Date: **March 1, 2013**	**Design Document – HDF5 Dynamic Data Structure Support** **FOR EXTREME-SCALE COMPUTING RESEARCH AND DEVELOPMENT (FAST FORWARD) STORAGE AND I/O**

LLNS Subcontract No.	B599860
Subcontractor Name	Intel Federal LLC
Subcontractor Address	2200 Mission College Blvd. Santa Clara, CA 95052

Table of Contents

Revision History

Date	Revision	Author
3/1/2013	1.0 Posted for internal FF review	Ruth Aydt and Quincey Koziol, The HDF Group

Introduction

This document discusses extensions to HDF5 to support the needs of the Big Data–HPC Bridge Arbitrarily Connected Graph (ACG) work.

At project inception, we believed a new HDF5 pointer datatype that would allow dynamic data structures in memory (such as linked lists or directed graphs) to be traversed and stored within an HDF5 container would be required. Subsequent discussions with the Intel ACG team led to a better understanding of the graph representations and access patterns inherent to the machine learning and graph analytics computational kernel, and prompted a modification in wording from "pointer datatype" to "dynamic data structure support" in the revised SOW dated February 8, 2013.

Ongoing discussions with the ACG team have resulted in an initial HDF5 representation of the ACGs, presented in Section 4.3 of the Big Data–HPC Bridge design document, that does not require any extensions to the existing HDF5 data model. That said, until we have actual experience with the proposed representation and its performance on the Exascale Fast Forward I/O stack, it is difficult to be certain that the proposed representation will be acceptable.

We feel that moving forward with the proposed representation, allowing the ACG team to gain experience with the HDF5 data model and exercising capabilities of the IOD and DAOS layers as they become available, is appropriate at this time. We anticipate adjustments to the representation as we gain experience—both to address issues that arise related to ease-of-use from the computational kernel perspective, and to leverage performance advantages offered by IOD and DAOS.

A number of possible HDF5 extensions that could be beneficial to ACGs have been identified in our discussions to date. These extensions, presented later, remain under active consideration for implementation if the need arises.

Definitions

- Arbitrarily Connected Graph (ACG): A graph with arbitrary edge relationships.
- Computational Kernel (CK): The application framework that supports the Exascale structured machine learning and graph analytics.
- Network Information (NI): Arbitrarily-typed data structures associated with vertices and edges of an ACG.[1]
- Power-Law Graph: A graph made up of vertices whose degrees follow a power-law distribution. That is, a graph in which a small number of vertices are connected to a large percentage of the edges.

[1] Sometimes the term "Network Information" is used to refer exclusively to vertex and edge data extracted from the initial raw data when the graph was built. In this document, we also use the term to refer to variables associated with vertices and edges that are updated as part of the graph computation.

- Vertex degree: The number of edges connected to a vertex.

Changes from Solution Architecture

The HDF5 Solution Architecture focused on extending HDF5 with pointer datatypes. As explained in the Introduction above, HDF5 pointer datatypes may not offer the best option given the data structures and access patterns inherent to the computational kernel.

We remain committed to the original intent, which is to support the ACG work with extensions to HDF5 datatypes or data structures if needed.

Specification

As background, we first present some characteristics of ACGs and computational kernels (CKs). Next, we discuss how these characteristics have influenced the initial HDF5 representation of the ACGs, presented in Section 4.3 of the Big Data–HPC Bridge design document. We conclude the section by presenting the HDF5 extensions identified to date that could be beneficial to ACGs, briefly describing where they might be used.

Background

In the most general terms, an ACG is composed of a set of vertices connected by edges, where the connections may be directional. There can be data, referred to as network information (NI), associated with the vertices and with the edges.

While there are considerable differences across graphs, for a given ACG the connections will be of the same type (directed or not), the NI for the edges will have the same structure, and the NI for the vertices will have the same structure. This means that while there is variability in the data structures for a wide range of ACGs, there is consistency—and knowledge before writing to HDF5—of the data characteristics for a given ACG.

The CK that performs the machine learning or graph analytics is designed to run an algorithm (a vertex program) on each vertex in the graph, potentially accessing some of the NI for the vertex, its edges, and its neighbors. The algorithm will typically update NI for the vertex, its edges, and/or its neighbors. Different vertex programs will access different NI. There are multiple "runs" of one or more vertex program(s) for each vertex in the graph during a single analysis. Different analyses may be run on the same ACG, meaning multiple "results NI" structures may be associated with the same ACG.

The CK will run on compute nodes of the HPC. Providing high-performance I/O for the CK, while minimizing memory used for data not needed by the algorithms or results, is our goal.

Given the nature of power-law graphs, dividing the vertices across processors to achieve balanced computation and to minimize (machine) network traffic is very challenging. GraphBuilder partially addresses this challenge, and GraphLab takes it further.

HDF5 Representation of ACGs

When thinking about how to represent ACGs in HDF5, the most natural representation from a graph perspective—a set of connected vertices with metadata associated with the vertices and the edges—is not well-suited for fast access by the CK. This natural representation was what we initially had in mind with the proposed HDF5 pointer (and/or enhanced HDF5 reference) datatypes, as they would allow the application to "follow the pointers" from one vertex to another. In addition, references in vertex objects could be used to reach associated NI, which we originally believed to be arbitrarily typed and varied, even within a single ACG.

The graph partitioning process already breaks the vertices into sets that do not strictly follow the graph connectivity. Furthermore, the CK often does much of its algorithmic work accessing only a fraction of the total NI. It would be a waste of bandwidth and memory to automatically load all of the NI for a vertex or edge, so storing the NI data with the vertex or edge object is unlikely the best option. Even storing pointers or references to the (unused) NI in the vertex or edge object is wasting space if the NI can be located—when needed—through another mechanism.

Segregating graph topology, immutable NI, mutable NI, and dictionary (vertex ID to string mapping produced by GraphBuilder) data into separate structures and separate HDF5 objects seems promising. We believe this approach will allow us to leverage the H5Group objects and Exascale FF stack hints to achieve partition-friendly data layout and well-targeted prefetch, and to replicate and free immutable data on multiple compute nodes without synchronization concerns. In addition, we believe careful mapping of the H5Datasets with variable length types to IOD objects by the IOD VOL will address the performance and space drawbacks these objects exhibited in the native HDF5 file format. This will be especially important for some ACG structures, such as adjacency lists, given the power-law graphs being represented.

The mutable NI that is updated frequently by the CK is relatively small in size, and may be staged as vertices and edges are loaded prior to processing by the active vertex program. NI for high-degree vertices (and their edges) may remain resident on the compute nodes if there is sufficient memory, while lower-degree vertices are shuffled in and out. Arranging the data in HDF5 Objects to facilitate optimizations such as these is our goal, improving execution time for the machine learning and graph analysis applications, and allowing very large ACGs to be processed.

Possible HDF5 Extensions

- *HDF5 Pointer Datatype:* An HDF5 pointer datatype would be used when the data being pointed to is contained within a single H5Dataset element.

  It would allow a linked list, tree, sub-graph or other dynamic data structure to be stored in each element of an H5Dataset.

- *HDF5 Local Element Reference:* An HDF5 local element reference would be used to reference another element in the same H5Dataset.

  It would provide a mechanism for moving through a dataset without having to specify the element indices directly. If the H5Dataset elements represented vertices, you could use a variable length list of local element references as the datatype for the H5Dataset, and populate each vertex element with local element references to the neighbors of the vertex. In effect, creating an adjacency list representation of the graph topology.

The local element reference is a simplification of the existing HDF5 "dataset region reference", which allows multiple elements (a region) in any H5Dataset to be referenced. The local element reference could be implemented with fixed size (storing coordinates of the element references) for fast storage and retrieval.

- *HDF5 Element Reference:* An HDF5 element reference would be used to reference an element in another H5Dataset.

  It would provide a mechanism for retrieving an element from another H5Dataset directly, without having to specify the dataset and element coordinates at access time. If the H5Datasets elements represented edges, an element reference could be used to reference the NI for an edge, where the NI is stored in a different H5Dataset.

  The element reference is also a simplification of the existing HDF5 "dataset region reference". As with the local element reference, the element reference could be implemented with a fixed size (storing the dataset object ID and the coordinates of the element referenced) for fast storage and retrieval.

These pointer and reference datatypes conceptually share the idea of "follow the pointer from here to there". The pointer type is used within a single dataset element. The reference types, including the existing HDF5 object reference and dataset region reference types, all point from one element to other element(s) or objects. The reference types point to data that is external to the element holding the reference.

With HDF5's compound datatype capabilities, a datatype for an H5Dataset could combine pointers and references to construct structures like linked lists of references to dataset elements.

- *HDF5 KV Store object:* An HDF5 KV Store object would provide applications with a direct interface to the IOD KV object.

  It would provide a mechanism for insertion and retrieval of key-value pairs that should perform very well. The key and the value would both be strings. Or, the HDF5 KV Store object might allow the user to specify one of the H5Datatypes for the key and another for the value, adding the potential for additional type checking and storage optimization.

  In the context of ACG representation, HDF5 KV Store objects could be used for NI (key = vertex or edge id, value = NI data) and for the dictionary produced by Graph Builder that maps strings found in the raw data to vertex ids (key = string, value = vertex id).

  In the initial representation of ACGs, presented in Section 4.3 of the Big Data–HPC Bridge design document, H5Attributes will be used to represent mutable NI variables associated with vertices and edges that are part of the graph computation's update cycle. Multiple HDF5 Attributes can be associated with a given HDF5 Object. On the Fast Forward stack, HDF5 Attributes will be implemented as key (attribute name) value (attribute value) pairs in IOD KV objects associated with HDF5 Objects. Each HDF5 Object will have its own IOD KV object to hold the HDF5 Attributes associated with it.

  The initial representation of mutable NI using HDF5 Attributes will let us gain experience with the IOD KV objects to assess their performance and appropriateness for use with ACGs and the CK, and may prompt a "first class" HDF5 KV Store object to be added. Given the nature of the CK processing, we may also want to allow hints that indicate if a given CN will be accessing the KV for read-only or for update, where a master node is in charge of updates for a given KV pair, but other CNs may read and dispose of the pair when done. For some algorithms, "outdated" values are also acceptable, allowing algorithms to proceed even without the latest NI data.

- *HDF5 Virtual Dataset: An HDF5 Virtual Dataset would be composed of pieces of other H5Datasets.*

  It would provide a mechanism for presenting a unified view of a set of HDF5 Dataset Objects that may be stored and accessed individually for performance and space reasons. While the unified view may not be the most appropriate for intense processing, it may be the most intuitive and appropriate for examining analysis results.

  In the context of representing ACGs, a virtual dataset might store the entire graph, which is composed of other datasets that represent partitions of the graph. In turn, those partitions could be virtual datasets composed of collections of vertices from other dataset(s).

API and Protocol Additions and Changes

Not applicable at this time.

Open Issues

The suitability of the proposed graph representation in HDF5 will be assessed as the project continues. Adjustments to the representation, including possible extensions to HDF5 to support the ACG work, may still be needed.

Risks & Unknowns

Until the computational kernel is running on the Exascale Fast Forward I/O stack, performance of various graph representations that rely on different underlying IOD objects (such as IOD KV objects versus IOD array objects) is unknown.

There is a risk in extending HDF5 to support ACGs until more experience is gained, both with the computational kernel on the HPC platform, and with the underlying stack. There is also a risk in delaying extensions, because of the relatively short project.

We plan to manage these risks by (1) continuing to work closely with the ACG, IOD, and DAOS teams, (2) keeping in mind the possible extensions that have been identified to date and others that may become apparent as more experience is gained, and (3) moving quickly to start implementation if the proposed representation does not meet the ACG needs.