

MDIO: Open-source format for multidimensional energy data

Altay Sansal¹, Sribharath Kainkaryam¹, Ben Lasscock¹, and Alejandro Valenciano¹

<https://doi.org/10.1190/tle42070457.1>

Abstract

MDIO is a fully open-source data storage format that enables computational workflows for various high-dimensional energy data sets including seismic data and wind models. Designed to be efficient and flexible, MDIO provides interoperable software infrastructure with existing energy data standards. It leverages an open-source format called Zarr to enable data usage in the cloud and on-premises file systems. An overview of the data model and schema for MDIO is provided, and an open-source Python library developed to work with MDIO data is demonstrated. We explain how MDIO supports different computational workflows and discuss applications for data management, seismic imaging, machine learning, wind resource assessment, and real-time seismic visualization. Overall, MDIO gives researchers, practitioners, and developers in the energy sector a standardized and open approach to managing and sharing multidimensional energy data.

Introduction

The energy industry relies heavily on data to make informed exploration, production, and asset management decisions. Different data formats facilitate scientific workflows depending on the business or application needs. SEG-Y (Society of Exploration Geophysicists, 2002) is a widely used data format for storing and sharing seismic data in the exploration geophysics industry. Academia introduced SEP (Claerbout, 1991), SU, and Madagascar (Fomel, 2013) formats and associated utility software to facilitate research in exploration seismic processing applications. Other formats for commercial seismic data workloads were developed and later open sourced, such as the Data Dictionary System (DDS), OpenVDS (Barré et al., 2022), and OpenZGY (OSDU, 2023). NetCDF4 (Unidata, 2021) and HDF5 (The HDF Group, 1997–2023) are popular formats in atmospheric and oceanic sciences, geophysics, and climate modeling. While we have endeavored to include popular binary storage formats, we acknowledge potential omissions and invite readers to suggest any overlooked alternatives for future updates.

This article explains the data model and schema for MDIO, a modern fully open-source energy data storage format. We review the input/output (I/O) access patterns for five applications: seismic data management, seismic data processing, machine learning, wind resource assessment, and seismic data visualization. We illustrate how the chunk size and data compression required by MDIO are essential for performance. We also review the setting of these parameters, with applications to both cloud object stores and local file systems. We highlight the performance considerations and potential cost savings of using data compression for storing data, with examples of defaults for seismic data applications. Based

on tools from the PyData ecosystem, MDIO can integrate with existing libraries to process and analyze scientific data without reimplementing many algorithms.

The MDIO format

MDIO is a format that self-describes seismic and wind data through Zarr (Miles et al., 2023) arrays and JSON metadata. Self-describing implies that the data and metadata within an MDIO data set are accessible directly from the file, eliminating the need for a separate database backend. It facilitates easier data sharing, collaboration, and analysis while providing useful defaults for lossless and lossy data compression. It also pads irregular data to form a regularized hypercube without incurring additional storage overhead while maintaining a mapping to the measured data. MDIO currently utilizes the Segyio (Equinor, 2023) library to parse SEG-Y files during ingestion and write text and binary headers during export. Our custom SEG-Y encoder features tree-based parallelism for efficiently writing the N-D header and trace arrays in a distributed system.

The library features a standard application programming interface (API) to read, write, and perform tensor operations on the data, abstracting the underlying file storage location and types such as network file storage or cloud object stores. MDIO includes converters for widely used file formats such as SEG-Y and NetCDF4.

Seismic and wind projects exhibit similar data patterns including multiple data sets sharing a common grid, coordinate reference system information, and time/depth series of spatial data. MDIO flexibly represents these features in a container, allowing for either sequential ordering or granular chunking, depending on the workflow requirements.

MDIO specification

This section briefly overviews the seismic data specification stored as MDIO. We built the MDIO format as an extension of the Zarr protocol. The current specification (MDIO 0.2.10) is shown in Figure 1 and has the following key features:

- **Global attributes:** These attributes include information about the creation of the data set, details on the geometry, and global statistics such as minimum, maximum, root-mean-square, mean, and standard deviation values of samples.
- **Metadata group:** This group contains text, binary, trace headers, and a live trace mask. The metadata elements hold additional information about the data such as acquisition parameters, processing history, and QC flags. Horizons associated with the seismic grid can also be stored together here. Horizon

¹TGS, Houston, Texas, USA. E-mail: altay.sansal@tgs.com; sribharath.kainkaryam@tgs.com; ben.lasscock@tgs.com; alejandro.valenciano@tgs.com.

data would best be stored as individual maps (2D arrays) within the metadata group, given the current MDIO schema.

- **Data group:** This group contains multidimensional arrays that users can divide into one or more chunks and store as Zarr arrays. These arrays contain seismic data or any other multidimensional array representing other attributes. The user can choose the size of the chunks and compress the data in a lossless or lossy manner. Supported compression algorithms include Blosc, Zstd (Collet and Kucherawy, 2018), LZ4 (Collet, 2011), and ZFP (Lindstrom, 2014).

MDIO provides optimized default chunk dimensions for several seismic and wind data types, simplifying the process for users. Table 1 compares the optimal chunking scenarios for different seismic data, heuristically tailored to various applications such as archival, orthogonal access, and visualization. Adjusting these settings can optimize performance in specific workflows. When using lossy compression options, it is feasible to enlarge the chunk dimensions for transferring more data segments simultaneously. However, maintaining the same chunk sizes could lead to faster loading times when an application has bandwidth limitations. It is important to note that there is a tradeoff between chunk

sizes and the number of chunks. This may affect performance if not tuned properly. We selected the following example chunk sizes, drawing from benchmarks and heuristics to accommodate three primary workloads: archival I/O performance (including data ingestion and export), the flexibility to rechunk for specialized cases in the future, and satisfactory visualization performance in the common directions in which users explore data.

Configuring chunk sizes

Chunk dimensions in cloud object stores significantly influence the performance and cost of read and write operations. Key takeaways include

- Chunk dimensions should be aligned with access patterns to optimize performance.
- When using cloud object stores, aim for chunk sizes between 4 and 8 MB.
- Larger chunk sizes are preferred in cloud object stores due to the following. (1) Object stores have higher latency per request. More data should be moved in one request. (2) Fewer API calls are more cost-effective.

General guidelines for any chunked data storage applicable to MDIO include

- In traditional parallel file systems, chunk sizes between 1 and 2 MB are preferred.
- Chunk dimensions that are a power of two can benefit downstream workflows such as deep learning.
- Chunk sizes should be calibrated based on the compression ratio, and optimal sizes should be maintained.

In cases where ideal chunk sizes are unknown beforehand, employing isotropic or uniform chunking as an initial strategy is recommended. Isotropic chunking maintains equal chunk sizes in every direction, while uniform chunking divides the entire data domain into an equal number of chunks across all dimensions. These methods are a practical starting point for undefined access patterns, ensuring satisfactory performance for any reading direction. Nonetheless, to fully harness the advantages of chunked storage formats, it is recommended to optimize chunk sizes based on specific use cases and rechunk data sets when necessary.

Users can install MDIO, an open-source library under the Apache 2.0 license, through widely used channels such as Conda and Pip.

Applications

Seismic data management. When managing and organizing seismic data sets stored in the SEG-Y format, manipulating the data can be challenging, mainly when dealing with many monolithic and large data sets. Data management operations involve indexing trace data headers of petabyte-scale data sets that need to support various spatial search queries. Relational databases are commonly used

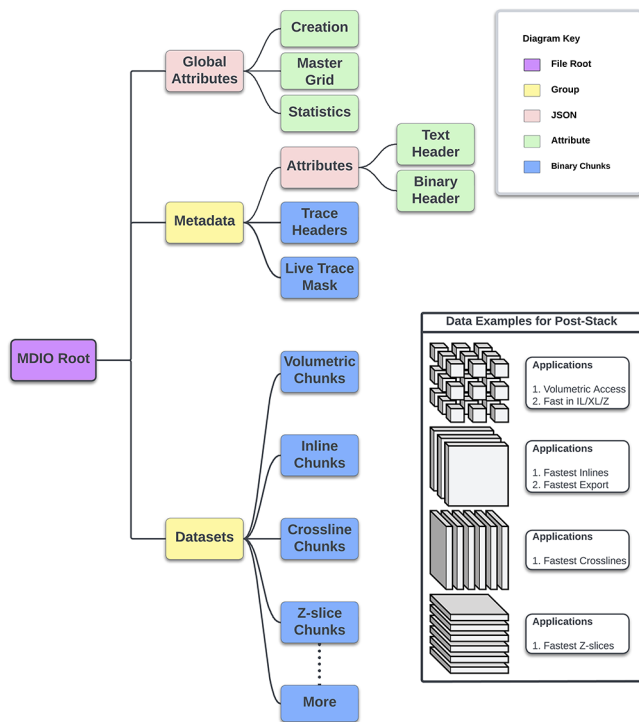


Figure 1. MDIO seismic data specification (v0.2.10). We show various access patterns for 3D seismic data sets.

Table 1. Some examples of multiworkflow optimized chunk sizes for MDIO. CDP = common-depth point.

Data type	Indices	Chunk dimensions	Chunk MB
3D, poststack	Inline, crossline, sample	128, 128, 128	8 MB
2D, migrated CDP	CDP, offset, sample	16, 64, 1024	4 MB
3D, prestack shot	Shot, cable, channel, sample	8, 2, 128, 1024	8 MB

to index trace headers, leading to increased costs and operational challenges. Indexing drives up costs due to database requirements and unnecessarily slow I/O-bound API requests. SEG-Y is designed for sequential access to trace and header data. In applications such as imaging and data visualization, the access patterns commonly devolve to making large numbers of requests to individual traces or header samples. This is not an optimal access pattern for cloud object stores. Demands to reduce costs while supporting scalability drive a growing need to access seismic data from anywhere, so first-class support of cloud-based storage makes sense.

A cloud-native storage format such as MDIO allows users to fully utilize cloud technology. Scalability, lowering storage costs, governance, and data quality were our main principles when designing MDIO. Figure 2 presents MDIO ingestion benchmarks for small, medium, and large SEG-Y files, conducted using an Amazon Web Services (AWS) c6gd.12xlarge virtual machine (VM) in the cloud. The VM has 24 physical cores, each supporting two threads, resulting in 48 virtual CPUs. Additionally, the VM features 96 GB of memory and a network bandwidth of up to 20 Gbps. Each run has been conducted five times, and the numbers shown are averaged. The spread of the benchmarks is shown at the tip of the histograms with the whisker plots. The experiment is set up as follows:

- Source SEG-Ys are located on the local solid-state drive (SSD).
- We ingested the data to two destinations: local SSD and cloud object store.

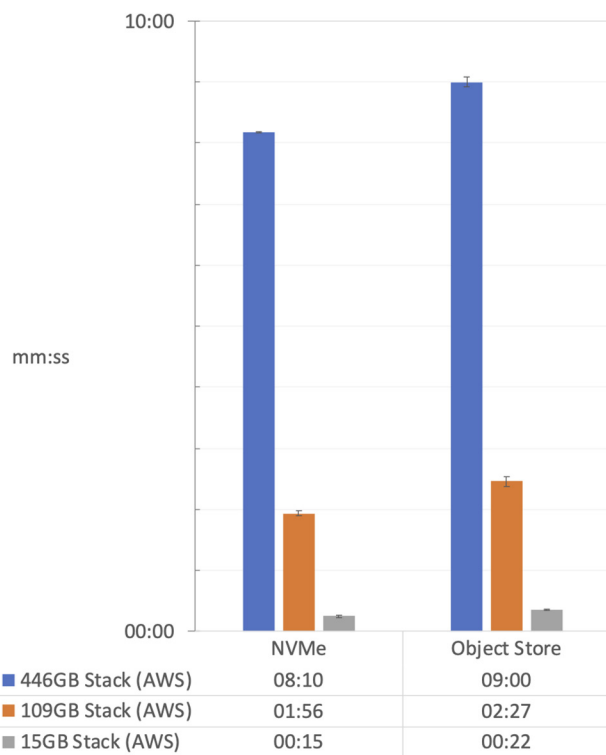


Figure 2. SEG-Y to MDIO ingestion times in minutes and seconds. The source SEG-Y is on local SSD. We benchmark ingesting to MDIO on local SSD and compare it to ingesting MDIO directly to the object store.

These steps are followed:

- 1) Ingest each file to MDIO on SSD and time it.
- 2) Ingest each file to MDIO on the object store and time it.
- 3) Repeat the first and second steps five times and average the timings.

We show the times under the bar graphs. We observe 8 to 9 minutes to ingest a 450 GB seismic stack.

By default, MDIO saves approximately 41% of our storage costs using lossless Zstd compression on our petabyte-scale pre- and poststack seismic data sets, a significant cost savings. Even after compression, thanks to chunking, MDIO provides easy access for QC because the whole data set does not have to be decompressed all at once.

A typical workflow in accessing seismic data is to subset large volumes using a geospatial polygon. Multiple users accessing a large MDIO file, consisting of many chunks, may have entitlements to different areas of the same data set. By using the masking capabilities of MDIO, parts of the files that are not entitled could be obfuscated during data retrieval or encoded back to SEG-Y, only preserving the relevant traces and headers. The same logic could be applied to masking the data in time/depth. Figure 3 demonstrates this concept with the polygon and the optimal chunk layout.

Seismic streaming is another application supported by MDIO. A streaming service allows users to access data on demand. In this use case, MDIO and Dask (Rocklin, 2015) can provide a distributed high-performance backend to serve requests for data on an object store. A lightweight client library can then serve the user's needs. While a prototype implementation of Async-Zarr (to be used in web applications) is publicly available, we chose the widely recognized and stable Dask library for handling distributed workloads. MDIO features a native Dask backend, presenting Zarr arrays as lazy Async-compatible Dask arrays.

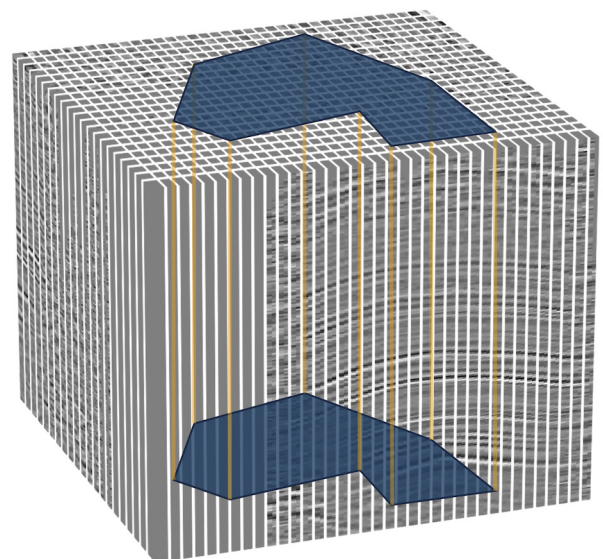


Figure 3. Concept of polygon limited seismic data retrieval with spatial chunks.

Seismic imaging and processing. In seismic imaging and processing, several factors determine the viability of the MDIO file format. There is a need to support both local high-performance computing and cloud computing. A file format must support high-performance read/write of seismic data, be interoperable with existing software, and be extensible with new technological developments.

Many seismic imaging and processing platforms adopt a hybrid model of on-premises computing with scalability to a private cloud. For this model to work effectively, the seismic data format must support both scenarios seamlessly, with applications potentially needing to access data from a local file system, remote object store, or a combination of the two. MDIO provides a unified set of APIs, allowing data access and manipulation from both sources. As discussed, MDIO also supports both lossy and lossless data compression, which can reduce the overall storage costs associated with data and improve the system's throughput. In algorithms such as POCS (Abma and Kabir, 2006), the data can be efficiently read in memory once, and the output can again be written to a chunk without locks, disk seeking, or many web requests. Figure 4 demonstrates a processing application that computes windowed 3D Fourier transforms.

In summary, chunk-aligned write operations can be safely executed in parallel across multiple processes and efficiently orchestrated using libraries such as Dask and MDIO. Nonetheless, it is crucial to recognize that despite the optimized access patterns, the I/O overhead may still be significant in specific workflows. Retaining the entire array or portions in memory could be more practical in such cases. For these situations, an in-memory variant of compressed MDIO could prove beneficial.

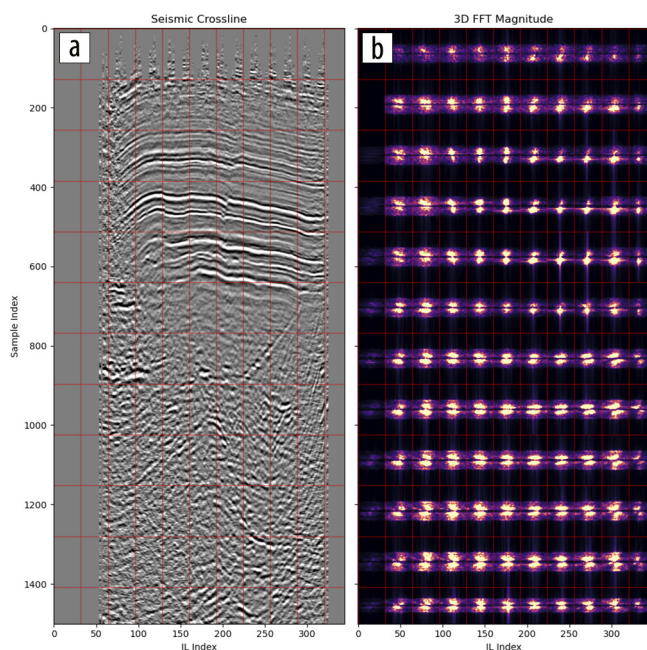


Figure 4. (a) Three-dimensional fast Fourier transforms are applied to 3D chunks, with an overlay of chunk boundaries on the seismic image. (b) The central wavenumber in the crossline direction (Kcrossline). The local coordinate axes for each chunk's resulting FFT magnitude are Kinline versus frequency. Each of the FFT processes reads either a single chunk or multiple chunks, forming a larger chunk and significantly reducing disk I/O.

Cloud object stores typically support large numbers of concurrent read and write operations on objects without performance degradation, but they can suffer a higher latency than other file storage solutions. The Zarr format naturally supports a high level of concurrency on read and write, optimizing its performance on object stores and potentially on a network file system. It is worth noting that performance can be limited by cloud service providers when thousands of machines access the same object. Arranging data in multiple chunks (preferably with randomized prefixes) further enhances parallelism (Google Cloud, 2023).

Figure 5 shows the throughput scaling of MDIO read and write of a 3D volume as a function of compute tasks for MDIO seismic data on Google Cloud Storage. The MDIO volume is read using Dask workers deployed on a Kubernetes cluster. Throughput is calculated by measuring the time to read the MDIO volume by scaling up the number of workers. Each worker contains a thread pool with two threads and 4 GB of memory.

Manipulation of seismic data for imaging, processing, and visualization requires the platform to efficiently support a set of tensor operations on the data. These may include reading slices (e.g., inline, crossline, and depth slices for visualization), header manipulation, and others. By decomposing the data into chunks, I/O using the MDIO can be significantly more performant than other popular alternatives such as SEG-Y and derivatives. For example, a header manipulation or indexing operation on an entire SEG-Y data set would require at least one read for every trace in the data set. Accessing data from an object store may require many requests (one per trace) or some transmission of trace data irrelevant to the task. By contrast, MDIO would require only one or relatively few reads to the chunked headers, which is more efficient in almost all cases. Another example may be a read to a slice or arbitrary line through a seismic survey. At the same time, this can be efficient for formats such as SEG-Y in the sort order of the data, but it becomes increasingly inefficient to slice data orthogonally to this. With MDIO, the shape of the chunks can be designed to provide consistent performance for different access patterns.

Most of the existing commercial seismic imaging and processing software is written in C, C++, and Fortran. Libraries such as TensorStore (Maitin-Shepard and Leavitt, 2022) provide C++ implementations of the Zarr protocol. MDIO is also designed to work seamlessly with popular scientific libraries and

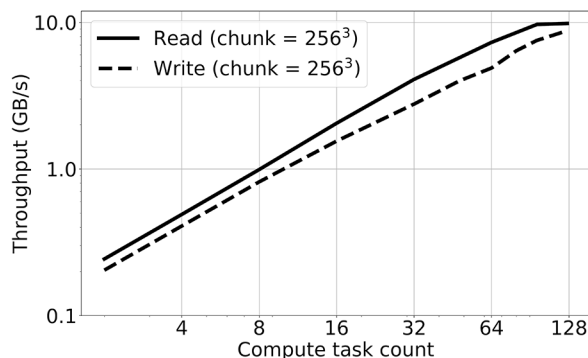


Figure 5. MDIO read/write performance as a function of the number of compute tasks.

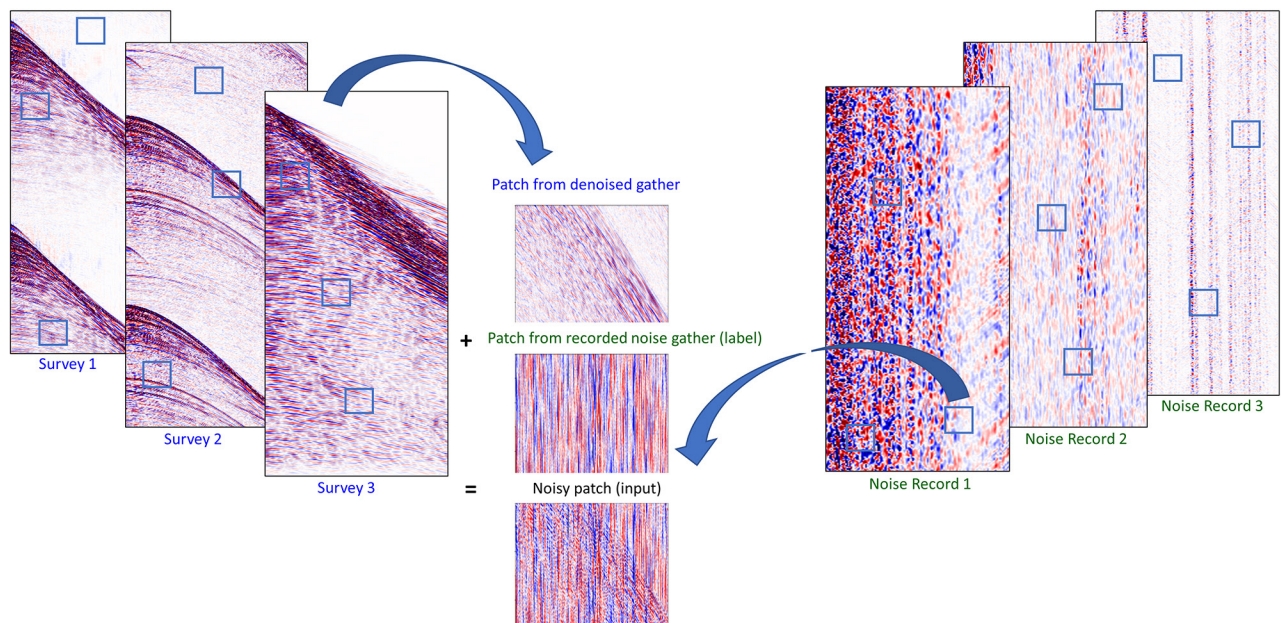


Figure 6. Generating seismic data tiles for training a deep learning model for swell noise removal using global data sets (from Valeciano et al., 2022).

frameworks such as NumPy and Dask. An imaging application employing Dask and Python with Zarr for 3D Marchenko imaging is discussed by Ravasi and Vasconcelos (2021). They found that the features of chunking, compression, multithreading, and multiprocessing read/write concurrency supported by Zarr were advantageous. The authors also demonstrate that the integration of Zarr with Dask and Kubernetes allowed them to create a fault-tolerant application that they could deploy to the cloud, realizing potential cost savings of using “spot pricing” for their computing.

Machine learning. The MDIO Python API is compatible with libraries such as NumPy, TensorFlow, and PyTorch, commonly used for the training and inference of machine and deep learning models. A data-loading pipeline is critical to training deep learning models with seismic data. Due to the seismic data set’s size, training on lines or patches is preferred to alleviate memory constraints. By using MDIO, patches can be extracted during training. This contrasts with an expensive data preprocessing stage, where patches are extracted from seismic data and stored as JPG/PNG images. Because of this, there is little additional overhead in creating a data pipeline for training compared to working natively with either NumPy data or collections of images. MDIO also carries a set of summary statistics that can be used to standardize inputs to a machine learning model. This is an important step in most cases.

An example application where MDIO was used to generate patches in the deep learning application is shown in Figure 6. Here, collections of patches are sampled from both a seismic survey and a set of swell noise recordings. A deep learning model is then trained to recognize the swell noise in the input seismic data and remove it (Valeciano et al., 2022). On inference, MDIO can sample similar patches from a seismic survey. The model then can remove noise automatically, reducing the time and expense of conventional time processing.

To show how MDIO can be used in model training pipelines, we provide a simplified example of a PyTorch (Paszke et al., 2019) data set, serving whole inlines from an MDIO file. The data set’s iteration logic can also be upgraded for more advanced slicing of crosslines, time slices, tiles, or volumetric data. The PyTorch data set object delegates I/O to MDIO (MDIOReader) in this code. The MDIO library then handles reading, slicing, and decoding the data from a seismic survey.

```

from torch.utils.data import Dataset
from mdio import MDIOReader

class InlineDataset(Dataset):
    def __init__(self, mdio_path):
        self.reader = MDIOReader(mdio_path)

    def __getitem__(self, idx):
        return self.reader[idx]

    def __len__(self):
        return self.reader.shape[0]

```

Wind resource assessment. Estimating the potential energy production of future wind farms is crucial in evaluating lease-bidding costs, deployment expenses, and return on investment. The growing demand for renewable energy sources is driving demand for detailed wind models across the globe. Like oil and gas exploration, wind farm assessments merge prior knowledge with physics-based modeling. Sophisticated tools such as the Weather Research and Forecasting modeling system (Skamarock et al., 2019) generate weather models with wind speed, orientation, and auxiliary data sets. Terabytes of data per study area result from the spatiotemporal nature of these simulations, often involving multiple 4D tensors with time, elevation, latitude, and

longitude coordinates. High-resolution studies focus on small areas with 100 m spatial and 30-minute temporal resolutions, while regional studies typically employ 1 km spatial and 1-hour temporal resolutions. In both cases, terabytes of data must be processed for analytics.

Wind models estimate a complete historical record of wind conditions across a study area. The time series are often reduced (via averaging or computing histograms) into monthly and yearly aggregates across the domain to derive meaningful insights from the data sets. While many operations are inherently parallel and involve simple calculations, they necessitate orthogonal and semi-random data access. The NetCDF4 format based on HDF5 is commonly used for storing climate data. By converting them to MDIO, we realize optimized I/O when computing a range of analytics. The primary calculations involve reducing monthly or daily grouped wind speed and direction data. Additionally, we fit

distributions and power-law curves to the reduced statistics using an embarrassingly parallel approach. After transitioning to optimized MDIO data sets, we leveraged the strengths of Zarr and Dask to distribute the processing more efficiently across a cluster of machines. This resulted in an approximately 100 times reduction in processing time compared to suboptimal NetCDF4 while maintaining the same resource allocation and software tooling. Figure 7 shows a sample of the aggregated statistics queried in real time by a web application with an MDIO backend.

Real-time visualization. As discussed in seismic imaging and data management applications, MDIO enables real-time visualization in web and desktop applications. Data can be streamed directly from the cloud to applications running on workstations, laptops, or mobile phones. For this use case, users typically want to be able to select and view 2D slices through large multidimensional seismic data sets. Common strategies

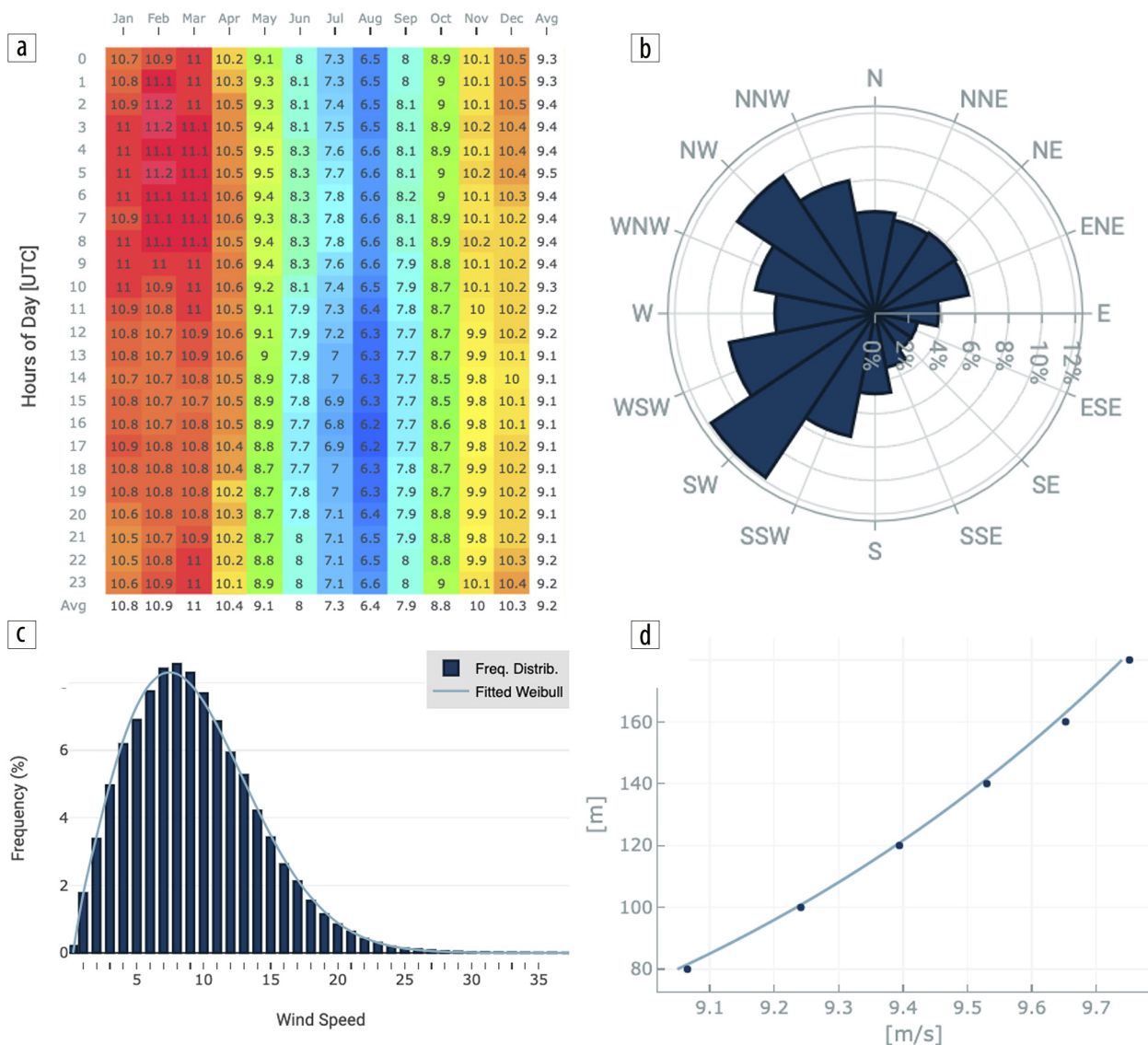


Figure 7. Sample plots of aggregated statistics queried in real time by a web application from a cloud object store using an optimized MDIO file. (a) The plot illustrates the wind direction distribution for a time range (also known as a wind rose). (b) An hour-of-the-year plot spanning months. (c) Depicts wind speed distribution alongside the best-fit Weibull distribution (used to calculate likely power output). Finally, (d) demonstrates wind speed variation by height, which aids in calculating wind shear across different turbine hub heights. These statistics, and more, are calculated for every point in the domain data set and provided for further analysis.

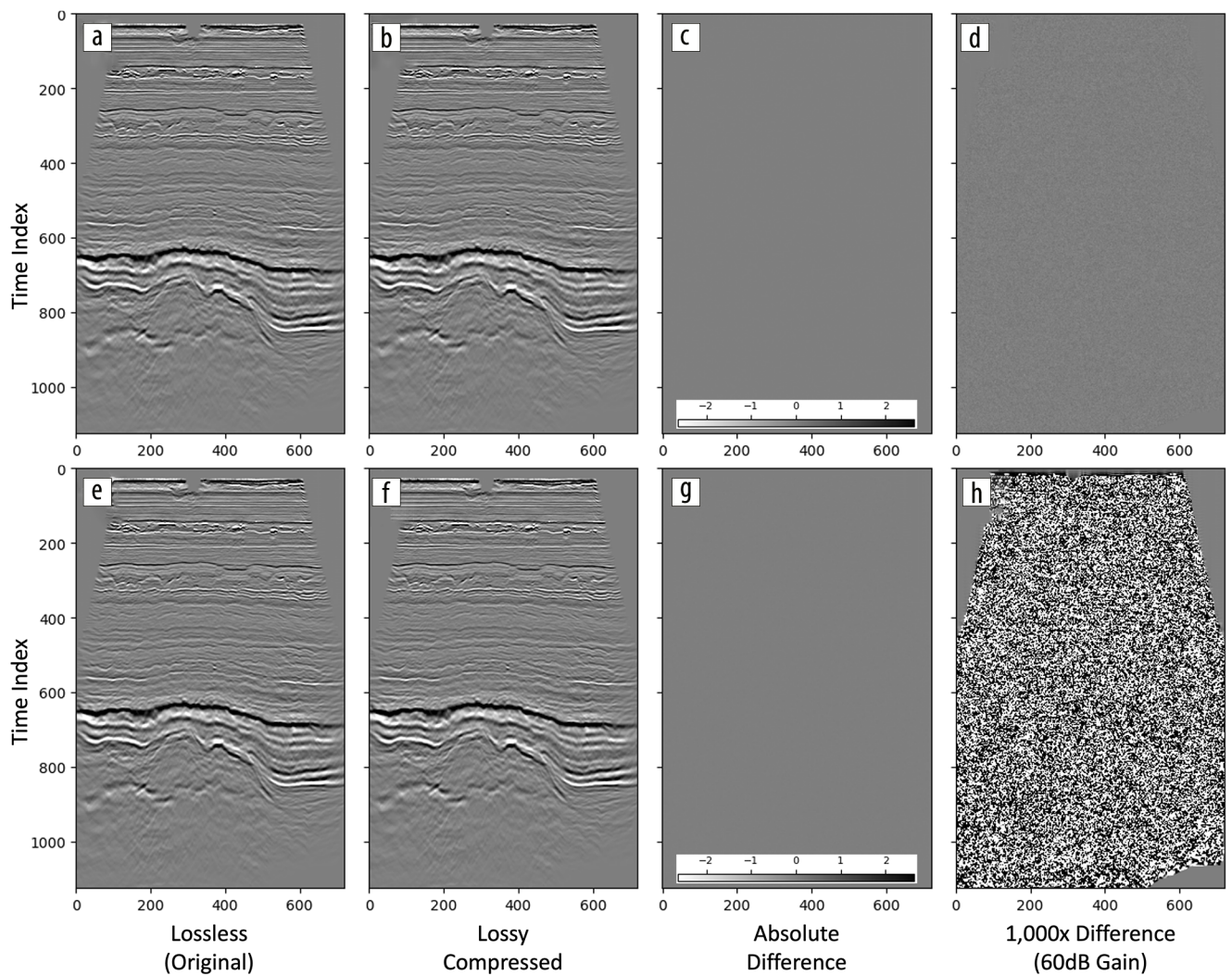


Figure 8. Images (a) and (e) show the original (float32) Volve angle stack data set. (b) A lossy version with a 5:1 compression ratio. (c) The absolute difference between (a) and (b). (d) The difference (c) gained by 1000x (60 dB). (f) A 25:1 compressed version of (e). (g and h) Absolute and gained differences between (e) and (f)

for making this process performant in all dimensions include creating transposed copies of the underlying data and employing concurrent multithreaded or multiprocessing reads. Data compression is vital to minimize storage costs and reduce bottlenecks in streaming data to a client application. Zarr naturally supports all of these requirements (discussed by Ravasi and Vasconcelos, 2021). Integration with Dask can make concurrency more cost effective in a cloud environment. Zarr chunking can also be configured to ensure that read performance is isotropic or optimized for a specific direction.

Minimizing the transmitted data volume in all use cases is advantageous, particularly for visualization where full precision float data are not generally needed. In this case, it is beneficial to use lossy compression. Figure 8 shows the result of ZFP compression on seismic data. The data set shown is an angle stack of the Volve survey (courtesy of data owners, published under CC by 4.0 license). The 5:1 compressed data are perceptually lossless, and the 60 dB gained difference shows no signal leakage. The 25:1 compressed data show a minimal absolute

difference, and the gained difference shows minor leakage. The data quality can be tuned to the downstream application needs and available bandwidth.

Usage examples

MDIO comes with a Python API for importing, exporting, and consuming data sets, accompanied by an easy-to-use and modular command line interface (CLI) designed for batch import and export workflows.

In this demonstration, we showcase its usage. Throughout the examples, we employ the openly accessible Teapot Dome filtered migration data set, courtesy of SEG, which can be downloaded from https://wiki.seg.org/wiki/Teapot_dome_3D_survey. Additional arguments and CLI options are available for importing, exporting, and accessing the data. For a more comprehensive look, we encourage readers to see the documentation page at <https://mdio.dev>.

Installing and importing SEG-Y. MDIO can be installed using either Pip or Conda packaging tools:

Using Pip:

```
pip install multidimio
```

Using Conda via the conda-forge channel:

```
conda install -c conda-forge multidimio
```

Using the Python API:

```
from mdio import mdio_to_seggy, seggy_to_mdio
from mdio import MDIOReader

seggy_to_mdio(
    seggy_path="filt_mig.sgy",
    mdio_path_or_buffer="filt_mig.mdio",
    index_bytes=(181, 185),
    index_names=("inline", "crossline"),
)
```

One can achieve the same result using the CLI:

```
mdio seggy import \
-i filt_mig.sgy \
-o filt_mig.mdio \
-loc 181,185 \
-names inline,crossline
```

Reading data. Data can be accessed using an MDIOReader instance. This reader offers options to return trace data exclusively or a tuple of the live mask, headers, and trace data. There are two methods to initialize the reader:

```
# with live mask and headers
mdio = MDIOReader(
    "filt_mig.mdio",
    return_metadata=True,
)

# without; returns just trace data
mdio = MDIOReader("filt_mig.mdio")
```

After initializing the reader, we can access attributes such as

- `mdio.text_header` — Return the parsed text header.
- `mdio.binary_header` — Return the parsed binary header.
- `mdio.grid` — Return the grid information. More information can be queried:
 - `mdio.grid.dim_names`
 - `mdio.grid.get_min("inline")` or `mdio.grid.get_max("sample")`
 - `mdio.grid.select_dim("crossline")`

There are also some convenient methods to translate between real and logical coordinates (i.e., querying an inline). For example,

the following snippet gives us the index for inline number 278, which can then be used to slice the data set accordingly:

```
index = int(
    mdio.coord_to_index(278, dimensions="inline")
)
```

Once we have the index, traditional NumPy and Python semantics can slice the data set. Based on how the reader was initialized, the data (or data + metadata) can be unpacked the following way. The headers and the live mask are read from the “metadata” group, whereas samples are read from the “data” group of MDIO.

```
# without metadata
samples = mdio[index]

# with metadata
live, hdr, samp = mdio[index]
```

Export. Using the Python API:

```
mdio_to_seggy(
    mdio_path_or_buffer="filt_mig.mdio",
    output_seggy_path="filt_mig_roundtrip.sgy",
)
```

One can achieve the same result using the CLI:

```
mdio seggy export \
-i filt_mig.mdio \
-o filt_mig_roundtrip.sgy
```

If we were using a cloud backend such as AWS S3, Azure Blob Storage, or Google Cloud Storage, we could simply provide the protocol and the prefix to the MDIO arrays. The following will ingest a local SEG-Y to AWS S3 using the Python API:


```
from mdio import MDIOReader, mdio_to_seggy,
seggy_to_mdio

seggy_to_mdio(
    seggy_path="filt_mig.sgy",
    mdio_path_or_buffer="s3://prefix/filt_mig.mdio",
    index_bytes=(181, 185),
    index_names=("inline", "crossline"),
)

mdio = MDIOReader("s3://prefix/filt_mig.mdio")
```

The same also applies to reading or exporting the data. Everything except the SEG-Y files can be directly accessed from the cloud provider. The SEG-Y files are not accessible from the cloud due to Segyio’s limitations. The authors are planning to adopt a cloud-compatible SEG-Y reader once available.

Conclusions

MDIO is a versatile and efficient library for managing multidimensional energy data that provides standardized storage and processing capabilities. Built as an extension of the Zarr protocol, it simplifies data sharing, collaboration, and analysis while offering optimized default chunk dimensions for various seismic and wind data types. By addressing the challenges of data management and cloud object storage, MDIO is a valuable resource for researchers, practitioners, and developers in the energy sector. Under the Apache 2.0 license, the actively maintained open-source library installs easily through popular channels such as Conda and Pip. 

Acknowledgments

We thank Sathiya Namasivayam, Charles Nguyen, Mohammad Nuruzzaman, Cable Warren, and the rest of the TGS Data and Analytics team for their valuable discussions and support. We also appreciate TGS for permitting us to publish this material. Additionally, we acknowledge the organizations and companies that generously provide public data sets and open-source tools for the community's benefit. We thank Vladimir Kazei, Matteo Ravasi, and an anonymous reviewer whose insightful comments improved this manuscript.

Data and materials availability

Data associated with this research are confidential and cannot be released.

Corresponding author: alejandro.valenciano@tgs.com

References

- Abma, R., and N. Kabir, 2006, 3D interpolation of irregular data with a POCS algorithm: *Geophysics*, **71**, no. 6, E91–E97, <https://doi.org/10.1190/1.2356088>.
- Barré, M., S. Walley, V. Savajol, J. Deng, C. Rhodes, A. van Welden, P. Endresen, M. Storm-Olsen, and A. James, 2022, Implementation of cloud-ready technology designed to address longstanding data interoperability and accessibility issues through the use of a comprehensive semi-automated seismic interpretation workflow: Second International Meeting for Applied Geoscience & Energy, SEG/AAPG, Expanded Abstracts, 1477–1481, <https://doi.org/10.1190/image2022-3743148.1>.
- Claerbout, J., 1991, Introduction to Seplib and SEP utility software: SEP-70: Stanford Exploration Project, 413–436.
- Collet, Y., 2011, LZ4: <http://lz4.github.io/lz4/>.
- Collet, Y., and M. Kucherawy, 2018, Zstandard Compression and the application/zstd media type: RFC Editor.
- Equinor, 2023, Segyio: Fast Python library for SEG Y files, <https://github.com/equinor/seggyio>, accessed 28 March 2023.
- Fomel, S., 2013, Revisiting SEP tour with Madagascar and SCons: *Journal of Open Research Software*.
- Google Cloud, 2023, Request rate and access distribution guidelines, accessed 21 March 2023, <https://cloud.google.com/storage/docs/request-rate>.
- Lindstrom, P., 2014, Fixed-rate compressed floating-point arrays: *IEEE Transactions on Visualization and Computer Graphics*, **20**, no. 12, 2674–2683, <https://doi.org/10.1109/TVCG.2014.2346458>.
- Maitin-Shepard, J., and L. Leavitt, 2022, TensorStore for high-performance, scalable array storage, <https://ai.googleblog.com/2022/09/tensorstore-for-high-performance.html>, accessed 1 June 2023.
- Miles, A., jakirkham, M. Bussonnier, J. Moore, D. P. Orfanos, A. Fulton, J. Bourbeau, et al., 2023, *zarr-developers/zarr-python: v2.14.1*: Zenodo, <https://doi.org/10.5281/zenodo.7633154>.
- OSDU, 2023, Open ZGY, <https://community.opengroup.org/osdu/platform/domain-data-mgmt-services/seismic/open-zgy>, accessed 21 March 2023.
- Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, et al., 2019, PyTorch: An imperative style, high-performance deep learning library: Cornell University, <https://doi.org/10.48550/arXiv.1912.01703>.
- Ravasi, M., and I. Vasconcelos, 2021, An open-source framework for the implementation of large-scale integral operators with flexible, modern high-performance computing solutions: Enabling 3D Marchenko imaging by least-squares inversion: *Geophysics*, **86**, no. 5, WC177–WC194, <https://doi.org/10.1190/geo2020-0796.1>.
- Rocklin, M., 2015, Dask: Parallel computation with blocked algorithms and task scheduling: Presented at the 14th Python in Science Conference.
- Skamarock, W. C., J. B. Klemp, J. Dudhia, D. O. Gill, Z. Liu, J. Berner, W. Wang, et al., 2019, A description of the advanced research WRF Model version 4.1: NCAR Technical Notes, <https://doi.org/10.5065/1dfh-6p97>.
- Society of Exploration Geophysicists, 2002, SEG Y rev 1 data exchange format, https://library.seg.org/pb-assets/technical-standards/seg_y_rev1-1686080991247.pdf, accessed 1 June 2023.
- The HDF Group, 1997–2023, Hierarchical Data Format, version 5, 1997–2023, <https://www.hdfgroup.org/HDF5>.
- Unidata, 2021, Network common data form (NetCDF): Unidata, <https://doi.org/10.5065/D6H70CW6>.
- Valenciano, A., O. Brusova, and C. Cheng, 2022, Efficient swell noise removal using a global deep neural network model: *First Break*, **40**, no. 2, 51–55, <https://doi.org/10.3997/1365-2397.fb2022012>.