Date: June 17, 2013	# Milestone 4.4 - Data Integrity Report ## 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	History	Author
June 17, 2013	1.0	Submitted to DOE	Mohamad Chaarawi, Quincey Koziol – The HDF Group

Introduction

This report serves as a description for the data corruption fault generation and testing process in the HDF5 level of the Exascale FastForward software stack. Currently, the IOD and DAOS layers of the stack are not functional to the extent that they can be incorporated in the data corruption testing, but as they become functional, we will incorporate them into this report.

Definitions

CN = Compute Node

EFF = Exascale FastForward

FS = Function Shipper, also known as Mercury

IOD = I/O Dispatcher

ION = I/O Node

VOL = Virtual Object Layer

Background

End-to-end integrity and fault detection is of fundamental importance in the storage of data – there is no point in storing information without having confidence in being able to reliably determine if it has been corrupted before being read back in.

Fault Detection of HDF5 Data

As part of the EFF software stack's end-to-end integrity checking, checksums have been added to HDF5 data stored in IOD K-V store objects, to HDF5 data elements stored in IOD array objects, and to data in IOD blob objects. These objects encompass all the application data stored, so checksums applied to them will ensure fault detection of any corrupted application data.

Currently, the scope of data integrity checks covers data element operations (H5Dataset reads and writes) and not metadata, due to the limitations in the underlying IOD storage library at this stage of the project. The following diagram describes the end-to-end integrity verification for HDF5 data element I/O:

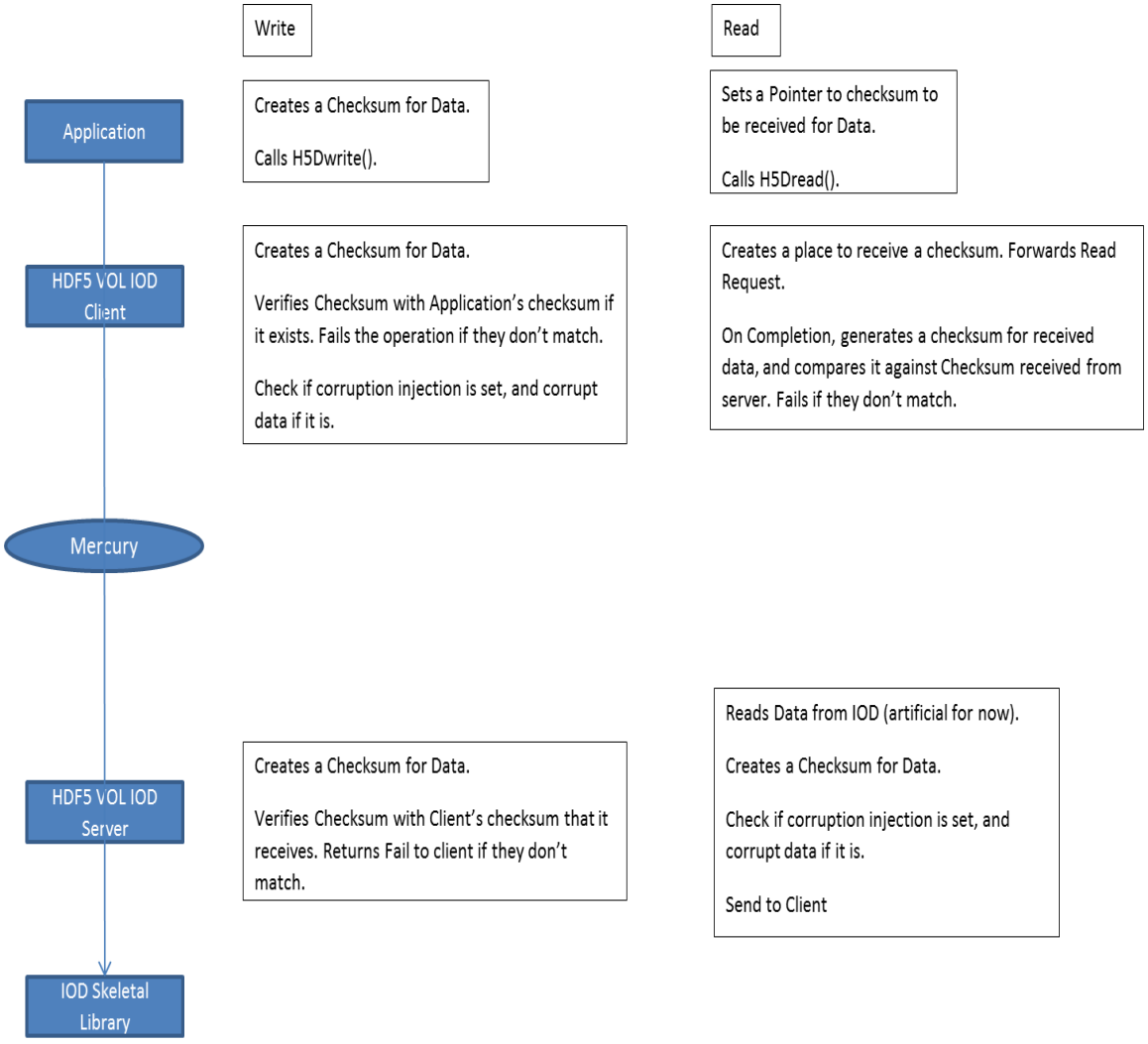

Figure 1: Data Integrity Workflow for HDF5 Dataset Elements

Note that the "corruption injection" property (in the HDF5 VOL IOD Client) is part of the integrity regression testing framework, not part of normal I/O operations.

Figure 1 shows the application using an optional checksum verification routine to give the HDF5 library a checksum value for the data written, which the HDF5 library will pass down the stack for verification. If the application does not provide a checksum, one is already generated internally at the VOL IOD client before data is shipped to the VOL IOD server. For read operations, the user will be able to provide a pointer to memory where the checksum can be stored when the read operation completes. The HDF5 API routines for generating and receiving checksums are:

```
herr_t H5Pset_dxpl_checksum(hid_t dxpl_id, uint32_t value);
```

```
herr_t H5Pset_dxpl_checksum_ptr(hid_t dxpl_id, uint32_t *value);
```

Note that all checksums given to the library must be generated using the checksum algorithm that HDF5 provides:

```
uint32_t H5checksum(const void *buf, size_t length, H5_checksum_seed_t *cs)
```

The H5checksum routine returns a checksum value for the data in *buf* with size *length* bytes. If the buffer is contiguous, the pointer to the checksum state structure, *cs*, can be NULL. Otherwise, for checksumming a set of non-contiguous regions that will be passed in a single call to H5Dwrite, *cs* captures the internal state of the checksum generation algorithm, allowing a single checksum to be generated for the set of non-contiguous data that is identical to the checksum that would be generated if the same data was checksummed in one contiguous block. The checksum state structure is defined as follows:

```
typedef struct H5_checksum_seed_t {

    uint32_t a;

    uint32_t b;

    uint32_t c;

    int32_t state;

    size_t total_length;

} H5_checksum_seed_t;
```

When creating a checksum for a set of non-contiguous buffers, H5checksum should be called on each contiguous portion of the buffer with that portion's corresponding length. The *a*, *b*, *c*, and *state* fields in the checksum state structure should be initialized to 0 for the first call to H5checksum, and *total_length* should be set to the total size in bytes of all the data to be checksummed (over all the non-contiguous sections of the data to checksum). Every subsequent call to H5checksum should use the previous *cs* value as a seed value to the current checksum call.

Testing Fault Detection of HDF5 Data

In order to verify that the fault detection facilities in the EFF software stack are operating correctly, we have added special properties that may be set to trigger introduction of data corruption at various levels in the software stack. Currently, as for the checksum verification above, this facility is only available for HDF5 data, but will be expanded to the IOD and DAOS layers as they become available.

Data corruption fault generation in HDF5 is indicated through a data transfer property that the user sets before issuing the I/O operation using that transfer property. The library will check for this flag and corrupt the data before it is sent across the network, if the property is set. The data corruption is detected at the target once the checksum of the corrupted data is generated and compared to the checksum that is sent along with the data itself.

The routine to inject a corruption in the data written is:

herr_t H5Pset_dxpl_inject_corruption(hid_t dxpl_id, hbool_t flag);

where corruption is injected if the flag is set to TRUE.

Open Issues

Metadata checksums are not yet generated by the HDF5 library, due to the early state of the IOD software.

As noted above, this report describes an early state of the end-to-end integrity in the EFF software stack. As facilities for storing data in IOD and DAOS become available, this report will be updated with information about fault detection in those layers.

Risks & Unknowns

The IOD and DAOS data integrity implementation is not completely specified yet and details described above may change if features in those layers change in an incompatible way.