An Evaluation of Hierarchical Data Format Version 5 (HDF5) as a Storage Model for Archive and Delivery of Video Data

Prepared by:
General Dynamics Information Technology, Inc.
294 Thames Avenue
Bay St. Louis, MS 39520
29 June 2016

Introduction: What is HDF5?

HDF5 is both a file format and a data model designed to support many heterogeneous data types, with an eye toward managing large, complex scientific datasets. For example, it has been used to store remotely-sensed satellite data, data from nuclear testing models, neurobiological data, and high-resolution MRI brain scans. It is designed to be highly portable and is supported by Java, C++, FORTRAN and Python language bindings.

HDF5 is scalable, fast and optimized. It supports parallel and network input/output (I/O) as well as standard, local disk-based I/O. It is extensible to allow developers to write additional file drivers to support other data storage frameworks. Its data storage model utilizes various compression and hyperslab (a.k.a. “chunking”) techniques to provide optimum access and efficiency.

The user may store heterogeneous data types together within one HDF5 file without regard for file size or the number of data objects in a file. Complex data relationships may be preserved through grouping and linking mechanisms, and metadata may be stored alongside the data to support archiving and data sharing. [1]

The Scenario

The question was put to us: can HDF5 be used as a storage format for OER video and used interactively in a Web-based user session? More specifically, the sponsor envisioned an interactive Web map containing a depiction of a dive track on the sea bottom, with points of interest along the track highlighted by markers. Selecting the marker would cause the browser to display a video clip taken at that location. Telemetry from the submersible, extracted from the ship’s onboard Scientific Computing System (SCS) and time-synchronized to the video stream, would also be displayed alongside the video. Is it possible to store these data objects together in an HDF5 file and display them to a user on demand?

Secondarily, we were also asked about the suitability of HDF5 not only as readily available online format but also as long-term archival storage. We answered this question first.

Is HDF5 suitable as an archive format?

The United States National Archives lists five criteria to evaluate when determining the propriety of a data format for archive purposes:

- The format should be publicly and openly documented;
- non-proprietary;
- in widespread use;
- self-documenting;
- can be opened, read, and accessed using readily-available tools. [2]

Publicly and openly documented

The HDF5 format and data model are well-documented. The HDF5 Group provides copious on-line documentation for the HDF5 format and user interface, including tutorials, a user's guide, a developer's reference guide, Release Notes for every version, and Frequently Asked Questions. These resources are available on-line as well as in their packaged source code distributions.

Non-Proprietary

HDF5 is an open standard and is non-proprietary in nature. The HDF5 library and developed code may be redistributed under a Berkely Software Distribution-like open source license, completely unrestricted with a minimum set of conditions; namely, the inclusion of copyright notices and disclaimers in any distributed source code. [3]

Widespread Use

HDF5 is used by dozens of scientific entities in the United States, including NASA, the Lawrence Livermore National Laboratory and the Environmental Protection Agency. Internationally, HDF5 is used by agencies in Australia, Canada, France, India, Russia, New Zealand, China, Senegal, and Italy, among many others. Uses range from energy research, medical imaging and analysis, computing research, high-resolution atmospheric and oceanographic modeling and bioinformatics. [4]

HDF5 is supported on Linux and Windows operating systems, available as precompiled binary distributions and RedHat managed packages (a.k.a. “RPMs”), and it may also be built from source code. [5]

Self-documenting

HDF5 is self-documenting in a technical sense. Applications may examine the structure and contents of a file without requiring external information to read it. Internally, user-defined attributes may be created to add descriptive metadata, applied globally to a file or to individual data objects within the file. Basic HDF5 data model features may be used to create and store data attributes arbitrarily. [6]

Readily-Available Tools

Several user-developed software tools come with the HDF5 distribution. These tools provide editing, conversion, integrity-checking and interactive editing capabilities. The Java-based HDFView application provides a graphical interactive file browser and editor. [7]

Based on these criteria, it is apparent that HDF5 would make a good candidate for an archive format. *As of this writing HDF5 is not on any discoverable list of approved archive formats.*

Can HDF5 be used in an on-demand video context?

Possibly. One may think of an HDF5 file as a “container”. It can store an arbitrary number of data objects and can organize them internally. This internal organization resembles a directory tree on a computer disk drive; HDF5 “Groups” are analogous to directory folders in that they may contain any number of arbitrary data objects (“Datasets”), including other Groups. Combined with user-defined metadata objects, which may be stored alongside the data objects they describe, one can express a rich, descriptive data structure within a single HDF5 file, or across multiple HDF5s.

But HDF5 is designed to store scientific data and not media, although images may be readily stored there. Indeed, the HDF5 data model contains a high-level “Image” API for Datasets meant to be interpreted as images. In this context the images are static full-frame images, typically renderings of model outputs or high-resolution satellite or medical imagery.

There have been experiments conducted to explore the feasibility of using HDF5 as a native video container. These approaches invariably treat each video frame as an image and store each image frame separately using the Image API. Still, to "play" these animations externally from the HDF5 file, these individual frames must first be reconstructed as a video file--that is, they must be extracted from HDF5 and written to disk in a video file format. Alternately, an application which can read HDF5 files, such as *HDFView*, must be written to display the frames in sequence. [8, 9]

This approach would only work on a video file format in which each frame is a complete image. But OER video, for instance, is stored in H.264 (MPEG Part 10) format. H.264 uses a compression technique known as *block-oriented motion-compensation compression*. This technique uses a combination of "key frames" followed by data blocks which contain only the differences between the previous frame and the next frame. [10, 11] As such we wouldn't be able to use the image-frame-per-Dataset

technique, since each frame in an H.264 video file is not necessarily a complete image frame.

Therefore we have determined that the only way to store video data in an HDF5 file is as a BLOB (Binary Large OBject), an opaque data object which represents an arbitrary collection of digital information. To deliver the video file, or to play it, will require first extracting the video file from HDF5 to disk or memory.

Possible Implementations Using HDF

To date there are no video player software applications which can play video natively from HDF5. In all cases, in order to play a video file stored in HDF5 it must first be extracted to disk or memory and then loaded into a player or delivered to a client. This does not necessarily mean that one shouldn't store video data in HDF5. It just may not be appropriate in an on-demand context.

As part of our research we conducted a conversation with Dr. Ted Habermann, formerly of NOAA and now Earth Science Director for the HDF Group. After discussion of the scenario and state of the art with regard to video technology and HDF5, we all agreed that there was no feasible way to play video directly from an HDF5 file. We did discuss various other possible implementations:

1. Store the video file as a BLOB in one HDF5 Dataset; store ancillary data in another Dataset, or store multiple video segments of interest along with appropriately-subsetted ancillary data in related Datasets. The ancillary data may be associated with specific time instances in the video through indexing, or with specific video segments. The ancillary data or an internal data index would be used to locate and describe each video segment.

2. Store "frame grabs" and ancillary data associated with the frame's time index in multiple Datasets. But then the resulting entity would no longer be video, and there would be no audio channel.

3. "Tidy Packaging" (Dr. Habermann's term): Store video as a BLOB together with ancillary data, but no indexing.

Fundamentally, these are all just different forms of packaging. One could technically achieve the same thing with a Linux tarball or a Zip archive, and avoid the requirement of the HDF5 library and tools. But HDF5 data access is meant to be fast and random, and Linux 'tar' is a slow, sequential format, while Zip is primarily intended for distribution packaging, archiving and temporary compression.

Conclusion

HDF5 can readily serve as a container format for video data files and ancillary information. Generally speaking HDF5 should be a suitable candidate for an approved archive format. It meets the criteria set forth by the National Archives, although consideration should be taken with regard to any technical requirements incurred with regard to installing and maintaining HDF5 software. Regarding the question of whether HDF5 should be used as a storage format for OER video and ancillary data, by virtue of HDF5's data agnosticism and internal organizational structure, it should be possible in theory to create a data model to support the use case described at the beginning of this document. However, to fully explore the idea behind the question, it would be necessary to determine through experimentation whether any latency associated with using HDF5 as a storage model in a high-performance environment will offset any advantages realized elsewhere.

References

[1] The HDF Group. "What is HDF5?" https://www.hdfgroup.org/HDF5/whatishdf5.html (accessed 29 June 2016).

[2] U.S. National Archives. "Frequently Asked Questions (FAQ) About Digital Audio and Video Records." http://www.archives.gov/records-mgmt/initiatives/dav-faq.html (accessed 29 June 2016).

[3] The HDF Group. "Licenses and Copyrights." https://www.hdfgroup.org/products/licenses.html (accessed 29 June 2016).

[4] The HDF Group. "HDF5 Users." https://www.hdfgroup.org/HDF5/users5.html (accessed 29 June 2016).

[5] The HDF Group. "Obtaining the Latest HDF5 Software." https://www.hdfgroup.org/HDF5/release/obtain5.html (accessed 29 June 2016).

[6] The HDF Group. "Why HDF?" https://www.hdfgroup.org/why_hdf/ (accessed 29 June 2016).

[7] The HDF Group. "HDF Tools.' https://www.hdfgroup.org/tools/ (accessed 29 June 2016).

[8] Kiraly, Robert. "An HDF5 to Video Exercise." http://oldcoder.org/general/h5dumptopng/ (accessed 29 June 2016).

[9] The HDF Group. “Image Viewer.” https://www.hdfgroup.org/products/java/hdfview/UsersGuide/ug06imageview.html (accessed 29 June 2016).

[10] Wikipedia.org. “H.264/MPEG-4 AVC.” https://en.wikipedia.org/wiki/H.264/MPEG-4_AVC (accessed 29 June 2016).

[11] Wikipedia.org. “Motion compensation.” https://en.wikipedia.org/wiki/Motion_compensation (accessed 29 June 2016).