HEP Reconstruction and Analysis Software for Exascale-Era Computing:

Progress Report (Phase I Deliverable)

Dana Robinson

The HDF Group

September 16, 2016

# Introduction

The primary objective of this project is to produce a prototype software system suitable for moving HEP experiment event data through multiple processing stages in an exascale-class computing facility. In particular, work will concentrate on demonstration of two critical components of a complete event-processing system: (a) high-performance I/O to a parallel filesystem, and (b) communication of event data through high-performance node interconnects (rather than through the filesystem), between processes of one application running across hundreds or thousands of nodes - a distributed program.

The full project has two stated goals:

1. Provide a plan to design a file schema using HDF5, suitable for the storage of experimental HEP data.

2. Determination of what, if any modifications of the organization of HEP-specific data structures are needed.

The work produced so far has largely consisted of both FNAL and THG working together to understand each other's software and to clarify the specifics concerning the execution of the remainder of the project. In this document, the work that will take place for the remainder of the project will be described in greater detail than in the original statement of work. Also, although not detailed extensively here, some preliminary exploratory modeling work has taken place using h5py and FNAL has begun to extend some of their framework classes so that the I/O programs will be easier to evaluate.

# Decide on Three Use Cases and Confirm Access Patterns

The three data products we will model (the three "use cases") will be:

* vector of MCHitCollection
* vector of MCTruth
* vector of associations between cluster and vertex data products

FNAL has provided THG with UML diagrams for these data products.

An important aspect of the association data product is that the "association" values should be more than simple binary "association present" values and must associate more complicated data values. The associations data product listed here uses an unsigned integer association value. It is also a requirement that the association data product associate data spread across multiple files (i.e., one half of the associated data in one file, the other half in a second file, and the associations themselves in a third file) to simulate the output of multiple data reconstruction pipelines in FNAL software. These cross-file associations will have to be tolerant of missing cluster and vertex data.

Each data product is consumed by FNAL software "by event". Data for an event are read, processing occurs, data are written back out, and the in-memory data are discarded. There is no need for cross-event data, even in the associations data products. Not all data stored for an event are consumed by each analysis program, though for this work all data will be consumed in the I/O programs.

# Generate Layouts/Schemas for the Data Products

Since HDF5 does not have a standard schema language, THG will provide C/C++[[1]](#footnote-1) programs that create empty (or trivially filled with dummy data) containers that can be inspected by HDFView, h5dump, etc. as well as a discussion concerning the particulars of the HDF5 structures and the reasoning behind them.

In general, straightforward event-based layouts will be preferred without worrying about performance or file size since the most direct FNAL --> HDF5 --> FNAL mapping should be the easiest to integrate into a future DDL-driven automated pipeline. If profiling shows unsatisfactory performance, the simpler structures will be refactored as necessary.

# Write Data Read/Write programs

Once HDF5 schema/containers have been defined, THG (in conjunction with FNAL when it comes to their APIs) will write simple C++ programs to move the data from Root to memory to HDF5 and back to memory. These programs will be used as a basis for the data comparison and profiling programs described below.

# Profile Data Size Between HDF5 and Root

HDF5 data product size will be trivially determined by inspecting the size of the HDF5 file in the file system. FNAL will provide Root data product sizes since these are easily determined from their software.

The final write-up will include a discussion of compression filters in HDF5, how chunking affects the modeled data products, and how data could potentially be re-arranged for better compression.

# Compare In-Memory Data Products Constructed from HDF5 and Root

A program will be constructed that loads identical data product event data that were stored in both Root and HDF5 and compares them to ensure that HDF5 is storing equivalent data to the original Root file. This will be done for each data product. The comparison will be performed using code provided by FNAL.

# Profile I/O speed between HDF5 and Root

We can use the data I/O programs constructed earlier as a base to generate profiling programs that are as identical as possible aside from the I/O calls. We’ll profile both write speed (i.e., serializing the in-memory representation to disk for both formats) and read speed (vice-versa). Profiling will be done in a straightforward manner, using optimized builds of the I/O stack and test programs and either monitoring wall-clock time via the Unix time command or using the C++ time library internally (or Intel TBB if higher resolution is necessary). If time permits, it would also be useful to perform per-event profiling.

FNAL has indicated that the file size should not significantly influence the I/O performance of Root data access, so we can use actual FNAL data files for profiling. There will be no need to copy data to files that only contain the data products under analysis.

# Final Paper

The final write-up will include a discussion of all of the above, but will also include a discussion of how an HDF5-based stack could be parallelized (producer-consumer, MPI, etc.). We will also lay out the groundwork for extending the work done under this contract.

1. C/C++ programs will be provided to FNAL but the HDF5 C library will be used in all cases, and not the C++ wrappers. [↑](#footnote-ref-1)