are stored in a single file per 5 min period. MiniSEED data are stored in one file per channel. All the ASDF data are stored in a single file. We record the average amount of time required to read random segments of data.

At each realization, we read data from a random subset of sequential channels between random start and end times. The number and position of the channels, the start time, and the duration of the data read are all drawn from uniform random distributions. The duration of data read at each iteration is between 0.5 and 2 s. The same random subset is read from each data format and the corresponding read times are recorded for 1000 iterations. Figure 5a shows the read times in multiples of the corresponding HDF5eis read time (values greater than one indicate slower reads than HDF5eis, whereas values less than one indicate faster reads). SEG-Y is substantially more efficient than miniSEED and the ASDF, which is unsurprising given that SEG-Y is intended for such large-*N* array data. The median relative read time for SEG-Y, however, is 6.5 times that of HDF5eis. The median relative read times for miniSEED and the ASDF are 22.9 and 40.6, respectively. HDF5eis is an order of magnitude faster than miniSEED and the ASDF, and it achieves this performance with the fewest lines of client code. Although high-level APIs could be developed to read data from other formats using similar amounts of client code, pithy code is characteristic of HDF5eis because its multidimensional array storage structure naturally leverages HDF5's hyperslab selection functionality. Notably, the ASDF is the slowest of the formats. This is because of the first and second design features outlined in the "Motivation" section.

The second experiment is designed to simulate random reads from the data set recorded by the 2D array of geophones described in the "Example use case" section. We again use random sample values to simulate recorded data, this time for a 2D array of 3C geophones with 16 rows, 16 columns, and a 500 s⁻¹ sampling rate over a 1 day period. MiniSEED and the ASDF data are stored in one file per channel and a single file, respectively, as in the first experiment. SEG-Y and HDF5eis data are stored in one file per 30 min period.

At each realization, a random segment of data between 2 and 6 s long is read from a random subset of rows, columns, and channels starting at a random start time. As in the first experiment, all random variables are drawn from uniform random distributions. Here, 1000 iterations are performed and the results are summarized in Figure 5b. All median relative read times are again greater than one; however, in this experiment, the median relative read time for SEG-Y is only 1.9. The median relative read times for miniSEED and the ASDF are 105.1



Figure 3. The hierarchical structure of the master file for a hypothetical multisystem survey (described in the "Example Use Case" section) monitoring microseismicity during a hydraulic fracturing treatment. The ellipses (\ldots) nested under the /timeseries group represent the time series data.

and 63.0, respectively. HDF5eis is nearly twice as fast as SEG-Y and two full orders of magnitude faster than miniSEED in this context.

DISCUSSION

In this paper, we present a data schema, which we name HDF5eis, for archiving, exchanging, and processing big multidimensional time series data from environmental sensors in HPC applications using the HDF5. We contribute it as (1) a functional tool and (2) an experimen-

```
import hdf5eis
with hdf5eis.File("demo.hdf5", mode="r") as f5:
    # (a) Extract 1s of DAS data for all channels.
   start_time, end_time = "2022-01-01T00:00:00Z", "2022-01-01T00:00:01Z"
    das = f5.timeseries["DAS", :, start_time, end_time]
    # (b) Extract 1h of DTS data from channels 256 to 511.
   start_time, end_time = "2022-01-01T00:00:00Z", "2022-01-01T01:00:00Z"
   dts = f5.timeseries["DTS", 256: 512, start_time: end_time]
    # (c) Extract 10 s of vertical-component data from downhole
          geophones 8 to 11.
   start_time, end_time = "2022-01-01T00:00:00Z", "2022-01-01T00:00:10Z"
       = f5.timeseries["geophones/downhole", 8: 12, 0, start_time: end_time]
    dwn
    # (d) Extract 4 s of 3C data from every second
          geophone in row 12 of the surface array.
    #
   start_time, end_time = "2022-01-01T00:00:00Z", "2022-01-01T00:00:04Z"
        = f5.timeseries["geophones/surface", 12, ::2, :, start_time: end_time]
    # (e) Extract geometry metadata for DAS data
    das_geom = f5.metadata["geometry/DAS"]
```

Figure 4. Illustrative API syntax for efficiently reading various subsets of data from a hypothetical multisystem survey (described in the "Example use case" section): (a) 1 s of DAS data for all channels, (b) 1 h of DTS data for channels 256–511, (c) 10 s of vertical-component data from downhole geophones 8 to 11, (d) 4 s of 3C data from every other geophone in row 12 of the surface array, and (e) geometry metadata for DAS fiber.



Figure 5. Comparison of random read access times for representative seismic data formats. (a) Results for simulated DAS data. (b) Results for simulated 2D geophone array. Experimental designs are described in the "Example Use Case" section. Read times are reported in multiples of corresponding HDF5eis read time. Horizontal orange lines represent the median relative read time. Boxes span the first and third quartiles. Whisker lengths are 1.5 times the interquartile range. Outliers are shown as translucent black circles. The horizontal dashed black line represents the HDF5eis read time, which is unity by definition.

tal draft proposal to provide a basis on which to discuss the merits and demerits of existing technology and how they might be leveraged in future data formats for the earth sciences. Although our focus is seismic data, we recognize that time series data from many disciplines within the earth sciences share similarities, and we have aimed to accommodate a wide variety of data types by maintaining schema flexibility. To promote interdisciplinary interoperability and collaboration, standards for future data formats in the earth sciences should be the product of intergenerational, interdisciplinary committees. In

contribution to such potential future efforts, we now discuss the merits and demerits of HDF5eis.

First, HDF5eis inherits impressive I/O performance from the highly optimized HDF5 library (Figure 4). Because extensive effort has been invested by the HDF Group in implementing and optimizing advanced storage layout schemas, memory caching algorithms, and data indexing protocols, it is likely that only a highly skilled team of data scientists will be able to improve on these. Whereas conventional seismic data formats rely on their own schemas and protocols, future formats will benefit greatly from leveraging advances made by data specialists.

Second, HDF5eis is flexible. In fact, its I/O performance owes considerably to its flexibility. Supporting arrays of arbitrary dimensionality enables users to conform storage structures to the characteristics of their specific data set. Such multidimensional array structures also enable the efficient retrieval of coordinate data channels via hyperslab selection. Furthermore, because HDF5eis implements storage for columnar data and UTF-8 encoded byte streams, many existing data format standards can be internally maintained. For example, QuakeML (Schorlemmer et al., 2011) can be used to store earthquake catalog data, and because

HDF5eis is independent of QuakeML, it is future-proof against changes to the QuakeML schema. The same can be said of any analogous format. Alternatively, one could store earthquake catalog data in tabular format as part of a relational database. This flexibility and interoperability with existing formats will reduce the overhead associated with adapting software to operate on HDF5eis files.

Third, HDF5eis freely inherits multiple lossless, in-flight compression algorithms, and functionality to define custom filters. These enable the user to tailor the level of compression to their specific needs and balance demands placed on the network, storage, and processing resources. Users do not need to decompress an entire file when they only need a small subset of its contents; however, users may choose to heavily compress data files (e.g., for transfer or long-term, cold storage) and decompress files wholesale for processing.

Fourth, HDF5eis files are self-describing and platform-independent which makes them universally portable. A user in one computational environment will have no problem reading data written in an entirely different environment. Moreover, HDF5eis files can be manipulated in a wide variety of languages because HDF5 APIs exist for many of them including C/C++, Fortran, Java, Python, Julia, and MATLAB. Although comparable tools to the HDF5 exist, such as zarr and xarray, HDF5 is significantly more mature, has a wide user base, and has APIs for the greatest number of languages.

Finally, the use of external links in HDF5eis minimizes the bookkeeping burden borne by the user. Users of our Python API can create zero-copy virtual data sets and seamlessly read data across file boundaries through a single point of access without worrying about in which file samples reside. Users can simultaneously access diverse ancillary data (e.g., metadata) through the same single point of access using a unified programming interface. This allows users to rise above the mundanities of data management and focus instead on scientific research.

HDF5eis is not, however, without limitations.

First, HDF5eis is not cloud native, meaning that it is not optimized for interacting with data on cloud resources. It is our opinion that future data formats in the earth sciences should work at least as well on the cloud as they do on a local workstation or computing cluster. The highly scalable data service (HSDS) from the HDF Group is one technology that could be leveraged in a cloud-native version of HDF5eis. HSDS is designed to maximize interoperability with the existing HDF5 data and libraries, which means that migrating to HSDS from the HDF5 would engender relatively little overhead. However, this also means that HSDS design decisions are influenced by noncloud-native technology, which may ultimately detract from cloud performance. As cloud-computing paradigms mature, so do the tools designed specifically for the cloud. Emerging cloud technologies, such as TileDB (Papadopoulos et al., 2016), offer great promise for providing dyed-in-the-wool cloud-native backend data formats that can be leveraged in the earth sciences.

Second, HDF5eis currently only supports the UTC time standard. To be more broadly useful, formats for time series data from environmental sensors should implement multiple time standards such as mean solar time, atomic time, and GPS time. In addition to these, elapsed time, representing the time elapsed since some arbitrary datum, would be useful for storing data without an absolute time frame, such as Green functions and (cross)correlograms, or data for which the elapsed time is more important than the absolute time, such as in typical exploration seismology. Adding support for these time standards is fairly straightforward in principle and may be the subject of future development, depending on user interest.

Third, HDF5eis only supports data sampled at regular time intervals. Because many sensor types sample at irregular intervals, supporting such data is important. Again, adding support for irregularly sampled time series is fairly straightforward and may be the subject of future development.

Finally, HDF5eis is not designed for real-time data acquisition and may not be suitable in certain contexts. For instance, data packets transmitted by regional seismic networks cannot be assumed to arrive in proper time order. Recording such data using HDF5eis, particularly when gaps in data transmission occur, is likely to be cumbersome and may produce inefficient data structures.

CONCLUSION

The HDF5 is a suitable substrate on which to build efficient and flexible data formats for handling big multidimensional time series data from environmental sensors in HPC applications. Implementing a simple indexing protocol, as is done by HDF5eis, can yield superior I/O performance in comparison with conventional seismic data formats. The time required to extract physical insights from large data sets can thus be effectively reduced when processing is I/O bound. HDF5eis is a functional tool for simplifying complex access patterns to diverse data in distributed computing environments. As data volumes swell and cloud technology matures, data-driven research in the earth sciences will increasingly demand data standards beyond the scope of those that exist today. HDF5eis sheds needed light on the path toward establishing standards for the next generation.

ACKNOWLEDGMENTS

The study was supported by the Southern California Earthquake Center Award #22145 and the Japan Organization for Metals and Energy Security.

DATA AND MATERIALS AVAILABILITY

No data have been required for this paper.

REFERENCES

- Ahern, T. K., R. Casey, D. Barnes, R. Benson, and T. Knight, 2014, Seed reference manual v2.4 : Incorporated Research Institutions for Seismology (IRIS)
- Allen, R., G. Crews, W. Guyton, C. A. McLemore, B. Peterson, C. S. Rapp, L. Walker, L. R. Whigham, D. A. White, and G. Wood, 1994, Digital field tape format standards SEG-D, revision 1: Geophysics, 59, 668–684, doi: 10.1190/1.1443626
- Arrowsmith, S. J., D. T. Trugman, J. MacCarthy, K. J. Bergen, D. Lumley, and M. B. Magnani, 2022, Big data seismology: Reviews of Geophysics, **60**, 1–55, doi: 10.1029/2021RG000769.
- Barry, K. M., D. A. Cavers, and C. W. Kneale, 1975, Recommended stan-dards for digital tape formats: Geophysics, 40, 344–352, doi: 10.1190/1 144053
- Ben-Zion, Y., F. L. Vernon, Y. Ozakin, D. Zigone, Z. E. Ross, H. Meng, M. White, J. Reyes, D. Hollis, and M. Barklage, 2015, Basic data features and results from a spatially dense seismic array on the San Jacinto fault zone: Geophysical Journal International, 202, 370-380, doi: 10.1093/gji/ ggv14
- Beyreuther, M., R. Barsch, L. Krischer, T. Megies, Y. Behr, and J. Wassermann, 2010, ObsPy: A Python toolbox for seismology: Seismological Research Letters, **81**, 530–533, doi: 10.1785/gssrl.81.3.530.
- Equinor ASA, 2022, segyio [software], https://github.com/equinor/segyio, accessed 21 February 2023.
- Gibbons, S. J., and F. Řingdal, 2006, The detection of low magnitude seismic events using array-based waveform correlation: Geophysical Journal
- International, 165, 149–166, doi: 10.1111/j.1365-246X.2006.02865.x.
 Goldstein, P., D. Dodge, M. Firpo, L. Minner, W. Lee, H. Kanamori, P. Jennings, and C. Kisslinger, 2003, SAC2000: Signal processing and analysis tools for seismologists and engineers, The IASPEI International handbook of earthquake and engineering seismology: Incorporated Research Institutions for Seismology (IRIS) 81, 1613–1620. Goldstein, P., and A. Snoke, 2005, SAC availability for the IRIS community:
- Incorporated Institutions for Seismology Data Management Center Elec-
- tronic Newsletter, 7. Habermann, T., 2015, HDF earth science program: The HDF Group. Harris, C. R., K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Rio, M. Wiebe, P. Peterson, P. Gerard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, 2020, Array programming with NumPy: Nature, 585, 357-362, doi: 10.1038/s41586-020-2649
- Hess, D., S. Azevedo, N. Falco, and B. C. Beaudoin, 2017, PH5: HDF5 based format for integrating and archiving seismic data: AGU Fall Meet-
- Krischer, L., J. Smith, W. Lei, M. Lefebvre, Y. Ruan, E. S. de Andrade, N. Pod-horszki, E. Bozdag, and J. Tromp, 2016, An adaptable seismic data format: Geophysical Journal International, **207**, 1003–1011, doi: 10.1093/ 11.1011/11.10111. gji/ggw319
- Krischer, L., J. A. Smith, and J. Tromp, 2015, SEIS-PROV: Practical prov-enance for seismological data: AGU Fall Meeting Abstracts, S53A-2768.
- Li, Z., and M. van der Baan, 2021, Reverse-time imaging of DAS microseismic data: First International Meeting for Applied Geoscience Energy,
- Expanded Abstracts, 392–396, doi: 10.1190/segam2021-3593211.1. Lin, F.-C., D. Li, R. W. Clayton, and D. Hollis, 2013, High-resolution 3D shallow crustal structure in Long Beach, California: Application of ambient noise tomography on a dense seismic array: Geophysics, 78, no. 4, Q45-Q56, doi: 10.1190/geo2012-0453.1.

- Lindsey, N. J., T. C. Dawe, and J. B. Ajo-Franklin, 2019, Illuminating seafloor faults and ocean dynamics with dark fiber distributed acoustic sensing: Science, 366, 1103–1107, doi: 10.1126/science.aay5881.
- McKinney, W., 2010, Data structures for statistical computing in Python: Proceedings of the 9th Python in Science Conference, 56–61.
 Mousavi, S. M., W. L. Ellsworth, W. Zhu, L. Y. Chuang, and G. C. Beroza, 2020, Earthquake transformer An attentive deep-learning model for simultaneous earthquake detection and phase picking: Nature Communi-cations, **11**, 3952, doi: 10.1038/s41467-020-17591-w. Papadopoulos, S., K. Datta, S. Madden, and T. Mattson, 2016, The TileDB
- array data storage manager: Proceedings of the VLDB Endowment, **10**, 349–360, doi: 10.14778/3025111.3025117.
- Ross, Z. E., M.-A. Meier, E. Hauksson, and T. H. Heaton, 2018, Generalized seismic phase detection with deep learning: Bulletin of the Seismological Society of America, **108**, 2894–2901, doi: 10.1785/0120180080.
- Ross, Z. E., D. T. Trugman, E. Hauksson, and P. M. Shearer, 2019, Searching for hidden earthquakes in Southern California: Science, 364, 767-771, doi: 10.1126/science.aaw6888.
- Schorlemmer, D., F. Euchner, P. Kastli, and S. Joachim, 2011, QuakeML: Status of the XML-based seismological data exchange format: Annals of Geophysics, **54**, 59–65, doi: 10.4401/ag-4874.
- Shelly, D. R., 2020, A high-resolution seismic catalog for the initial 2019 Ridgecrest earthquake sequence: Foreshocks, aftershocks, and faulting complexity: Seismological Research Letters, **91**, 1971–1978, doi: 10
- Shelly, D. R., W. L. Ellsworth, and D. P. Hill, 2016, Fluid-faulting evolution in high definition: Connecting fault structure and frequency-magnitude variations during the 2014 Long Valley Caldera, California, earthquake swarm: Journal of Geophysical Research: Solid Earth, **121**, 1776–1795, doi: 10.1002/2015JB012719.
- The HDF Group, 1997–2022, Hierarchical Data Format, version 5, https:// www.hdfgroup.org/HDF5/, accessed 21 February 2023.

Unidata, 2016, Network Common Data Form (netCDF): Unidata.

- Wapenaar, K., D. Draganov, R. Snieder, X. Campman, and A. Verdel, 2010a, Tutorial on seismic interferometry: Part 1 Basic principles and applications: Geophysics, **75**, no. 5, 75A195–75A209, doi: 10 1100/12457445 1190/1 34574
- Wapenaar, K., E. Slob, R. Snieder, and A. Curtis, 2010b, Tutorial on seismic interferometry: Part 2 Underlying theory and new advances: Geophysics, **75**, no. 5, 75A211–75A227, doi: 10.1190/1.3463440. Zhu, W., and G. C. Beroza, 2018, PhaseNet: A deep-neural-network-based
- seismic arrival time picking method: Geophysical Journal International, **216**, 261–273, doi: 10.1093/gji/ggy423.



Hilary Chang received a B.S. (2009) in finance from National Taiwan University and a B.S. (2019) in earth sciences from the Memorial University of Newfoundland. She is a Ph.D. candidate at the Massachusetts Institute of Technology. She is a member of AGU and SEG. Her research interests include near-surface site effects, ambient noise tomography, temporal

monitoring, and DAS.

Biographies and photographs of the other authors are not available.