EAGE DIGITAL 2020

EAGE

Seamless translation of modern file formats to SEG-Y through the file system interface

Lucas C. Villa Real
IBM Research
lucasvr@br.ibm.com

Maximilien de Bayser
IBM Research
mbayser@br.ibm.com

Introduction

Seismic processing and analysis is at the center of several activities ranging from exploration and production of reservoirs to hydraulic fracturing and underground gas storage. Supporting these industries there is a decades-old open standard that is ubiquitous in software and surveying equipments: SEG-Y.

Despite its wide adoption in the industry, the SEG-Y file format is inneficient in many ways: there is no support for data compression nor for spatial indexes; headers and data are interleaved on disk, meaning that certain header attributes may only be found through sequencial file scans; several metadata are made optional by the standard and are frequently missing from files (e.g., coordinates are present in the file while the spatial reference system is not), etc.

Alternative formats to SEG-Y such as (Krischer et al., 2016; Hess et al., 2017) do not present the same limitations. However, incorporating support for such formats into legacy software is not always possible. Consequently, companies are faced with the challenge of converting archived SEG-Y files into the new format, a task that often proves expensive and time consuming, or to manage two separate software stacks (one for each file format).

This paper presents a virtual file system interface that exposes modern seismic file formats as if they were SEG-Y – preventing the modification of existing programs while saving storage space when such formats incorporate data compression features. While our study uses an in-house self-describing file format based on HDF5 as backing storage, the technique applies to other implementations as well.

The SEG-Y data format

The first version of SEG-Y was defined in 1975 specifically for recording seismic data on 9-track tapes and has become ever since the de-facto standard for seismic storage. Although the second version from 2002 makes it independent from tape devices (Hagelund and Levin, 2017), it is still defined as a "data stream format", meaning that random access cannot be assumed. It also continues to include support for legacy standards such as the EBDIC character encoding and the IBM floating point format that were standard on the IBM mainframes and tape hardware back in the day (Barry et al., 1975). Accordingly the byte ordering is Big Endian in contrast to the Little Endian ordering that is prevalent today, even on variants of the new IBM Power CPUs.

The format, shown in Figure 1, is defined as follows: the stream starts with a fixed 3200 EBDIC text header, followed by a 400 byte binary header and by a variable number of 3200 byte extended text headers. After that comes a stream of pairs of trace header and trace data. The trace header is 240 bytes long while the trace data is variable in length. The length of each trace can be given in the trace header, although it is also possible to define a fixed length for all traces in the file header.

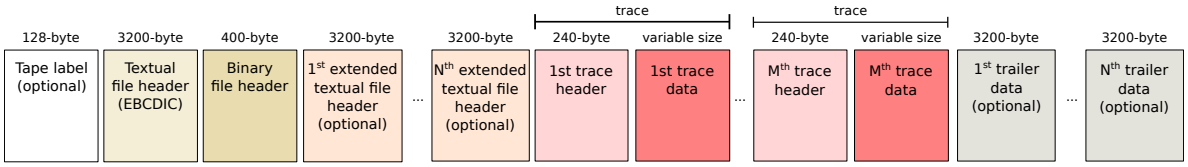

Figure 1 *SEG-Y file format (revision 2)*

The specification for file and trace headers defines a rich set of metadata, such as timestamps, coordinates and transduction units. In practice, many SEG-Y files' metadata are filled up with zeros, except for coordinates (often stored in the wrong field). Worse, the standard does not define a metadata field for the geographic coordinate system that was used, making the interpretation of coordinates a guesswork in many cases.

Even though the general design makes sense for the requirements of recording and transmitting a live stream of data coming from sensors, the SEG-Y format is not well suited for the needs of processing and archival. It induces an inefficient use of computing resources and contains unnecessary complexity that can only be explained due to its long history. The standard documents itself in some places offers alternative interpretation to the meaning and correct usage of some fields.

Arguably, the biggest issue is the lack of support for random access. Although the 2002 version of the standard includes a metadata flag for fixed trace length, due to the prevalence of malformed headers one cannot confidently use random access in these cases.

Another issue is the interleaving of trace metadata and data. In many computing tasks, such as indexing, only the metadata portion is needed. If all the metadata headers were contiguous, large chunks of metadata could be read directly from disk to memory very efficiently, offloading the entire operation to a DMA controller. Trace data could also be read efficiently if it were not for the Big Endian byte ordering. Data interleaving also reduces the efficiency of common lossless compression algorithms such as Gzip. These algorithms work best if they can work on large chunks of similar data.

Finally, although the standard is open, the lack of a reference implementation of libraries and tooling encourages different interpretations and shortcuts that negatively affects the interoperability between files generated by different tools.

Improved file formats for seismic data

There have been recently several proposals of alternative representation for seismic data on disk. The most prominent of these are based on HDF5 (Folk et al., 2011), a popular container for scientific and high-performance applications.

ASDF, the Adaptable Seismic Data Format, is one such format designed for seismology based on self-describing files. The format is intended to be used by researchers and analysts; it does not aspire to replace formats suited for data archival, streaming, and low-latency applications Krischer et al. (2016).

Another format is PH5 (Hess et al., 2017), which uses a feature from HDF5 that allows compressed waveforms to be stored separately from metadata. By doing so, updates to the metadata can be performed without having to reprocess and retransmit the entire dataset of an experiment to the end users. The drawback is that data is no longer self-contained in a single file; users need to keep track of the main header file plus several waveform files in order to distribute complete datasets across different machines.

SeismicH5 is an in-house file format that we implement as part of our seismic studies. Based on HDF5, its datasets and metadata (which may be compressed) provide a 1:1 mapping to SEG-Y that ease the conversion between the two. Differently from SEG-Y, our format stores traces contiguously on disk, which allows random seeks to be performed at constant time and improve compression rates. SeismicH5

retains all metadata featured in the SEG-Y specification under the following HDF5 *datasets*:

`/text_headers:` the EBCDIC textual file header, encoded as a string data type.
`/binary_header:` a compound data type representing the binary file header.
`/trace_headers:` a compound with all members of the SEG-Y trace header structure. This dataset is written in fixed sized chunks; appending of new trace headers is possible.
`/traces:` represents the trace data as chunks of IEEE single-precision floating point numbers. Differently from SEG-Y, they are always encoded in Little-Endian.

Many tasks in the processing of seismic data rely heavily on metadata attributes. For example, given that the position of a sound source is known one might want to find all traces that come from hydrophones in the same seismic line. Since the data from different hydrophones might be distributed almost randomly on the file, it is essential to use indices to speed these operations up.

In our format we have added the possibility to embed indices that are based upon the meta-data fields determined by the user. These indices are entirely optional and can be discarded and regenerated at will. Indices and metadata are represented by the following extra datasets:

`/metadata:` mixed data types indicating the original sample format (IEEE, IBM, 1-, 2-, or 4-byte integers), the size of the original SEG-Y file, and other flags needed to convert the file back to SEG-Y.
`/indices:` several indices to speed up searches for traces based on specific metadata.

Disguising modern seismic file formats as SEG-Y

Despite the advantages of file formats based on HDF5 containers, most legacy programs cannot be readily modified: sometimes their source code is not available, the companies who developed those software no longer exist, or the development and testing efforts may be too high.

To work around this problem we implement SEGY-FUSE, a virtual file system interface that enables legacy programs to consume data from modern file formats (such as our SeismicH5 prototype) without modification to their source code. Given a collection of seismic files encoded as HDF5, our approach exposes the same set of files as *virtual* SEG-Y files under a different file system namespace. When read requests (comprised of file offset and how many bytes to read) arrive from the application on that namespace, the mapping information embedded in the HDF5 file is looked up and the contents of the SEG-Y file are reconstructed on-the-fly.

Because the translation of file offsets to HDF5 datasets is only needed when an application requests to read a SEG-Y file, no information besides the basic file attributes (i.e., ownership, access permissions, and timestamps) needs to stay resident in memory. This approach allows a just-in-time strategy for the allocation of data structures and parsing of HDF5 datasets, leading to minimal consumption of memory resources, even when reading extremely large datasets.

Depending on the application and on limitations of the file system interface, the read buffer provided by the application (in which the SEG-Y data is expected to be written) can be potentially smaller than the size of a seismic trace. Consequently, it is possible that two or more sequential requests made to SEGY-FUSE translate to the same trace. To prevent the same trace from being retrieved from HDF5 twice (which can be especially expensive when trace data is compressed on disk), the last trace requested by the application is kept uncompressed in memory until either (i) the file is closed by the application or (ii) the application issues a read on another offset of the file that is covered by a different trace.

An overview of our architecture, based on the FUSE file system framework (Rath and Szeredi, 2019), is given in Figure 2. A collection of seismic files in HDF5 format is managed by an instance of the virtual file system. On top of it there is a collection of virtual SEG-Y files that map 1:1 to the HDF5 underneath. Existing applications are then able to seamlessly consume the data from the HDF5 files through SEGY-FUSE as if they were regular SEG-Y files.

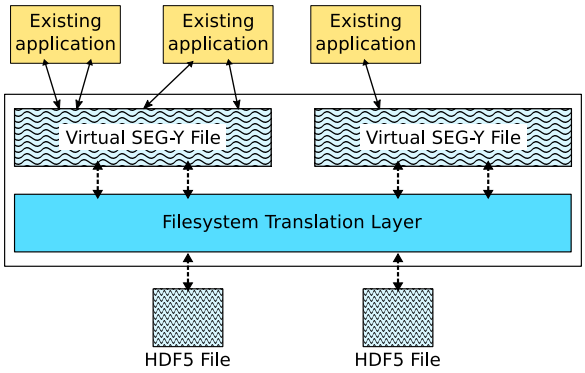

Figure 2 *Backwards compatibility with SEG-Y through the file system interface*

Performance considerations

This section presents the performance impact of the file system virtualization on file readers. We note that, although the SeismicH5 supports compression of trace data and headers using several standard algorithms, an analysis of compression formats is not in the scope of this paper.

We used a synthetic dataset with 3,744,501 uniform traces with 1504 samples each, totalling 6.1 GiB. The same file compresses down to 5.6 GiB with an LZF filter. Three tests have been conducted: (1) reading the original SEG-Y file over a native EXT4 file system, (2) reading that same file over a *pass-through* FUSE file system, and (3) reading the virtualized SEG-Y file through our SEGY-FUSE translator. System caches were cleared prior to each of the 10 runs.

Reading mode	**Time (seconds)**	**Overhead**
SEG-Y over EXT4	38.6 ± 0.15	–
SEG-Y over FUSE-Passthrough	38.8 ± 0.19	0.05%
SEG-Y over SEGY-FUSE	44.6 ± 0.19	15%
SEG-Y over SEGY-FUSE (LZF compressed)	71.6 ± 0.28	84%

Table 1 *Time to read a full seismic file*

As shown in Table 1, the FUSE file system framework imposes a negligible overhead when reading the whole file over a *pass-through* layer. In our current implementation, dynamically translating from HDF5 to SEG-Y is 15% slower in comparison to the time needed to read the original file. That slowdown comes mostly from I/O serialization, which we are looking to overcome by using asynchronous I/O and prefetching strategies. We also note that the single-threaded LZF decompressor puts a noticeable impact on data retrieval, suggesting that alternative compression algorithms may be more suited for this task. Still, the implementation as it stands proves to be useful enough for several real-life use cases.

Conclusion

We have shown in this paper that it is possible to have modern and legacy seismic file formats coexisting in a virtualized environment that avoids the replication of data at the storage layer. Using HDF5 containers as a reference in our study, we demonstrate that translation between file formats is possible with minimal performance impact at runtime, as long as the information required to recreate the SEG-Y file can be efficiently retrieved from its corresponding container.

The advantage of our approach is twofold. First, it is economically more attractive to have a seamless translation mechanism than to invest on the acquisition of new software or to modify existing systems. Second, it allows integration into the rich scientific computing ecosystem around the HDF5 format, which includes I/O libraries for high-performance-computing, visualization tools, plug-ins, and a vibrant community of users and developers alike.

References

Barry, K., Cavers, D. and Kneale, C. [1975] Recommended standards for digital tape formats. *Geophysics*, **40**(2), 344–352.

Folk, M., Heber, G., Koziol, Q., Pourmal, E. and Robinson, D. [2011] An overview of the HDF5 technology suite and its applications. 36–47.

Hagelund, R. and Levin, S.A. (Eds.) [2017] *SEG-Y revision 2.0 Data Exchange format*.

Hess, D., Azevedo, S., Falco, N. and Beaudoin, B.C. [2017] PH5: HDF5 Based Format for Integrating and Archiving Seismic Data. *AGU Fall Meeting Abstracts*.

Krischer, L. et al. [2016] An Adaptable Seismic Data Format. *Geophysical Journal International*, **207**(2), 1003–1011.

Rath, N. and Szeredi, M. [2019] The reference implementation of the Linux FUSE (File System in Userspace) interface.