Date: February 25, 2014	*Burst Buffer Space Management – Prototype and Production* **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

NOTICE: THIS MANUSCRIPT HAS BEEN AUTHORED BY THE HDF GROUP UNDER THE INTEL SUBCONTRACT WITH LAWRENCE LIVERMORE NATIONAL SECURITY, LLC WHO IS THE OPERATOR AND MANAGER OF LAWRENCE LIVERMORE NATIONAL LABORATORY UNDER CONTRACT NO. DE-AC52-07NA27344 WITH THE U.S. DEPARTMENT OF ENERGY. THE UNITED STATES GOVERNMENT RETAINS AND THE PUBLISHER, BY ACCEPTING THE ARTICLE OF PUBLICATION, ACKNOWLEDGES THAT THE UNITED STATES GOVERNMENT RETAINS A NON-EXCLUSIVE, PAID-UP, IRREVOCABLE, WORLD-WIDE LICENSE TO PUBLISH OR REPRODUCE THE PUBLISHED FORM OF THIS MANUSCRIPT, OR ALLOW OTHERS TO DO SO, FOR UNITED STATES GOVERNMENT PURPOSES. THE VIEWS AND OPINIONS OF AUTHORS EXPRESSED HEREIN DO NOT NECESSARILY REFLECT THOSE OF THE UNITED STATES GOVERNMENT OR LAWRENCE LIVERMORE NATIONAL SECURITY, LLC.

Copyright 2014, The HDF Group. All Rights Reserved.
B599860-SS

Table of Contents

Revision History ii

1 Introduction 1

2 Glossary 2

3 Transactions 2

4 Putting Data into the Burst Buffer: IOD 3
4.1 Add Updates to a Transaction 3
4.2 Replicate Readable Data from DAOS to BB 5
4.3 Possible Modifications for Production Version 5

5 Reading Data: IOD 6
5.1 Reading with a Tagged TID 6
5.2 Reading with a TID 6
5.3 Possible Modifications for Production Version 6

6 Removing Data from the Burst Buffer: IOD 6
6.1 Evicting with a Tagged TID 6
6.2 Evicting with a TID 7
6.3 Evicting Updates of Aborted Transactions 7
6.4 Possible Modifications for Production Version 7

7 Mapping of HDF5 Objects to IOD Objects 8

8 Putting Data into the Burst Buffer: HDF5 8
8.1 Add Updates to a Transaction 9
8.2 Replicate Readable Data 9
8.2.1 H5Dataset prefetch 10
8.2.1.1 Possible Modifications for Production Version 10
8.2.2 H5Group prefetch 11
8.2.2.1 Possible Modifications for Production Version 11
8.2.3 H5Map prefetch 11
8.2.3.1 Possible Modifications for Production Version 12
8.2.4 Prefetch Container Version – Possible Addition for Production Version 12

9 Reading Data: HDF5 12
9.1 Possible Modifications for Production Version 13

10 Removing Data from the Burst Buffer: HDF5 13
10.1 H5Dataset evict 13
10.2 H5Group evict 14
10.3 H5Map evict 14
10.4 Possible Additions for Production Version 14

Use or disclosure of data contained on this sheet is subject to the restriction on the title page of this document.
Copyright © The HDF Group, 2014. All rights reserved

Revision History

Date	Revision	Description	Author
Feb. 10-18, 2014	1.0 – 1.7	Initial drafts incorporating material from previous documents and discussions with EFF architecture team.	Ruth Aydt, The HDF Group
Feb 18, 2014	2.0	Version sent to DOE stakeholders for discussion at February stakeholder meeting.	Ruth Aydt
Feb 25, 2014	2.1	Update to reflect IOD's eviction of BB data for aborted transactions. Update colors in Fig 1 and fix minor typos.	Ruth Aydt

Use or disclosure of data contained on this sheet is subject to the restriction on the title page of this document.
Copyright © The HDF Group, 2014. All rights reserved

1 Introduction

The BB provides two primary functions in the context of the EFF stack. First, as the name suggests, it provides a buffer where bursty data can be written quickly and then drained off to spinning disk in the background at a slower rate between bursts. Second, it provides another level in the cache hierarchy of the system, where data that exceeds CN memory can reside for faster access than would be possible if read directly from disk.

This document describes data movement into and out of the BB, represented by the colored lines in Figure 1. Operations available at the IOD level of the software stack are covered first, followed by operations at the HDF5 level. Capabilities that will be delivered as part of the EFF prototype project are described, and some additional features that have been identified as potentially beneficial in a production version of the EFF stack are introduced.

This document covers these concepts at a fairly high level. Readers are directed to the *IOD Design Document* for a more complete discussion of the IOD capabilities and implementation. The *User's Guide to FastForward Features in HDF5* will be updated as part of the Q7 deliverables to fully document the prefetch and evict APIs for HDF5.

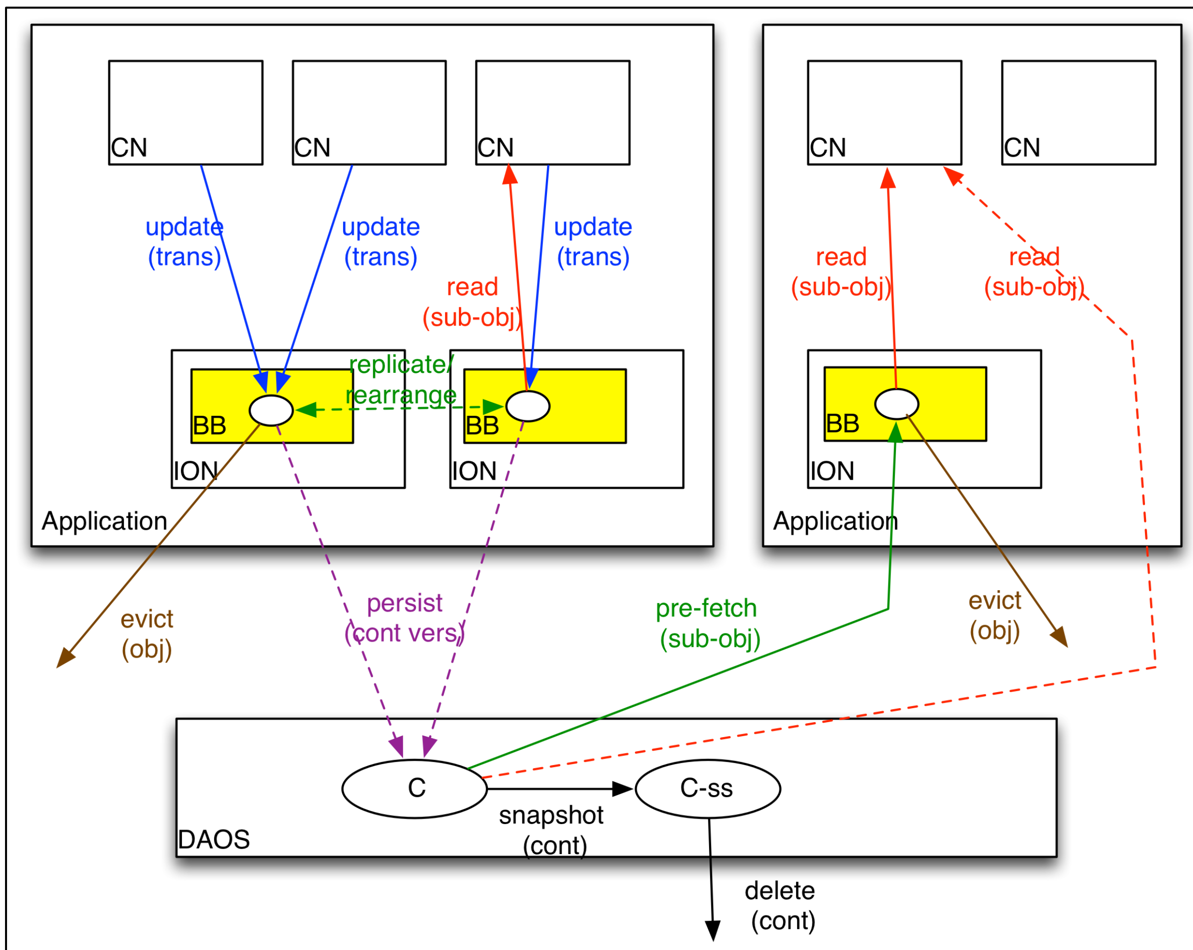

Figure 1: Burst Buffer Data Movement

Use or disclosure of data contained on this sheet is subject to the restriction on the title page of this document.
Copyright © The HDF Group, 2014. All rights reserved

2 Glossary

BB = Burst Buffer – physical storage media within the ION

CN = Compute Node

CV = Container Version

DAOS = Distributed Application Object Storage

commit = Atomically make a set of updates to a container visible.

delta = The difference in the data for a given object between two, possibly non-consecutive, CVs.

EFF = Exascale FastForward

IOD = I/O Dispatcher

ION = I/O Node

persist = Atomically commit a given container version to DAOS storage.

RC = Read Context – an open read handle on a CV

rcntxt_id = read context id at the HDF5 level

TID = Transaction ID at IOD level.

Tagged TID = Transaction ID with IOD-supplied tag used to identify a replica of part or all of an object at IOD level.

Tr # = Transaction Number

Updates = The changes to objects in a container in a given transaction. Updates are not visible to readers until the transaction is committed.

VL = Variable Length

VOL = Virtual Object Layer

3 Transactions

As background for the BB data movement discussions, we briefly summarize concepts and terminology used to talk about transactions in the EFF stack.

At the HDF5 level, a transaction holds a set of updates to a container that will all become visible at the same time. Recall that (1) multiple transactions can be in progress at any given time, (2) transactions can be finished[1] in any order, and (3) transactions are committed in strict numeric order when all lower-numbered transactions have been committed or aborted.

At the HDF5 level, the term *Container Version* refers to the state of the container after a transaction has been committed. When transaction #T is committed, the updates in the transaction are applied atomically to the container and a new container version #CV=T becomes readable. When there is an open read context (RC) on a given CV, the container contents of that CV are guaranteed to remain readable until the RC is closed. In addition, the container

[1] A transaction is finished when no more updates will be added to it.

Use or disclosure of data contained on this sheet is subject to the restriction on the title page of this document.
Copyright © The HDF Group, 2014. All rights reserved

contents of highest CV (the highest committed transaction) are guaranteed to be readable (an RC could be obtained for the CV) until another transaction is committed.

In IOD, the terminology is slightly different and a transaction ID (TID) is used to refer to:

a) An IOD write transaction
   - Same usage as at HDF5 level
b) An IOD read transaction
   - Similar to the RC at the HDF5 level
c) A replica of data from an object or sub-object in a read transaction, where the replicated data is in the BB; replica creation is discussed further in the next section. This type of TID is also known as a "Tagged TID".
   - At the HDF5 level, this is referred to as a Replica ID, and must always be used in conjunction with an Object ID and a RC.

4 Putting Data into the Burst Buffer: IOD

There are two primary IOD-user-initiated ways for data to be written to the BB.

1. Add IOD object updates to a transaction.

   - IOD objects can be Arrays, KV Stores, or Blobs.

   - Object updates can add, modify, or delete data, for example "add a new KV entry", "change the value of the KV entry with key = k", or "delete KV entry with key = k"

   - Object updates are stored in log format to the BB, and the data becomes readable when the transaction is committed.

2. Replicate some or all of the readable data in an IOD object.

   - If the source of the replicated data is DAOS, this is a *prefetch*

   - If the source of the replicated data is the BB, this is referred to by IOD as a *multi-format replica* or *semantic resharding*, depending on the type of object.

     BB->BB replication is not discussed further in this document and is not supported by the prototype HDF5 APIs for EFF.

Other IOD-user-initiated operations, such as creating an IOD container or an IOD object, also add data to the Burst Buffer. This document ignores the issue of container and object creation, focusing on the data movement related to object contents.

In our discussions we will assume all objects are in a single container and we will primarily focus our examples on Array and KV objects.

4.1 Add Updates to a Transaction

The IOD APIs *iod_array_write, iod_array_write_list, iod_kv_set, iod_kv_set_list, iod_kv_unlink_keys, iod_blob_write, iod_blob_write_list, and iod_blob_append* add Array, KV Store, and Blob object updates to a transaction. All take a container handle, a TID, one or more object handles, an indication of "where" the data should be written in the object (the Array

Use or disclosure of data contained on this sheet is subject to the restriction on the title page of this document.
Copyright © The HDF Group, 2014. All rights reserved

cell(s), the key(s), the byte range(s) or "at the end"), and—with the exception of *iod_kv_unlink_keys*—the data.

Tables 1 and 2 represent the user-level data in five transactions and the resulting container versions.

Tr #	Array Updates (1D, datatype=int, 5 elements)	KV Store Updates
5	5, 5, 5, 5, 5	[A, 5] [B, 5]
4	_, _, _, _, _	[B, 4]
3	_, _, 3, 3, 3	del[A]
2	_, 2, 2, _, _	[B, 2]
1	1, 1, 1, 1, 1	[A, 1]

Table 1: IOD object updates in five transactions

CV #	Array Data	Needed Updates	KV Store Data	Needed Updates
1	1, 1, 1, 1, 1	1	[A, 1]	1
2	1, 2, 2, 1, 1	1, 2	[A, 1] [B, 2]	1, 2
3	1, 2, 3, 3, 3	1, 2, 3	[B, 2]	1, 2, 3
4	1, 2, 3, 3, 3	1, 2, 3	[B, 4]	3, 4
5	5, 5, 5, 5, 5	5	[A, 5] [B, 5]	5

Table 2: IOD object contents in five container versions and updates needed to read each version

Table 1 illustrates updates for two IOD objects (Array and KV Store) added to five transactions. For the Array, commas separate cell values, and _'s are used to indicate no updates to a given cell in a given transaction. For the KV Store, the nomenclature is [key,value], with del[key] meaning a key (and its value) is deleted from the store.

Object updates are logged to the BB by IOD as they are added to a transaction. The transactions are shown in reverse numeric order in Table 1 because updates in later transactions are effectively "layered" onto earlier committed transactions.

Table 2 shows the data values in the two IOD objects (Array and KV Store) at each CV after all five transactions have been committed. The table also indicates the updates (identified by transaction number) that are needed to access the object data at a given CV when the data is physically arranged in the original log format.

A CV can be *persisted* to DAOS in an atomic commit from IOD to DAOS, triggered by an IOD-user request. Every CV need not be persisted. When a CV is persisted, IOD figures out the *delta* (the data that has changed) since the last persisted CV, and transfers only that data to

Use or disclosure of data contained on this sheet is subject to the restriction on the title page of this document.
Copyright © The HDF Group, 2014. All rights reserved

DAOS; it does not write the complete container contents for the CV being persisted, nor does it write the individual updates for every CV since the last persisted CV.

> For example, referring to Table 1, say CV 1 has already been persisted and the user requests that CV 3 be persisted. Conceptually, IOD will transfer the delta {A:"_,2,3,3,3", KV:del[A], [B, 2]} to DAOS and atomically commit the changes.

4.2 Replicate Readable Data from DAOS to BB

The IOD API *iod_obj_fetch* is used to make a *replica* of data that is stored on DAOS in the BB. This API takes an object handle, a TID indicating the CV, and parameters that control which data in the object will be replicated and how the replicated data will be laid out in the BB.

In Quarter 7, IOD will support fetching of complete objects. In Quarter 8, support for object subranges, and mapping of the replicated data to different layouts on the IONs will be added for Array, KV Store, and Blob objects.

When *iod_obj_fetch* is called to create a replica, IOD returns a special *Tagged TID* that must be used to read data from the replica, or to evict the replica from the BB. The Tagged TID has the same type as IOD's TID. But, it contains not only the TID value that was passed into the call (the value that indicates the desired CV), but also a tag that identifies the replica to IOD. Every Tagged TID associated with the same container is unique.

4.3 Possible Modifications for Production Version

IOD's current prototype implementation, where a Tagged TID is returned when an object (or sub-object) is prefetched, has a number of limitations that recently came to light.

The Tagged TID mixes the specification of "which copy" of a subset of the container data with the specification of which version of the container contents are being requested. From a user perspective, the different dimensions (which copy on storage/which version in the evolution of the container contents) make it hard to manage the TIDs consistently. Having the ability to reference "this copy of data" separate from "as of this version of the container" would be preferred.

In addition, the current Tagged TIDs do not allow the HDF5 layer (or other users of IOD) to manipulate multiple related IOD objects together. The Tagged TID is assigned per-fetch, and the fetch granularity is per object (or sub-object). This is especially challenging when trying to take advantage of the BB as part of the cache hierarchy. The desire to manipulate multiple IOD objects as a whole will be discussed in more detail in the HDF5 sections of this document.

In general, it seems that the identifiers for "which CV", "which Object", "which Replica" might be specified and tracked separately. This would allow triplets of the three to specify a particular copy of data for a particular object at a particular container version. Or, to use wildcards for one or more components (e.g., [CV=x, Object=*, Replica=y] --> evict all the data in the BB that is in CV x and that was fetched together); [CV=*, Object=z, Replica = *] --> evict all the data in the BB for Object z, regardless of the version or replica it is part of.) Along these lines, the "which Replica" could be passed in by the caller, allowing a way to create a batch of related prefetches across multiple objects that could later be evicted together.

Use or disclosure of data contained on this sheet is subject to the restriction on the title page of this document.
Copyright © The HDF Group, 2014. All rights reserved

5 Reading Data: IOD

The IOD APIs *iod_array_read, iod_array_read_list, iod_kv_get_value, iod_blob_read, iod_blob_read_list, and iod_blob_append* read data from Array, KV Store, and Blob objects. Additional IOD APIs not enumerated here also perform read operations to retrieve information about the Array, about the number of entries in a KV store, and so on.

All IOD APIs that read data from the BB take a container handle, a TID (or Tagged TID), one or more object handles, an indication of "where" the data can be found in the object (the Array cell(s), the key(s), the byte range(s)), and where in memory the read data should be stored.

5.1 Reading with a Tagged TID

If a Tagged TID is passed to the IOD call, the referenced replica will be used to read the data. If all of the data requested is not available from the replica, IOD will retrieve the remaining data from DAOS.

When IOD reads data from DAOS, it passes through the memory of the ION on its way to the memory of the CN, but does not get copied into the BB.

5.2 Reading with a TID

If a (non-tagged) TID is passed to the IOD call, IOD will retrieve data from the appropriate updates, if they are still in the BB, and will retrieve the remaining data from DAOS.

When IOD reads data from DAOS, it passes through the memory of the ION on its way to the memory of the CN, but does not get copied into the BB.

5.3 Possible Modifications for Production Version

Allowing the user to specify the replica to read gives considerable power, but also imposes a burden. In a production version, allowing the user to specify the CV and object data desired, and having IOD find the closest copy of the data – either in the BB or on DAOS – would be preferred. The user could still specify the replica, but would not have to. If the data requested was not all in available from a specified replica, IOD would look for it first elsewhere in the BB and then in DAOS.

6 Removing Data from the Burst Buffer: IOD

The IOD API *iod_obj_purge* removes data from the BB, freeing the space to be used for other data. This call takes an object handle and a TID (or Tagged TID).

Although the IOD API uses the term "purge", the team later decided to call this operation *evict* in the documentation and in the HDF5 API, and that is the term we use here. Regardless of the term used, the resulting action is that data is removed from the BB and no copy of the data is made before it is removed.

6.1 Evicting with a Tagged TID

If a Tagged TID is passed to the call, the referenced replica for the object (or sub-object) will be evicted from the BB.

Use or disclosure of data contained on this sheet is subject to the restriction on the title page of this document.
Copyright © The HDF Group, 2014. All rights reserved

6.2 Evicting with a TID

A (non-tagged) TID must always be persisted before it can be evicted. If a (non-tagged) TID is passed, and the evict would not violate the guarantees about the readability of CVs with open RCs and readability of the highest CV, IOD will evict all updates for the object with update numbers less than or equal to the transaction number indicated by the TID.

If the user tries to evict an object that violates the readability guarantees, the call to *iod_obj_purge* will fail.

The reader is directed to the IOD Design Document for a more complete description of the evict guidelines used by IOD.

6.3 Evicting Updates of Aborted Transactions

When a transaction is aborted—either by the user or by the library in response to transaction dependencies—IOD automatically evicts updates in the BB associated with the aborted transaction. In addition, IOD will not log any updates for the aborted transaction that arrive after the abort.

6.4 Possible Modifications for Production Version

With the exception of clean-up after an aborted transaction, IOD leaves Burst Buffer space management entirely under the control of user-initiated calls in the EFF prototype implementation. While there are certainly times when the ability to manage the space at a per-update / per-object granularity can have enormous benefit, having some assistance from IOD in other cases would make it easier for no-longer-needed data to be removed from the BB.

Higher-level libraries and applications will not always have direct access to all of the object handles that have been created/opened, and the current IOD API requires an object be open in order for data in the object to be evicted from the BB. In addition, there are time when the ability to specify a "larger granularity" of data to evict would be helpful

For example:

- "Evict all data for all objects at this CV"
  - Useful when a CV has been persisted and there is no expectation that any data will be read from it, as in a checkpoint.
  - Useful when the higher levels don't (or can't) track each object and/or when higher level isn't designed to hold objects open until the data in the BB for the object is evicted. (e.g., VPIC)
- "Evict all data for all objects in this container"
  - Useful at file close
- "Evict all data for this object at this CV"
  - Useful when you are done analyzing a timestep and you want to get rid of multiple replicas.

While not possible with the current prefetch/replica implementation, having the ability to prefetch data for multiple objects at a given CV as a batch, and having a single replica ID associated with that batch that could be used to evict all of the objects at once would be useful in workflows that incrementally step through data held in multiple related objects.

Use or disclosure of data contained on this sheet is subject to the restriction on the title page of this document.
Copyright © The HDF Group, 2014. All rights reserved

Another observation is that the use cases and behaviors for the eviction of updates (data written to the container in transactions) and replicas (copies of data) seem quite different. In the case of eviction of updates, you are removing data from the BB that resulted from write operations, and the log structure of the updates means care must be taken not to evict data that is needed to satisfy read operations for later CVs. In the case of the replicas, you are removing data from the BB that was there to optimize read operations, and the likelihood that you want to prefetch/evict batches of data from multiple objects in a coordinated fashion is high. Perhaps the management of these two types of BB data should be done with different APIs to reflect the underlying fact that one is clearing out "bursty write data" and the other "cached read data".

7 Mapping of HDF5 Objects to IOD Objects

At the HDF5 layer of the stack, the primary data storage object types are H5Group, H5Dataset, H5Map, H5NamedDatatype, and H5Attribute.

As discussed in the document *HDF5 Data in IOD Containers Layout Specification*, most of the HDF5 objects are mapped to multiple IOD objects by the HDF5 to IOD VOL. This mapping is summarized in Table 3.

The user of HDF5 makes calls to the HDF5 API that operation on HDF5 objects, and the HDF5 IOD VOL layer translates those operations into IOD API calls on IOD objects. Our discussions in this document will focus primarily on the H5Dataset and H5Map objects.

HDF5 Object	Primary IOD Object	Auxiliary IOD Objects
H5Group	KV	KV (metadata) KV(Attributes) Arrays (data for attributes)
H5Dataset	Array	KV (metadata) KV(Attributes) Arrays (data for Attributes) Blobs, when variable length data in cells
H5Map	KV	KV (metadata) Array (Attributes) Arrays (data for Attributes)
H5NamedDatatype	Blob	KV (metadata) Array (Attributes) Arrays (data for Attributes)
H5Attribute	Array	No auxiliary IOD Objects; H5Attributes are always associated with another H5Object or with the H5File (container)

Table 3: Mapping of HDF5 Objects to IOD Objects

8 Putting Data into the Burst Buffer: HDF5

As was the case at the IOD level, there are two primary HDF5-user-initiated ways for data to be written to the BB.

Use or disclosure of data contained on this sheet is subject to the restriction on the title page of this document.
Copyright © The HDF Group, 2014. All rights reserved

1. Add HDF5 object updates to a transaction.
    - HDF5 objects can be Groups, Datasets, Maps, NamedDatatypes, and Attributes.
    - Updates to the HDF5 objects are translated into updates to the IOD Array, KV Store, and Blob objects that are used to instantiate the HDF5 objects on the EFF stack.
2. Replicate some or all of the readable data in an HDF5 object from DAOS to the BB.
    - This is a *prefetch.*

8.1 Add Updates to a Transaction

The HDF5 APIs *H5Dwrite_FF and H5Mset_FF* respectively add H5Dataset and H5Map object updates to a transaction. Please refer to the *User's Guide to FastForward Features in HDF5* document, updated quarterly, for details on these APIs and others that operate on the complete set of HDF5 objects with the EFF transactional semantics.

All HDF5 APIs that add updates to a transaction take an object ID, a transaction ID, and other parameters conveying the details of the update. The HDF5 operations made by an HDF5 user are translated by the HDF5-IOD VOL into IOD calls (see Section 4.1) on the related IOD objects.

8.2 Replicate Readable Data

The HDF5 API offers prefetch commands corresponding to each of the HDF5 object types.

At the HDF5 level, a user must have a read context (RC) on a CV in order to read or prefetch data from HDF5 Objects, because data is read from committed transactions and the RC guarantees that consistent data can be read for all objects in the container at the CV. In the HDF5 API, a read context id (rcntxt_id) is one of the parameters in every API used to read data from the H5File (the container). The rcntxt_id parameter is an input parameter–there is no expectation that it will be modified by the calls that access readable data.

The IOD fetch design and implementation delivered in the prototype phase of the EFF project presents some problems for HDF5, primarily due to its use of a Tagged TID with replicated object data. These problems were not noticed when the IOD API was proposed, but became more apparent as design and implementation at the HDF5 level progressed.

When the HDF5-IOD VOL maps an HDF5 Object to multiple IOD Objects, as shown in Table 3, all of the IOD Objects are passed the same TID, obtained from the rcntxt_id. Since the IOD API returns a unique Tagged TID for each IOD object that is fetched, there would be multiple Tagged TIDs to track for each HDF5 Object (except H5Attributes).

For the prototype phase, only the Primary IOD Object will be prefetched and tracked when the HDF5 user issues a prefetch request for an HDF5 Object. Some of the possible IOD-level modifications that were outlined in earlier sections would help address this restriction at the HDF5 level, and could be implemented as the EFF I/O stack moves from prototype to production.

Use or disclosure of data contained on this sheet is subject to the restriction on the title page of this document.
Copyright © The HDF Group, 2014. All rights reserved

At the HDF5 level, we treat the Tagged TID returned by IOD as a Replica ID, and do not modify the rcntxt_id parameter that is passed in the prefetch call. The keeps the notions of "container version" and "copy of data" separate.

The HDF5 APIs to prefetch H5Datasets, H5Groups, and H5Maps are outlined in the following sections. There are also prefetch APIs for H5NamedDatatypes and H5Attributes. Support for prefetch of H5Attributes will likely not be done in the prototype phase of the EFF project, as the data size is typically small, and any prefetched Attribute can't easily be linked back to the H5Object the Attribute is associated with.

8.2.1 H5Dataset prefetch

herr_t **H5Dprefetch_ff** (*hid_t* dset_id, *hid_t* rcntxt_id, *hrep_t* *replica_id, *hid_t* dapl_id, *hid_t* es_id)

Prefetch an H5Dataset or subset of an H5Dataset at the container version associated with the rcntxt_id.

Only the primary IOD Array object associated with the H5Dataset will be prefetched; auxiliary IOD objects (including Blobs that hold VL data) will not be.

`replica_id` is set to indicate where the pre-fetched Array data can be found and is passed to subsequent read and evict calls.

`dapl_id` is a data access property list identifier. It will be used to specify subsets (hyperslab selection) and layout of the fetched data on the BBs.

Q7: Prefetch full objects with default layout
Q8: Prefetch subsets; ability to control layout, at the minimum, (1) the default layout and (2) "put it all on 'my' BB"

8.2.1.1 Possible Modifications for Production Version

Features that could be valuable, but that may not (or will not) be possible in the prototype phase of the project:

Allow the user to specify the replica_id as an input parameter to the call rather than having IOD assign the ID – this would allow the user to prefetch data for multiple objects as a "batch" and later to evict the data together.

dapl_id is a data access property list identifier.
Properties to allow:

1. Prefetch attributes along with dataset. [Default would be attributes aren't prefetched]
2. For VL datatype, don't prefetch the IOD Blob object(s) that hold the VL data. [Default would be to prefetch the blobs]
3. Indicate the metadata associated with the datatype (datatype indicator, byte order, dimensions, etc.), should not be prefetched [Default would be to prefetch it]
4. Choose "first-index-order" or "last-index-order" organization on BB.
5. Place data on the ION that the CN issuing the prefetch is connected to. [planned in Q8]
6. Round-robin layout of the data with a given stripe per ION, with stripe size expressed in terms of # of cells [may be possible in Q8]

Use or disclosure of data contained on this sheet is subject to the restriction on the title page of this document.
Copyright © The HDF Group, 2014. All rights reserved

8.2.2 H5Group prefetch

herr_t **`H5Gprefetch_ff`** `(`*hid_t* `group_id`, *hid_t* `rcntxt_id,` *hrep_t* `*replica_id,` *hid_t* `gapl_id,` *hid_t* `es_id )`

Prefetch an H5Group at the container version associated with the rcntxt_id.

Only the Primary IOD KV object associated with the H5Group will be prefetched; Auxiliary IOD objects will not be.

`replica_id` is set to indicate where the pre-fetched KV data can be found and is passed to subsequent evict calls. [Note: Since H5Group data is most frequently used in path traversal, and since there is no way to indicate which replica to read when traversing a path, the prefetched H5Group data will not be accessed in the prototype phase of the project.]

`gapl_id` is a group access property list identifier. It will not be used in the prototype phase of the EFF project.

Q7: Prefetch full objects with default layout.
Q8: No additional functionality.

8.2.2.1 Possible Modifications for Production Version

Features that could be valuable, but that will not be possible in the prototype phase of the project:

Allow the user to specify the replica_id as an input parameter to the call rather than having IOD assign the ID – this would allow the user to prefetch data for multiple objects as a "batch" and later to evict the data together.

gapl_id is a group access property list identifier.

Properties to allow:

1. Prefetch attributes along with group.
2. Specify one sub-range of keys (link names) to fetch.
3. Specify multiple sub-ranges of keys (link names) to fetch.
4. Place all the fetched data on the ION that the CN issuing the prefetch is connected to.
5. Specify placement of individual sub-ranges of keys (link names).
6. Prefetch the H5Group and all its members and all their attributes.
7. Prefetch the H5Group and all its descendants.

8.2.3 H5Map prefetch

herr_t **`H5Mprefetch_ff`** `(`*hid_t* `map_id`, *hid_t* `rcntxt_id,` *hrep_t* `*replica_id,` *hid_t* `mapl_id,` *hid_t* `es_id )`

Prefetch an H5Map or subset of an H5Map at the container version associated with the rcntxt_id.

Only the Primary IOD KV object associated with the H5Map will be prefetched; Auxiliary IOD objects will not be.

Use or disclosure of data contained on this sheet is subject to the restriction on the title page of this document.
Copyright © The HDF Group, 2014. All rights reserved

`replica_id` is set to indicate where the pre-fetched KV data can be found and is passed to subsequent read and evict calls.

`mapl_id` is a data access property list identifier. It will be used to specify subsets (key ranges) and layout of the fetched data on the BBs.

Q7: Prefetch full objects with default layout
Q8: Prefetch subset of keys; ability to control layout, at the minimum, (1) the default layout and (2) "put it all on 'my' BB"

8.2.3.1 Possible Modifications for Production Version

Features that could be valuable, but that may not or will not be possible in the prototype phase of the project:

Allow the user to specify the replica_id as an input parameter to the call rather than having IOD assign the ID – this would allow the user to prefetch data for multiple objects as a "batch" and later to evict the data together.

mapl_id is a map access property list identifier.

Properties to allow:
1. Prefetch attributes along with map
2. Specify multiple sub-ranges of keys to fetch
3. Place all the fetched data on the ION that the CN issuing the prefetch is connected to. [planned in Q8]
4. Specify placement of individual sub-ranges of keys.

8.2.4 Prefetch Container Version – Possible Addition for Production Version

There is no easy way to prefetch all H5Objects in a container at a given container version in the prototype EFF HDF5 APIs. This capability could be useful when reading all data in a container that has just been opened for read when all the data fits into the BB. For example, when restarting a simulation, especially if the data could be prefetched by a job scheduler.

HDF5 would need additional support from IOD to offer such a capability.

9 Reading Data: HDF5

The HDF5 APIs *H5Dread_FF* reads H5Dataset object data. Please refer to the *User's Guide to FastForward Features in HDF5* document, updated quarterly, for details on this API and others that perform read operations on the complete set of HDF5 objects with the EFF read context semantics.

All HDF5 APIs that perform read operations take an object ID, an RC ID, and other parameters conveying the details of the read. The HDF5 calls made by an HDF5 user are translated by the HDF5-IOD VOL into IOD calls (see Section 4.1) on the related IOD objects.

The H5Dread_ff API signature is shown as an example:

Use or disclosure of data contained on this sheet is subject to the restriction on the title page of this document.
Copyright © The HDF Group, 2014. All rights reserved

herr_t **`H5Dread_ff`**(*hid_t* `dset_id`, *hid_t* `mem_type_id`, *hid_t* `mem_space_id`, *hid_t* `file_space_id`, *hid_t* `dxpl_id`, *void* * `buf`, *hid_t* `rcntxt_id`, *hid_t* `es_id`)

For the prototype EFF deliverables, in order to read from a replica in the BB, the user will have to pass the `replica_id` that was returned in the `H5Dprefetch_ff` call as one of the properties in the data transfer property list, `dxpl_id`. If all of the requested data is not in that replica, IOD will read the remaining data from DAOS.

If no replica is specified in the property list, IOD will look in the BB for the requested data in the log-format transaction updates for the specified CV, and any data that is not found there will be read from DAOS.

Similar behavior will occur when reading other H5Object types.

9.1 Possible Modifications for Production Version

If IOD could find the "closest data" for a read request as discussed in Section 5.3, the user would never have to specify a replica id in order to access data that had been prefetched into the BB. While replica IDs could still be set if desired, they would not be required.

The management of multiple prefetched IOD objects for a single HDF5 object will also need attention. For example, if the cell values are VL, there will be an IOD Object for every cell in the H5Dataset. Those need to be prefetched and "findable" as well as the Array cell values that hold the Blob object IDs. If IOD can 'find the best' without having to have the replica ID specified, both the necessary Blobs and Arrays should be found in the BB when requested by the HDF5-IOD VOL, on behalf of the HDF5 user.

10 Removing Data from the Burst Buffer: HDF5

The HDF5 APIs layer on top of IOD's *iod_obj_purge* to evict data from the BB. At the HDF5 level, there are H5Xevict_ff calls defined for each of the HD5Objects (H5Group, H5Dataset, H5Map, H5NamedDatatype, and H5Attribute), where the "X" would be "G", "D", "M", "D", or "A", depending on the type of object being evicted.

The data being evicted from the BB will be resident because of either (1) updates to a transaction or (2) a prefetch. At the HDF5 level, when evicting prefetched data, the user must pass the `replica_id` returned by the prefetch call in order to evict the replica. If no `replica_id` is passed, the non-tagged TID will be passed on to the IOD purge call and the logged updates will be evicted.

The HDF5 APIs to evict H5Datasets, H5Groups, and H5Maps are outlined in the following sections. There will also be evict calls for H5NamedDatatypes and H5Attributes, but those are not shown.

10.1 H5Dataset evict

herr_t **`H5Devict_ff`** (*hid_t* `dset_id`, *uint64_t* `container_version`, *hid_t* `dapl_id`, *hid_t* `es_id`)

Evict data associated with an H5Dataset from the BB.

dapl_id is an dataset access property list identifier.

Use or disclosure of data contained on this sheet is subject to the restriction on the title page of this document.
Copyright © The HDF Group, 2014. All rights reserved

- If there is a `replica_id` property in the property list, the Primary IOD Array Object (or sub-object) in the indicated replica will be evicted.
- If there is no `replica_id` property in the property list, the logged updates for the given CV and all lower CVs for the Primary and Auxiliary IOD objects associated with the H5Dataset object will be evicted, according to the IOD eviction rules for non-tagged TIDs.

Q7: Full functionality as described.

10.2 H5Group evict

herr_t **`H5Gevict_ff`**`(` *hid_t* `group_id`, *uint64_t* `container_version`, *hid_t* `gapl_id`, *hid_t* `es_id` `)`

Evict data associated with an H5Group from the BB.

gapl_id is a group access property list identifier.

- If there is a `replica_id` property in the property list, the Primary IOD KV Object (or sub-object) in the indicated replica will be evicted.
- If there is no `replica_id` property in the property list, the logged updates for the given CV and all lower CVs for the Primary and Auxiliary IOD objects associated with the H5Group object will be evicted, according to the IOD eviction rules for non-tagged TIDs.

Q7: Full functionality as described.

10.3 H5Map evict

herr_t **`H5Mevict_ff`**`(` *hid_t* `map_id`, *uint64_t* `container_version`, *hid_t* `mapl_id`, *hid_t* `es_id` `)`

Evict data associated with an H5Map from the BB.

mapl_id is a map access property list identifier.

- If there is a `replica_id` property in the property list, the Primary IOD KV Object (or sub-object) in the indicated replica will be evicted.
- If there is no `replica_id` property in the property list, the logged updates for the given CV and all lower CVs for the Primary and Auxiliary IOD objects associated with the H5Map object will be evicted, according to the IOD eviction rules for non-tagged TIDs.

Q7: Full functionality as described.

10.4 Possible Additions for Production Version

As more functionality is added to the prefetch capabilities, the evict capabilities will need to be expanded as well. For example, if the Auxiliary IOD Objects are prefetched with the Primary IOD Object (presumably sharing the same replica_id), the Auxiliary Objects would need to be evicted – or their eviction controlled by property settings.

For H5Groups, support eviction of a Group and its members or a Group and its descendants.

In general, provide APIs that allow the user to evict data from the BB at a 'larger' granularity than the current (object or replica / container version). For example, a new evict call might take the triple parameters (object ID, container version, replica ID), and allow wildcards for some of

Use or disclosure of data contained on this sheet is subject to the restriction on the title page of this document.
Copyright © The HDF Group, 2014. All rights reserved

the parameters. Or, “match properties” for CV (LE, EXACT, GT, ALL, NE, …). This may make more sense for replicas than for logged updates – at least for some of the properties.

Evict “batch” of data that was prefetched together and has same replica ID.

Support the ability to pin some data in BB and evict the unpinned data.

Use or disclosure of data contained on this sheet is subject to the restriction on the title page of this document.
Copyright © The HDF Group, 2014. All rights reserved