

|  |  |
| --- | --- |
| **Date:March 21, 2013** | **Design Document – HDF5 to IOD Object Mapping****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 |

|  |
| --- |
| **Limited Rights Notice.  These data are submitted with limited rights under Prime Contract No. DE-AC52-07NA27344 between LLNL and the Government and Subcontract No. B599860 between LLNL and Intel Federal LLC.  This data may be reproduced and used by the Government with the express limitation that it will not, without written permission of Intel, be used for purposes of manufacture nor disclosed outside the Government.****The information contained herein is confidential and proprietary, and is considered a “trade secret” under 18 U.S.C. § 1905 (The Trade Secrets Act) and Exemption 4 to FOIA.  Release of this information is prohibited.** |

Table of

Table of Contents

Introduction 1

Definitions 1

Background 1

The HDF5 to IOD Object Mapping 2

1.1 HDF5 Group -> IOD Key Value Store Object 2

1.2 HDF5 File -> IOD Container 3

1.3 HDF5 Dataset -> IOD Array Object 4

1.4 Metadata format 5

Open Issues 5

Risks & Unknowns 5

**Revision History**

|  |  |  |
| --- | --- | --- |
| **Date** | **Revision** | **Author** |
| Mar. 21, 2013 | 1.0First Draft | Mohamad Chaarawi |
| Mar. 21, 2013 | 1.1More details on file format | Mohamad Chaarawi |

# Introduction

The document details how HDF5 File, Group, and Dataset objects map to IOD objects. The application will call the HDF5 library while running on the system’s compute nodes (CNs). Using the VOL architecture, the IOD VOL plugin will use a function shipper (FS) to forward the VOL calls to a server component running on the I/O nodes (IONs). At that point, the VOL calls are translated into I/O Dispatcher (IOD) API calls and executed at the IONs. The translation includes mapping HDF5 object into IOD objects. The IOD will be responsible for storing the data on distributed storage using DAOS.

# Definitions

CN = Compute Node

EFF = Exascale FastForward

FS = Function Shipper

IOD = I/O Dispatcher

ION = I/O Node

VOL = Virtual Object Layer

# Background

The HDF5 VOL intercepts all HDF5 calls that would potentially touch the storage and routes them to an internal or user-developed plugin. This allows for HDF5 objects to be stored in different file formats or storage abstractions that are hidden from the application, allowing the application to continue using the same HDF5 API and data model while benefiting from new storage methods and architectures.

The overall architecture of the HDF5 library with the addition of an IOD VOL plugin looks like this:



# The HDF5 to IOD Object Mapping

The EFF storage software stack contains several components essential to the proper functioning and performance of the application I/O. The scope of this document is from the HDF5 library to the high level IOD API routines. We will not discuss here how IOD implements its API internally or its interaction with the DAOS library and underlying distributed storage.

## HDF5 Group -> IOD Key Value Store Object

An HDF5 group maps to a IOD KV store object. To access the KV store object, one must have its IOD ID. From this ID, one can open that object and retrieve/add/update the KV pairs in that KV store object.

The KV pairs in the KV object describe links to the child objects that are linked to from this group. The key is a string that contains the link name and the value associated with that key contains that object’s IOD ID. For example, a group “G1” that is a parent of dataset “G1/D1” and another group “G1/G2” will contain the following KV pairs:

<”D1”, D1\_iod\_id>

<”G2”, G2\_iod\_id>

In addition to the “links” to its children, an HDF5 group has its own metadata associated with it (for example, the creation properties). Attribute objects can be attached to an HDF5 group, all which need to be stored and retrieved from the file. To be able to store that information, we will use the scratch pad field that IOD provides for each object. In this 32-byte scratch pad we will store two IOD IDs for auxiliary KV store objects, KV1 and KV2, that we create to store metadata and attribute information for the object, respectively. (KV1’s ID is stored in the upper 16 byte of the scratch pad and KV2’s ID in the lower 16 bytes).

Here is an example that summarizes the mapping of HDF5 group with ID G1\_iod\_id with two child objects and three attribute objects in IOD (Note that the file format for attributes is out of scope for this quarter’s deliverables for this document):



## HDF5 File -> IOD Container

An HDF5 file is mapped to an IOD container. The container is created/opened by the file name. In addition to that, the HDF5 library specifies that each file contains a root group. The root group always has a predefined ID (for now we will make that ID be 0). From that root ID, one can reach any object in the container by traversing the KV entries in the group KV store objects, retrieving the object names and corresponding IOD IDs of each group to reach the final object.

Since the root object is an actual HDF5 group, it will be stored in IOD the same way we store any group as described in 1.1. The following diagram shows an HDF5 file with a group /G1 under the root group, and an attribute “attribute1” attached to the file:



## HDF5 Dataset -> IOD Array Object

An HDF5 dataset is stored as an IOD array object. The IOD array dimensions are determined from the HDF5 dataspace specified by the application. Similarly, the size of each IOD array element is determined by the HDF5 datatype. The IOD array creation operation allows us to chunk a dataset in the same way that HDF5 allows the application to store a chunked dataset.

Similarly to groups, HDF5 stores metadata associated with datasets (i.e. the datatype, dataspace, etc…) and allows users to attach attributes to a dataset. This is handled again with an IOD scratch pad field in the array object that contains IDs for a auxiliary metadata KV store object and attribute KV store object for each dataset, stored in the same order as for the group scratch pad information. Here is a diagram that describes a dataset in IOD with attribute “attribute1”:



## Metadata format

The metadata stored in auxiliary KV objects for group and dataset objects is formatted in the same manner as specified in the HDF5 File Format Specification: <http://www.hdfgroup.org/HDF5/doc/H5.format.html>

# Open Issues

Information about how HDF5 Link, Attribute and Named Datatype objects will be finalized and included in a future revision of this document, due in Q4 of the project.

# Risks & Unknowns

The IOD implementation is not completely specified yet and details described above may still change if IOD’s features change in an incompatible way.