Date: 17 March 2017	# Legion Runtime Support (Milestone 7.1)

NOTICES

Acknowledgment: This material is based upon work supported by Lawrence Livermore National Laboratory subcontract B613306.

Legion Runtime Support (Milestone 7.1)

Scot Breitenfeld and Neil Fortner

{brtnfld, nfortne2}@hdfgroup.org

I. HDF5-FF Support in Legion

Legion is a high-level data-centric and task-based application runtime. Legion's primary data model is based on *Logical Regions* which are the cross product of an *N*-dimensional index space and a multi-variable field space. Logical regions are distinct from the physical regions (memories) that underlie the logical region and provide the physical instantiation of the data. The data model also provides a method of coloring the index space and/or the field space, which can be used to partition an index space or slice the field space of the logical region. The runtime can then use this coloring and a partitioning applied to the logical region to manage a distributed instance of the logical region.

II. Previous Work from Milestone 2.2

The Legion runtime was the first application ported to the FastForward storage stack that uses internal threading as part of its task-based execution model and, consequently, exposed concurrency problems with the HDF5 threadsafe implementation on IOD and DAOS. For Milestone 2.2, the HDF Group successfully demonstrated the Legion example application running on a single client task and a single server process. However, attempts to scale to higher task counts consistently crashed the application. An in-depth analysis of the previous IOD and DAOS issues with Legion and the subsequent improvements is detailed in [1].

III. Demonstration Objectives and Example Application

The newest version of DAOS removes the restriction of not allowing reading from a read context when any lower transactions have yet to be committed. Therefore, the bulk synchronous nature capabilities of DAOS, i.e. no bulk synchronization blocking, using the transaction and epoch model of DAOS will be shown in the remaining sections. A typical workload generator program, an existing proxy-io program, is used to simulate this behavior. Figure 1 shows the typical workload for the read/write phase for different shards in a proxy-io Legion program [2]. Additionally, Figure 1 highlights Legion's capability of scheduling different phases of tasks based on explicit dependencies and, consequently, allows for reading tasks to run concurrently with writing tasks on the same logical region [2]. This milestone demonstrates this ability to perform independent I/O phases (write and read) associated with each shard of global data for DAOS.

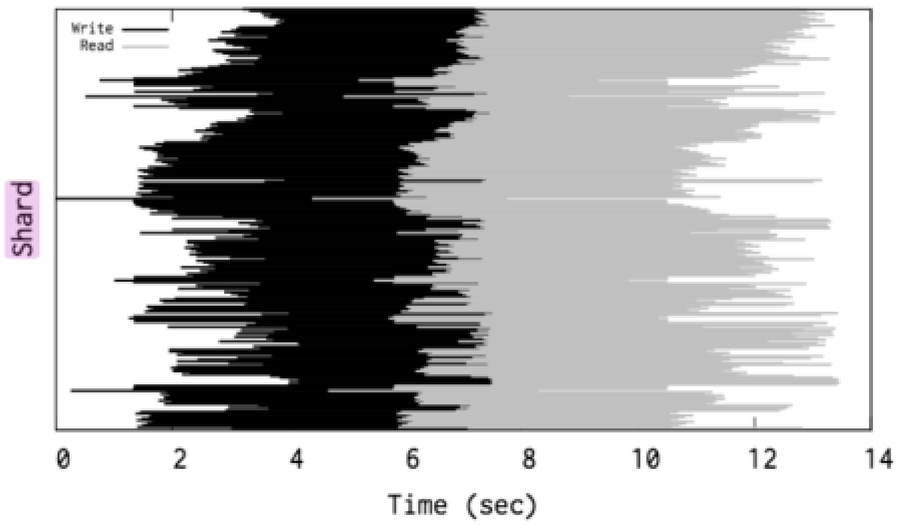

Figure 1: Time-series of the I/O phases (write and read) associated with each shard's global data structure being persisted [2].

As part of demonstrating Legion's use of HDF5 on the DAOS storage stack, an example application was created that demonstrates the capabilities the storage stack. The application, *tester_io*, is part of Legion's Github repository [3] and can be found in the source tree at *test/hdf_attach_subregion_parallel*. *Tester_io* is a straightforward tour of the HDF5-DAOS features from a simple Legion code and is designed to run the Legion runtime on multiple compute nodes, talking to multiple I/O and storage server nodes.

IV. Legion Implementation for current DAOS

Milestone 2.2 implementation of Legion, using IOD and DAOS, required transactions and event stacks to be handled by the application. These new parameters needed to be passed into the HDF5 APIs, so new Fastforward versions of the HDF5 APIs were created which had a *'_ff'* appended to the original HDF5 API names. Therefore, in addition to the added complexity of handling transactions and the event stack within Legion, all the HDF5 APIs utilizing IOD/DAOS had to be replaced with the Fastforward HDF5 equivalent APIs.

However, in the current implementation of HDF5 on top of DAOS, transactions and the event stack are handled internally in the HDF5 library, and consequently, the original HDF5 API could be used as no extra parameters are needed. Therefore, the number of changes from the original Legion and HDF5 implementation was minimal. Two new HDF5 APIs were introduced into Legion:

> *H5VLdaosm_init* – Initialize the ***VOL*** plugin by connecting to the pool and registering the driver with the HDF5 library.
> *H5VLdaosm_term* – Shut down the DAOS ***VOL***.

These functions must be called only once by all the processes within Legion. Additionally, a process barrier was need in the *tester_io* before calling *H5VLdaosm_term*() to avoid DAOS shutdown before all the Legion tasks had completed their I/O.

Furthermore, Virtual dataset support is a new feature added to HDF5. A virtual dataset (VDS) does not store any data directly, rather, it stores data in a set of source datasets. Source datasets are created like any other source dataset, and the virtual dataset is created with a set of mappings between regions in the virtual dataset and regions in the source dataset. I/O requests to the VDS are then translated into a set of I/O requests to the source datasets whose mappings overlap with the requested region in the VDS.

Virtual datasets mapped closely to Legion's data model and were added to Legion (Milestone 3.1) to replace the use of H5Lcreate_external, which was used to create external links in a master HDF5 file, linking to the shard files. However, H5Lcreate_external is not, and will not, be supported in the current HDF5 DAOS implementation. Since the current HDF5 DAOS implementation does not include the VDS APIs, this functionality of creating a master HDF5 file linking the shard files via VDS was removed in the current Legion implementation. The functionality can easily be restored once the HDF5 VDS APIs are ported to DAOS.

V. Example Legion HDF5/DAOS I/O application

The example relies on the low-level networking layer gasnet [4] for network-independent, high-performance communication primitives. The mpirun command in the gasnet wrapper script *gasnetrun_ibv* needed to be updated to include DAOS specific parameters, in red,

```
mpirun -hostfile $HOME/.host_file -x DD_MASK="all" --ompi-server file:~/uri.txt -np %N %C }.
```

The HDF5 I/O example verifies the reading of the written values by setting -DTESTRIO_CHECK, and the timing of the reading and writing phases is enabled by setting –DTESTERIO_TIMERS. The pool id is passed to Legion through the environment variable ***pid***. The pool id is obtained by first starting the DAOS server,

```
orterun -np 1 --report-uri ~/uri.txt /scratch/DAOS_INSTALL/bin/daos_server -c 1
```

and issuing the command to create the pool,

```
export pid=$(orterun -np 1 --ompi-server file:~/uri.txt /scratch/DAOS_INSTALL/bin/dmg create)
```

The command for running *tester_io* on 150 nodes with 131072 elements and 256 shards is,

```
GASNET_BACKTRACE=1 GASNET_USE_XRC=0  GASNET_MASTERIP='192.168.1.117'
GASNET_SPAWN=-L $HOME/packages/gasnet/bin/gasnetrun_ibv -n 150 tester_io -n 131072 -s 256
```

which on boro (Intel's cluster) will run on three nodes. The independent I/O phases (write and read) for each shard for this example are plotted in Fig 2.

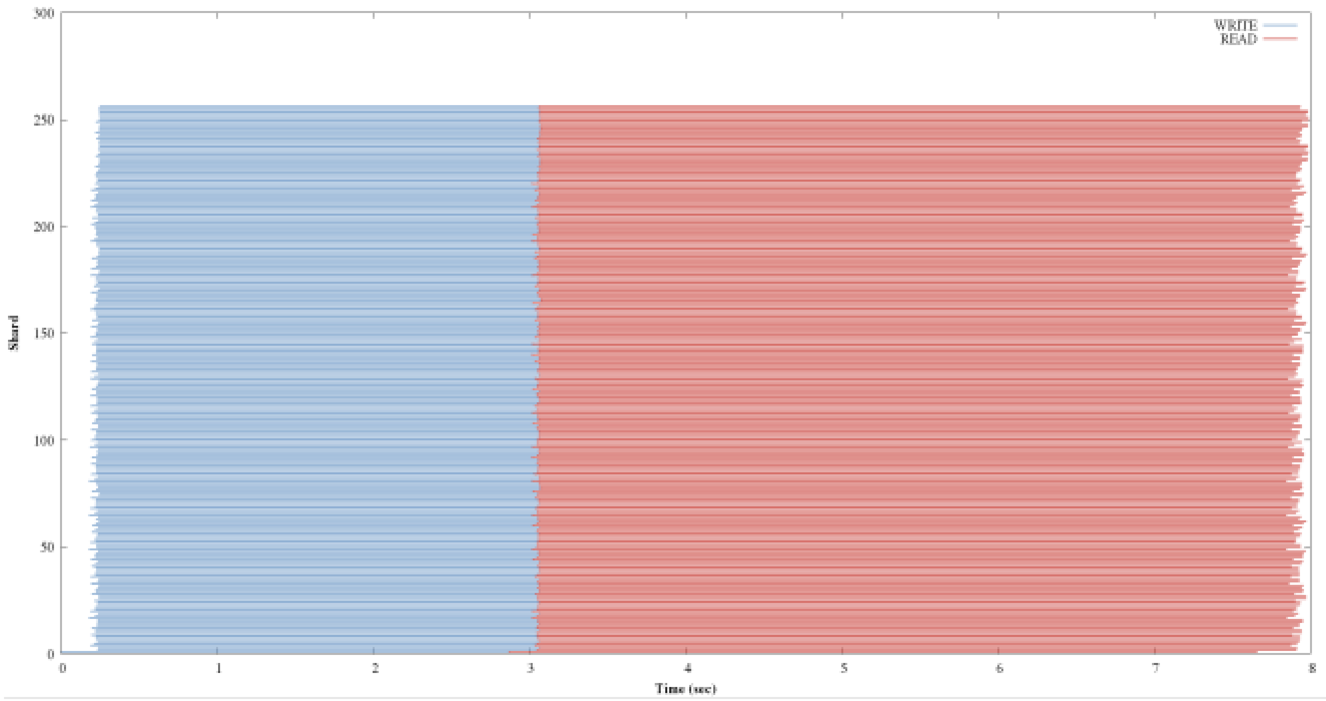

Figure 2: Time-series of the I/O phases (write and read) associated with each shard's global data structure being persisted for HDF5 with DAOS.

The individual tasks in Legion are schedule independently and the completion of the task can vary since Legion can schedule different phases of I/O based on data dependency.

There is a current issue with simulating larger problems, for example doubling the previous simulation's number of elements to 252144, which will cause the DAOS server to crash. This issue is addressed in Section VI.

VI. Network interface CCI issues

HDF5 maps individual dataset elements to object KV records. This is an acceptable mapping, but it exposed a current limitation in the current network interface CCI that is used by Mercury to ship data from the clients to the servers. Currently DAOS issues a bulk transfer for each individual record. This works well if we have large records that need to be updated atomically, but if we have small record sizes, as is the case in HDF5 applications that use Datasets with small datatypes (e.g. byte, int, double, etc.), this will result in a large amount of bulk transfers. CCI has a limitation on the number of inflight bulk transfers, where it basically stops progressing when this limitation is reached and issuing more operations from the client would cause a crash at the server.
This will be addressed in the future in two ways. The DAOS implementation for managing small records will change to use rectangle trees where those small records are managed together and the number of bulk transfers and memory allocations will decrease significantly. Furthermore, DAOS will switch away from using CCI and will actually use OFI where elementary testing proved that this limitation is not encountered.

References

[1] *M3.1 Legion on the Exascale FastForward I/O Stack Extreme Scale Storage and I/O RND*
[2] Watkins, N., Jia, Z., Shipman, G., Maltzahn, C., Aiken, A. McCormick, P. *Automatic and Transparent I/O Optimization with Storage Integrated Application Runtime Support*, PDSW 2015, November 15-20, 2015 Austin, TX, USA
[3] https://github.com/StanfordLegion/legion
[4] https://gasnet.lbl.gov/