Architecture and Design of Cray DataWarp

Dave Henseler*, Benjamin Landsteiner*, Doug Petesch*, Cornell Wright†, and Nicholas J. Wright‡
*Cray Inc., Saint Paul, MN 55101
Email: {dah, ben, dpetesch}@cray.com
†Los Alamos National Laboratory, Los Alamos, NM 87544
Email: cornell@lanl.gov
‡Lawrence Berkeley National Laboratory
National Energy Research Scientific Computing Center, Berkeley, CA 94720
Email: njwright@lbl.gov

***Abstract*—This paper describes the architecture, design, use, and performance of Cray DataWarp. DataWarp is an infrastructure that uses direct-attached solid-state disk (SSD) storage to provide more cost-effective bandwidth than an external parallel file system (PFS), allowing DataWarp to be provisioned for bandwidth and the PFS to be provisioned for capacity and resiliency. Placing this storage between the application and the PFS allows application I/O to be decoupled from PFS I/O and in some cases avoiding it altogether. This reduces the time required for the application to do I/O, while also increasing the overlap of computation with PFS I/O, typically reducing application elapsed time. DataWarp allocates and configures SSD-backed storage for jobs and users on demand, providing many of the benefits of both software defined storage and storage virtualization.**

***Keywords*-Filesystems & I/O, DataWarp, SSD**

I. Introduction

Although hard disk drive (HDD) capacity has increased significantly in recent years, HDD bandwidth increases have been much more modest. In contrast, solid-state disk (SSD) devices provide high bandwidth and their prices have been declining. At the present time, SSD bandwidth ($ / GB / second) is less expensive than HDD bandwidth. This leads to SSDs being the economically preferable device for use cases dominated by bandwidth considerations. Placing SSD storage between the application and a HDD-backed Parallel File System (PFS) such as Lustre [1] or GPFS [2] can allow the PFS to be more cost-effective as it only needs to be provisioned for capacity and resiliency rather than peak bandwidth. This also reduces the cost of connectivity from the supercomputer to the PFS. Placing this storage between the application and the PFS allows application I/O to be decoupled from PFS I/O and in some cases avoiding it altogether. This reduces the time required for the application to do I/O, increases the overlap of computation with PFS I/O, and typically reduces application elapsed time.

Cray DataWarp software allocates and configures SSD-backed storage for jobs and users, providing many of the benefits of both software-defined storage [3] and storage virtualization. This includes automation, standard APIs, virtualized data path, scalability, and transparency. DataWarp is an implementation of the burst buffer concept [4] for Cray systems, and more.

Automated tasks include on-demand allocation, configuration, and migration of data to and from a PFS. Users request use of DataWarp primarily through simple job script directives and then use DataWarp through POSIX APIs. If desired, a C library provides an additional way for applications to make use of DataWarp. Command line tools expose functionality in a RESTful interface that is capable of showing status and requesting a particular configuration. DataWarp configures compute nodes to interact with their assigned storage and provides for per-instance metadata. Striping across multiple server nodes allows DataWarp to have access to additional capacity and bandwidth. Use of DataWarp is optional and job script directives specify the degree to which data is intended to be exposed and shared to other jobs. Workload Managers (WLMs) manage the lifetime of storage allocations to jobs and ensure that requested capacity remains dedicated to jobs for as long as is needed.

PCI-attached SSDs are located on Cray XC service nodes connected directly to the Aries high speed network. The SSD bandwidth is closely matched to the Aries bandwidth, taking maximum advantage of the Aries network and avoiding the additional cost of provisioning connectivity and bandwidth to external SSDs.

DataWarp will be released in four phases. Each phase introduces new features and refines or replaces the functionality present in prior phases.

A. Phase 0

Phase 0 supports statically configured compute node swap and single server scratch file systems. Cray released Phase 0 in CLE5.2.UP02 in Fall 2014.

B. Phase 1

Phase 1 supports dynamic allocation and configuration of DataWarp storage to jobs. This includes requests for swap, and job- and application-controlled explicit data movement between DataWarp and PFS storage. An instance of storage that supports explicitly moving data is described hereafter

as a *scratch instance*. Applications interact with scratch instances through mount points and explicitly request data to be copied between the scratch instance and PFS via job script-accessible and application-accessible APIs. Scratch instances also support policies intended to detect and halt excess I/O activity that would otherwise lead to premature SSD failure. Phase 1 supersedes Phase 0, though DataWarp server nodes can still be used with the Phase 0 infrastructure by isolating them from the Phase 1 configuration. Cray released Phase 1 in CLE5.2.UP04 in Fall 2015.

C. Phase 2

Phase 2 adds implicit movement of data between DataWarp and PFS storage, described hereafter as a *cached instance*. It also includes accounting support and Data Virtualization Service (DVS) [5] client-side caching. With a cached instance, job scripts and applications are only required to switch from using a PFS mount point to using a DataWarp mount point. When using the DataWarp mount point, I/O between the application and PFS is transparently buffered on the DataWarp server nodes. Jobs and applications may additionally interact with the cached instance through other APIs to further improve performance. Cached instances support excess I/O activity detection as in Phase 1. The accounting functionality works with scratch and cached instances and measures I/O interactions between compute nodes and DataWarp nodes. DVS client-side caching improves performance for workloads with small reads and writes. Cray plans to release Phase 2 in Summer 2016.

D. Phase 3

Phase 3 adds the ability to run applications directly on DataWarp server nodes. It also includes improved resiliency and recovery support, and allows jobs to request multiple job instances per job. Cray plans to release Phase 3 in Winter 2016.

II. Use Cases

The following is a list of anticipated use cases that will leverage the functionality of DataWarp.

A. Parallel File System Cache

DataWarp can be used to cache PFS data. Unlike SSD caching infrastructures that provide a block-based cache between a file system and a hard disk, DataWarp provides a file-based cache between the application and a PFS. The DataWarp server software is fully integrated with both the kernel page cache and the PFS client.

Some examples of PFS cache use cases are:

- **Checkpoint/Restart:** In order to tolerate both compute node and system failures, applications can periodically write checkpoint files. All checkpoints are first written to DataWarp to take advantage of the high bandwidth. Some of those checkpoints reside only on DataWarp and support fast restart in the event of a compute node failure. Other checkpoints will be asynchronously copied by DataWarp to the PFS to support restart in the event of a system failure. Job script directives and C library functions allow users to specify the policy behind which files are copied to the PFS and when.
- **Periodic output:** Applications that produce periodic output (for example time series data) can write those results to DataWarp and then resume computation while DataWarp copies that data to the PFS asynchronously.
- **Application libraries:** Some applications reference a large number of libraries from every rank, for example Python-based applications. Those libraries can be copied from the PFS to DataWarp once and used by all ranks directly from DataWarp storage.

When using a DataWarp scratch instance, movement of data between DataWarp and the PFS is explicitly requested by the job and/or the application and is performed by the DataWarp service. When using a DataWarp cached instance, movement of data between DataWarp and the PFS can also be done implicitly (read ahead and write behind) by the DataWarp service without any intervention from the application.

B. Application Scratch

DataWarp can provide storage that functions like a big `/tmp` file system for each compute node in a job. Applications that use "out-of-core" algorithms could use DataWarp storage this way. Typically this data never reaches the PFS, but it can if desired by using job script directives or C library functions.

C. Shared Data

DataWarp can provide storage for data that is shared by multiple jobs over long periods of time. The jobs may be related or not, and may run concurrently or serially. Use cases include:

- **Shared input:** This would usually be a read-only input file or database, for example a bioinformatics database used as input by multiple analysis jobs. The data can be copied from PFS to DataWarp once and then shared by many jobs.
- **Ensemble analysis:** This is often a special case of the shared input case for a set of similar runs with different parameters on the same inputs, but could also allow for some minor modification of the input data across the runs in a set. Many simulation strategies make use of ensembles.
- **In-transit analysis:** This refers to passing the results of one job to the input of a subsequent job, typically using WLM job dependencies. The data can reside only on DataWarp storage and never touch the PFS. This includes various types of workflows that go through a sequence of processing steps, transforming the input data along the way for each step. This can also be

used for processing of intermediate results while an application is running, for example visualization or analysis of partial results, potentially even for steering the main application.

D. Compute Node Swap

DataWarp can provide compute node swap for jobs that require it. This is often needed by pre- and post-processing jobs that cannot fit in compute node memory and therefore need to swap during execution. If only unused system libraries and services are swapped out (i.e., the application only needs a small amount of additional compute node memory) the performance penalty to the application is limited to the initial swap out and application performance may improve.

As with any swap implementation, swap performance is only acceptable for limited or transient overcommitments of node memory. For swap to SSD, excessive swapping can reduce the SSD endurance disproportionate to other uses.

E. Applications Running On DataWarp Server Nodes

DataWarp can allow applications to run directly on DataWarp server nodes, giving the applications direct access to the local performance of the SSD storage. Specifically, the high input/output operations per second (IOPS) rate for small transfers, file creation, and file deletion. This will place specific ranks/PEs on the DataWarp server nodes running in the same MPI_COMM_WORLD communicator as ranks on compute nodes.

The DataWarp server nodes used to host applications are dedicated for this purpose only. They are treated as compute nodes, requested explicitly by a job and allocated exclusively to a job by the WLM. Administrators can manually move DataWarp server nodes between this use and normal DataWarp use.

III. Architecture, Design, and Implementation

This section describes the architecture, design, and implementation of DataWarp. See Figure 1 for a pictorial representation of the hardware architecture. Compute nodes (CN) access DataWarp (DW) over the high-speed Aries network. DataWarp nodes access PFS resources, such as Lustre, via a path that traverses both the Aries network and non-Aries network, such as Infiniband.

Figure 2 shows the software architecture component diagram. WLMs work with DataWarp to ensure compute node applications have access to DataWarp resources. Files can be staged between DataWarp and a PFS via job script requests or application requests routed to DataWarp through DVS. Application I/O is routed to DataWarp servers through DVS. Application data is temporarily stored on DataWarp server SSDs in order to accelerate application I/O operations.

A. Instances

When requested, DataWarp storage space is assigned dynamically and in the amount requested. This is referred to as an *instance*. The space is allocated on one or more DataWarp server nodes and is dedicated to the instance for its lifetime. The space taken on each node for the instance is referred to as a *fragment*. The requested capacity is used as a job scheduling constraint by the WLM.

The bandwidth associated with an instance is dedicated to it if there is only one instance present on the assigned DataWarp server nodes. For example, if two jobs each request half the total DataWarp capacity and each job gets exclusive access to all the capacity on half the DataWarp server nodes, then each job gets half the total DataWarp bandwidth. Conversely, if each job gets half the capacity on all DataWarp server nodes, each job gets all of the total DataWarp bandwidth when their I/O does not overlap. In the first example, performance is more deterministic, but depending on the degree of I/O overlap, the second example application's I/O may perform up to twice as fast.

A DataWarp instance has a lifetime that is specified when it is created and can be one of:

- **Job instance:** A *job instance* lifetime is the same as the lifetime of the job that created it (created at job start, destroyed at job end), and is accessible only by the compute nodes allocated to the job that created it. A job instance is relevant to all use cases except shared data.
- **Persistent instance:** A *persistent instance* lifetime is not tied to the lifetime of any single job. Access can be requested by any job, but file access is authenticated and authorized based on the POSIX file permissions of the individual files and directories. Jobs request access to a persistent instance by providing the persistent instance's name in a job script directive. A persistent instance is relevant to the shared data use case. Persistent instances are explicitly created by a user, typically outside of a job, since their lifetime is not associated with any job's lifetime.

When an instance is destroyed, DataWarp will ensure that any data that needs to be written to the PFS has been before it releases the space for reuse. In the case of a job instance, this can delay the completion of the job.

B. Application I/O

The DataWarp service dynamically configures access to DataWarp instances on all compute nodes assigned to a job using a given instance. Application I/O is forwarded from compute nodes to the instance's DataWarp server nodes using the Data Virtualization Service (DVS). DVS provides POSIX-based file system access to the DataWarp storage.

A DataWarp instance can be configured as one of three types:

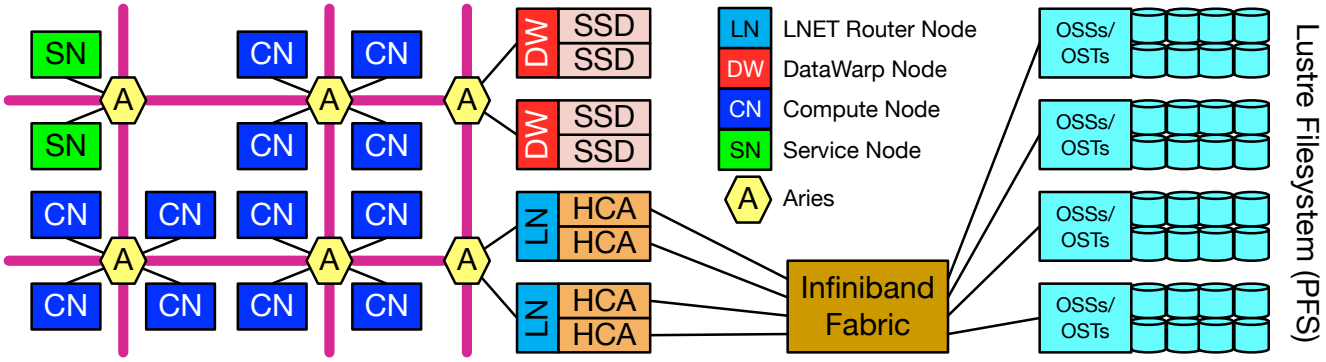


Figure 1. DataWarp hardware architecture overview.

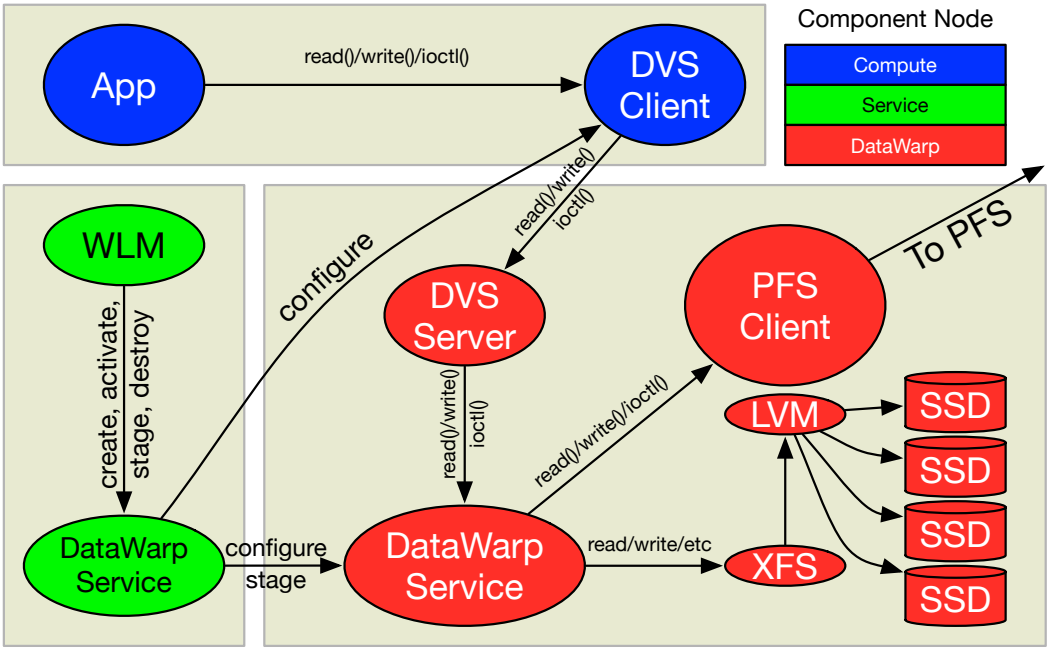


Figure 2. DataWarp software architecture component diagram.

- **Scratch:** When using a scratch instance (i.e., DataWarp Phase 1) all movement of data between the instance and the PFS must be explicitly requested using the DataWarp API. The actual data transfers are performed directly between the DataWarp nodes and the PFS.
- **Cached:** When using a cached instance (i.e., DataWarp Phase 2) all movement of data between the instance and the PFS is done implicitly. The instance configuration and application-accessible C API can change the implicit behavior.
- **Swap:** When using a swap instance, swap files are created on a DataWarp instance that compute nodes access over iSCSI.

An instance can be scratch, cached, swap, scratch+swap, or cached+swap, but never scratch+cached. A single instance used for scratch+cached would lead to scratch usages of the instance running out of space, as the cached usage would fill up the instance in order to maximize cache hits. Phase 3 will allow for jobs to request multiple job instances which will allow for a job to request both a scratch and a cached instance.

A scratch or cached instance can be accessed in one or more of the following ways:

- **Striped:** In striped access mode, individual files are striped across multiple DataWarp server nodes (aggregating both capacity and bandwidth per file) and are accessible by all compute nodes using the instance.
- **Private:** In private access mode, individual files are striped across multiple DataWarp server nodes (aggregating both capacity and bandwidth per file) but the files

are only accessible to the compute node that created them (e.g. like `/tmp`). Private access is not supported for persistent instances because a persistent instance can be used by multiple jobs with different numbers of compute nodes; it is not supported for cached instances because all files are visible in the PFS.
- **Load-balanced:** In load-balanced access mode, individual files are replicated (read-only) on multiple DataWarp server nodes (aggregating bandwidth but not capacity per instance), and compute nodes choose one of the replicas to use. Load-balanced mode is useful when the files are not large enough to stripe across a sufficient number of nodes.

Each permutation of instance (job or persistent), type (scratch or cached), and access mode (striped, private, or load-balanced) maps to unique mount points on the compute nodes. The mount point for each is provided to the job via environment variables. See Table I for the valid combinations of access mode, configuration, and instance type.

Table I
ACCESS MODE SUPPORT FOR SCRATCH AND CACHED CONFIGURATIONS

	Job Instance		Persistent Instance	
access mode	scratch	cached	scratch	cached
striped	yes	yes	yes	yes
private	yes	no	no	no
load-balanced	yes	yes	yes	yes

Within a scratch instance, the directory structure (and associated directory overhead) for the instance is managed differently depending on the access mode: for striped access, one of the DataWarp servers manages the directory for the instance; for private and load-balanced access, each DataWarp server manages the directories for files assigned to it. Within a cached instance, the directory structure resides on the PFS itself, and every DataWarp server can access it directly.

Every DataWarp server performs PFS I/O for the files and stripes that reside on it. This provides parallel access to the PFS, but from fewer nodes than if it had been directly accessed from compute nodes. For a scratch instance, PFS I/O is explicitly initiated by the job or application, and it transfers an entire file to or from the DataWarp server nodes. For a cached instance, PFS I/O is implicitly initiated by the DataWarp service based on modified data thresholds, the DataWarp file system becoming full, or reads of data not currently present. The DataWarp service manages space using least-recently-used tracking and provides read-ahead and write-behind functionality. As a result, the file system never becomes full since it is always able to flush modified data to, or to reclaim data from, the PFS.

C. Workload Manager Integration

Multiple workload managers (WLMs), including Moab/TORQUE [6], [7], PBS [8], and Slurm [9], support DataWarp on Cray systems. DataWarp directives are specified in the job script file and passed to the DataWarp infrastructure by the WLM at various points in a job lifetime. These callouts allow DataWarp to:

- Create/configure instances on DataWarp server nodes
- Stage files between a PFS and DataWarp instances (without compute nodes assigned if the WLM supports it), at both the beginning and end of a job
- Configure/unconfigure compute node access to DataWarp instances
- Unconfigure/clean-up instances on DataWarp server nodes

The DataWarp job script directives include the ability to:

- Create and configure compute node access to a DataWarp job scratch instance:
  ```
  #DW jobdw type=scratch
  access_mode=[striped,private,loadbalance]
  capacity=n ...
  ```
- Create and configure compute node access to a DataWarp job cached instance:
  ```
  #DW jobdw type=cache
  access_mode=[striped,loadbalance] capacity=n
  pfs=path ...
  ```
- Configure compute node access to a DataWarp persistent instance:
  ```
  #DW persistentdw name=xxx
  ```
- Create swap space for each compute node in a job:
  ```
  #DW swap nGiB
  ```
- Stage files from the PFS to DataWarp storage before a job runs:
  ```
  #DW stage_in type=file|directory|list
  source=path destination=path
  ```
- Stage files from DataWarp storage to the PFS after a job completes:
  ```
  #DW stage_out type=file|directory|list
  source=path destination=path
  ```

The `#DW jobdw` directive takes many additional optional parameters that allow users to specify I/O limits and set additional type-specific attributes. For detail refer to the DataWarp User Guide [10].

The capacity requested for a DataWarp instance is a consumable resource, used by the WLM as a scheduling constraint. This means that jobs requesting DataWarp capacity will only start execution when that capacity is available and can be dedicated to the job.

If the WLM supports it, stage-in may be done before compute resources are assigned to the job and stage-out after those compute resources have been released. This enhancement can significantly decrease the amount of time compute nodes are assigned; see Figure 3.

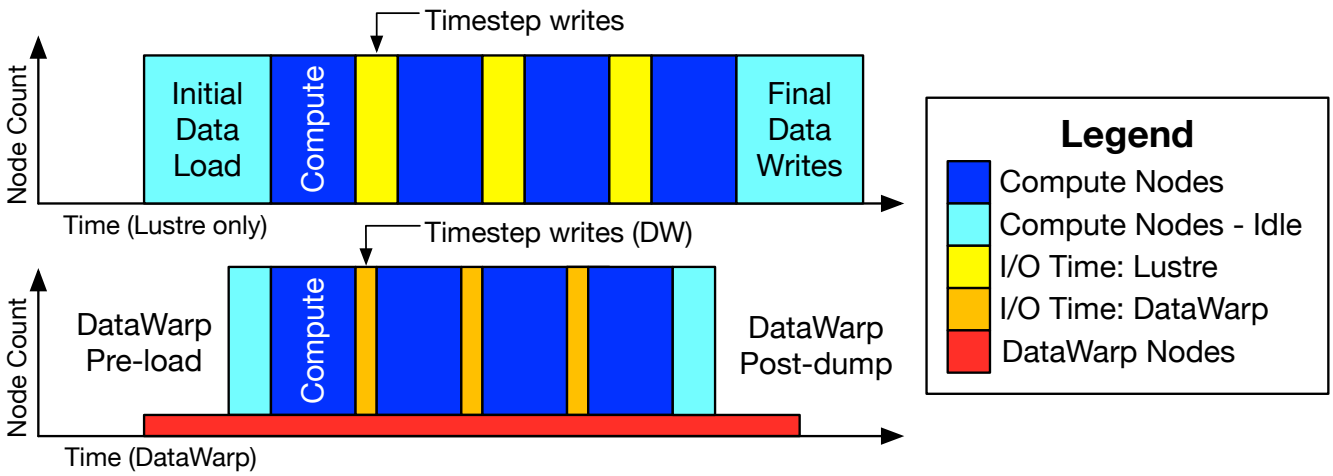


Figure 3. Loading up and draining DataWarp before compute nodes are assigned can reduce the amount of time compute nodes are allocated to a job and idling. I/O performed during an application's lifetime is also accelerated if interacting with DataWarp since DataWarp bandwidth exceeds PFS bandwidth.

```
#!/bin/sh
#DW jobdw type=scratch access_mode=striped,private capacity=100TiB
#DW persistentdw name=rrr
#DW stage_in type=file source=/pfs/user/input destination=$DW_JOB_STRIPED/input
#DW stage_out type=directory source=$DW_JOB_STRIPED/results/ destination=/pfs/user/results/

export TMPDIR=$DW_JOB_PRIVATE
aprun -n 5000 a.out --tmpfiles=$DW_JOB_PRIVATE \
    --database=$DW_PERSISTENT_STRIPED_rrr/abc \
    --parameter-file=$DW_JOB_STRIPED/input \
    --resultsdir=$DW_JOB_STRIPED/results
```

Figure 4. Example job script with DataWarp #DW directives. This batch job requests a 100TiB scratch instance with access modes striped and private. It also requests access to the persistent DataWarp instance named rrr. The user's parameter file input is staged into the job scratch instance before the aprun executes, and the result files are staged back out to the PFS after the aprun completes.

D. Job Example

See Figure 4 for an example of a job that requests a 100TiB capacity DataWarp job instance, references an existing DataWarp persistent instance, requests staging-in of a file, and requests staging-out a directory.

The DataWarp-enabled job file is submitted to the WLM just like any job file, e.g., for Slurm one would use `sbatch job.sh`. Each compute node has striped access to the persistent instance named `rrr` via the path defined by `$DW_PERSISTENT_STRIPED_rrr`, striped access to the job instance via the path specified by `$DW_JOB_STRIPED`, and private access via the path referenced at `$DW_JOB_PRIVATE`.

E. Application Programming Interfaces

DataWarp provides command-line clients to query and control the DataWarp configuration. In addition to the job script directives discussed previously, there are commands to create and destroy persistent instances, list instances, list job references to instances, query server node state, and more. Some commands or their functionality are limited to administrators only. In general, administrative users can see all DataWarp configuration and non-administrators can only see persistent instances and any configuration data associated with their userid.

All command-line clients interact with the DataWarp RESTful API over HTTPS. All requests for DataWarp status, set up, job-directed staging, and teardown go through this API. Users are authenticated with MUNGE [11] and authorized in accordance with site configuration.

DataWarp also provides a C library API for use by applications which includes starting, querying, waiting, and terminating stage activity between a PFS and a DataWarp instance. It also allows for retrieving or setting the stripe configuration of a file. For cached instances, the library additionally supports getting and setting the modified threshold, read-ahead policy, and sync behavior policy. Staging can be asynchronous with respect to an application or job, is performed concurrently from all DataWarp servers assigned to an instance, and can be done immediately or deferred until the end of a job. A user might stage out only the last successful application-directed checkpoint-restart files.

In addition, some POSIX API calls have unique semantics when used with DataWarp:

- **statfs:** will return the aggregate size of the DataWarp instance, with the free space computed as the product of the number of servers and the current free capacity of the server with the least currently free space since that server will likely fill up first.
- **stat:** stat information is only guaranteed to be accurate when a file is not open for write (i.e., it is updated at close).
- **sync:** (only for type=cache) The behavior of a POSIX sync, fsync, or fdatasync call is controlled by the value of the sync_to_pfs attribute for a file, where the default is specified per instance. If true then it will sync data from DataWarp storage to the PFS. If false, a sync operation will just sync modified data cached in the DataWarp server's node memory to the DataWarp instance's storage file system.
- **close:** (only for type=cache) The behavior of a POSIX close is controlled by the value of the sync_on_close for a file, where the default is specified per instance. If true then the close will sync data to the PFS. If false, close will just sync modified data cached in the DataWarp instance's server node memory to the DataWarp instance storage file system.
- **unlink:** (only for type=cache) A POSIX unlink operation will remove the PFS file and remove all DataWarp instance state and data related to the file. If the file is open, the invalidate and removal is deferred until the last close.
- **POSIX permissions:** (for type=cache and staging with type=scratch) The ability to access files (e.g. open, create, unlink, read, write, etc.) is controlled by the PFS permissions on directories and files.

F. Storage Management

A DataWarp server node contains one or more PCI SSD cards. Architecturally, any block storage device could work but only qualified SSDs are supported. The Logical Volume Manager (LVM) software [12] is used to aggregate the SSD cards on a server into a volume group and to partition the volume group into striped logical volumes on demand. When creating an instance, the DataWarp service configures the instance by creating logical volumes from the currently unallocated storage on as many server nodes as are needed to satisfy the requested capacity. Depending on the site configuration and request preference, capacity can be chosen optimizing for bandwidth (use more servers) or minimal interference (use fewer servers). Each piece of an instance on each server is called a *fragment*. See Figure 5 for an example with three instances and varying numbers of fragments spread across up to three servers.

An XFS file system is created on each logical volume to manage the storage space [13]. XFS is a reliable and high performance file system for use with DataWarp. It provides:

- integration with striped LVM volumes
- SSD TRIM
- configurable allocation sizes
- scalable space allocation, file creation, and file deletion
- extended attributes, and
- space management APIs

The DVS stripe size is managed by the DataWarp service, taking into consideration the:

- SSD erasure block size to minimize write amplification
- LVM stripe size to maximize device concurrency
- XFS allocation and stripe unit sizes, and
- PFS stripe size to minimize lock conflicts on shared files

Each DataWarp server node on a system can be configured either for use by the DataWarp infrastructure (described in this paper) or by the customer for other use. An administrator associates each DataWarp server node with a DataWarp pool. Storage is allocated by DataWarp to jobs (if supported by the WLM), and to users from a specific DataWarp pool with a common allocation granularity. Figure 6 shows the status of four DataWarp server nodes belonging to the pool `wlm_pool`, and two DataWarp server nodes belonging to the pool `admin_pool`, from the perspective of `dwstat`, one of the DataWarp command line tools. Figure 7 shows a graphical representation of the association.

There are tradeoffs in picking a pool allocation granularity. Picking a smaller allocation granularity will allow requests for smaller capacity instances to span more DataWarp servers, but then they have to share server bandwidth with other instances. Picking a larger allocation granularity can reduce the degree to which servers are shared but also result in a high compute-to-server ratio and limited bandwidth. Pool allocation granularity equal to the capacity of each server node prevents sharing of servers. Requests are then sized in such a way that the number of servers, and thus bandwidth, can be guaranteed.

The DataWarp service provides instance- and application-based I/O accounting and configurable I/O limits. Accounting data is collected using RUR [14] and includes statistics such as reads/writes performed per compute node and per server node.

To prevent a runaway application from consuming too much of the remaining device endurance, excess I/O activity detection is determined using three different user- and site-configurable policies:

- **Maximum Files Created:** Users that know how many files their application will create can specify an upper bound to DataWarp and have subsequent creates fail.
- **Maximum File Size:** Users that know the maximum file size to ever be read or written by their application can specify an upper bound to have reads and writes in violation of the policy fail.

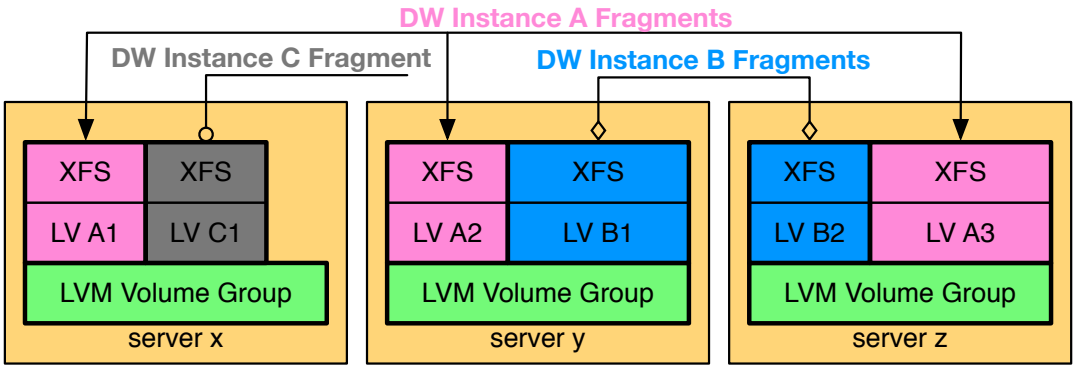


Figure 5. Three DataWarp instances A, B, and C mapped to three servers, x, y, and z. Instance A consists of fragments that map to LVM logical volumes A1, A2, and A3 on servers x, y, and z, respectively. Instance B consists of fragments that map to LVM logical volumes B1 and B2 on servers y and z, respectively. Instance C consists of a single fragment that maps to LVM logical volume C1 on server x.

```
crayadm@login:~> module load dws
crayadm@login:~> dwstat pools nodes
      pool units quantity    free  gran
  wlm_pool bytes    24TiB   24TiB  1TiB
admin_pool bytes  12.8TiB 12.8TiB 16MiB

    node       pool online drain  gran capacity insts activs
nid00022   wlm_pool   true false 16MiB   6.4TiB     0      0
nid00023   wlm_pool   true false 16MiB   6.4TiB     0      0
nid00024   wlm_pool   true false 16MiB   6.4TiB     0      0
nid00025   wlm_pool   true false 16MiB   6.4TiB     0      0
nid00048 admin_pool   true false 16MiB   6.4TiB     0      0
nid00049 admin_pool   true false 16MiB   6.4TiB     0      0
```

Figure 6. Sample dwstat command output for pools and nodes

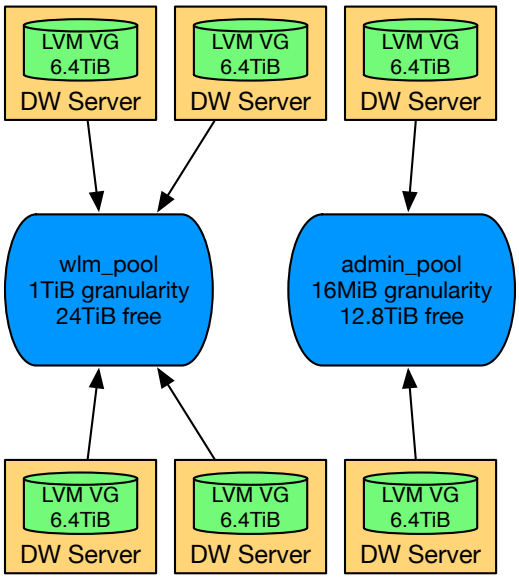


Figure 7. Six server nodes belonging to two DataWarp storage pools. The pool *wlm_pool* has four nodes worth of storage in it, and instance fragments are created that are multiples of 1TiB. The pool *admin_pool* has two nodes worth of storage in it, and instance fragments are created that are multiples of 16MiB. The larger allocation granularity associated with *wlm_pool* means that 0.4TiB of space per server node is inaccessible.

- **Maximum Writes Over Time:** Users that know how much data their application will write over a period of time (e.g., entire batch job, every 2 hours, etc.) can specify a time window and a byte value to establish an upper bound on writes in a rolling window that, when exceeded, will prevent future writes from succeeding. As time progresses, the window slides forward and writes that occurred outside of the window are no longer counted against the byte threshold.

G. Resiliency

If a DataWarp server node fails, only jobs using instances on the failed node are affected. Those jobs will see I/O errors. New instance creation requests will avoid failed server nodes. The WLM will reduce available capacity by that of the failed nodes.

DataWarp periodically monitors the remaining SSD endurance and will stop assigning a server node to new instances when the remaining endurance falls below a configurable threshold. DataWarp server nodes can be drained for maintenance, in which case they do not get assigned new instances and existing job instances are allowed to complete normally. SSDs can also move to new DataWarp server nodes and existing persistent instances on those SSDs will still be usable by future batch jobs.

DataWarp does not provide any data redundancy. While it would be possible to replicate data across multiple DataWarp servers, the increased HSN and SSD traffic would cause a significant reduction in the effective DataWarp bandwidth (and to a lesser extent, a reduction in capacity and endurance), reducing the cost effectiveness of DataWarp.

IV. Performance

DataWarp bandwidth is proportional to the number of DataWarp servers used for the instance. See Figure 8a for the maximum sequential read rate for a scratch instance when using 1 or 2 DataWarp server nodes vs. Lustre OSTs

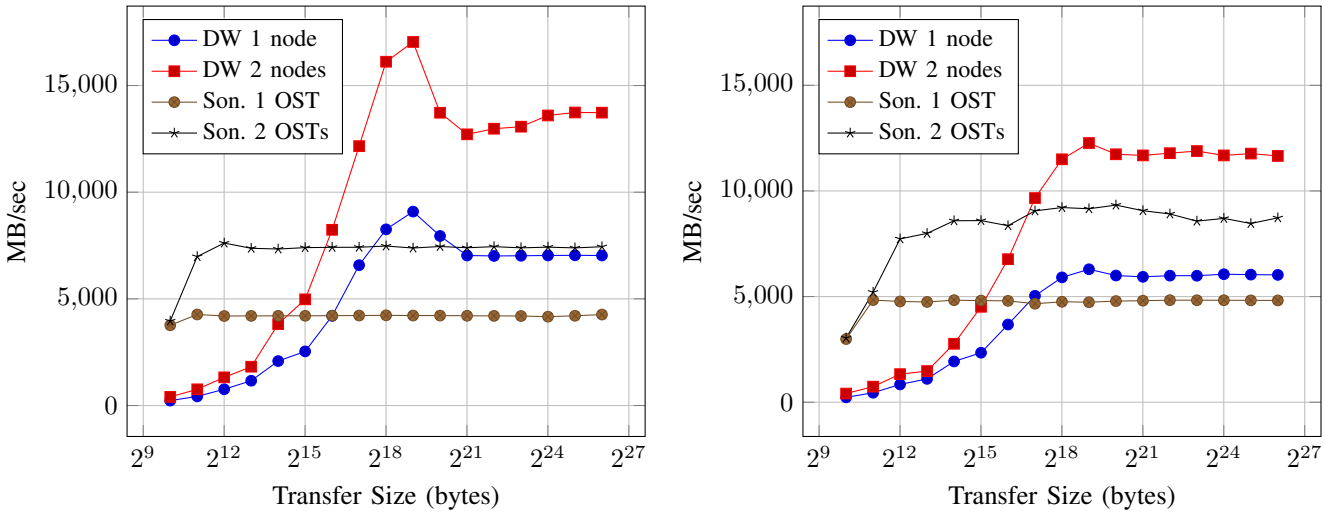


Figure 8. Maximum sequential read rate (8a) and write rate (8b) between DataWarp and Sonexion 2000 with one or two servers or OSTs, respectively.

on Sonexion 2000. We performed measurements on Cray-owned XC systems running CLE 5.2.UP04 with Phase 1 software. We used the IOR benchmark running in POSIX file per process mode accessing 192GiB of data. The DataWarp experiment used 2 ranks per node on 8 or 16 compute nodes for the 1 or 2 servers, respectively. The Lustre experiment used 2 ranks per node on 16 compute nodes. Performance suffers at small transfer sizes due to the lack of client-side caching in Phase 1, but exceeds Lustre performance for 64-128KiB and larger transfer sizes. See Figure 8b for the maximum sequential write rate for the same setup. For DataWarp, read and write performance peaks at 512KiB transfer size before decreasing and leveling off. At 512KiB, the transfer size is large enough to make good use of the Aries network and amortize the fixed RPC cost overhead while being small enough to perform well when the server is low on free memory.

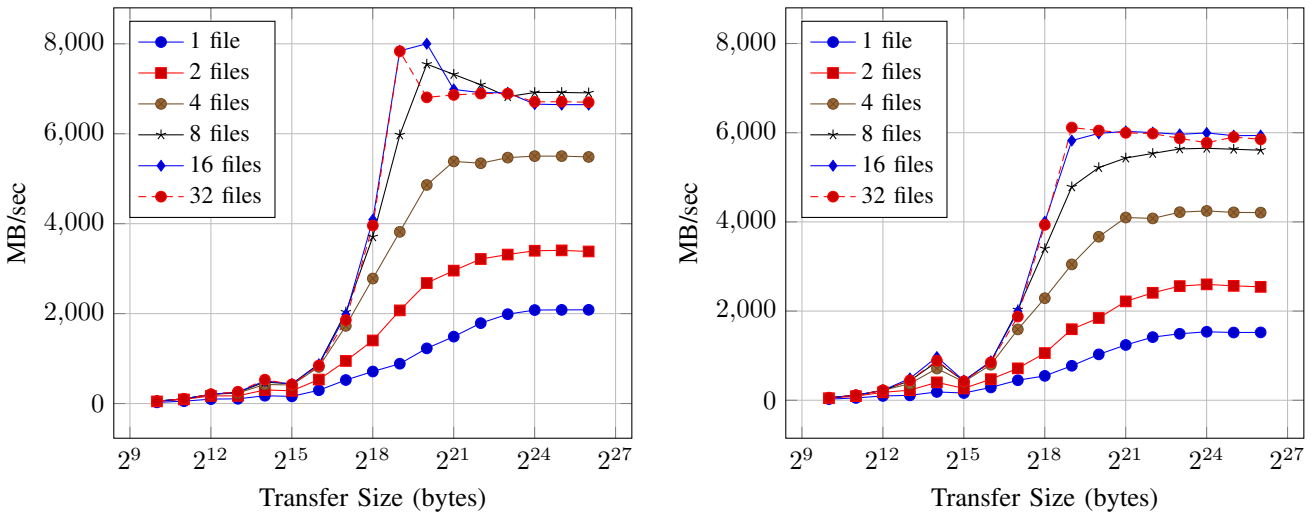
A single compute node can saturate an entire DataWarp server node for both reads and writes if enough streams are used. See Figure 9a for reads and Figure 9b for writes. For reads, saturation occurs at around 8 streams per node. For writes, saturation occurs at around 16 streams per node.

The MSC Nastran structural analysis software [15] exhibits an I/O pattern on some data files that is challenging for spinning disk hardware. Data files are repeatedly read sequentially both forward and backward. On forward passes, prefetching enables fast performance. On backward passes, prefetching is bypassed and the storage hardware must be involved with every read operation. DataWarp's SSD hardware allows for these reads to complete without access penalty, whereas PFS spinning disk hardware must frequently perform expensive seeks. Figure 10 shows a comparison using MSC Nastran between a DataWarp environment and a Lustre environment. This example displays file position over time accessing the SCR300 file, showing the forward and backward passes of reading the factored matrix. DataWarp performance in the forward and backward passes is consistent and fast. On sequential backwards reads, Lustre performance performs well at first due to cache hits in the client cache from spatial locality, but suffers once the spinning disk hardware is accessed. The end result is the DataWarp environment performs twice as fast.

DataWarp performance scales to larger environments. Figure 11 shows the results of running the IOR benchmark on a customer installation with 264 DataWarp servers, each with 2 Intel P3608 SSDs [16]. The test was IOR POSIX file per process running on 5280 compute nodes, 2 ranks per node, and 10560 ranks running Phase 1 DataWarp on CLE5.2.UP04. Additional parameters include a transfer size of 512KiB with 16GiB of data per file, and all files striped across all servers. In aggregate, the write rate achieved 1.54 TB/sec and the read rate achieved 1.66 TB/sec.

V. Conclusion

Provisioning a PFS for both bandwidth and capacity on large HPC systems is cost-prohibitive. DataWarp introduces a storage layer between user applications and the PFS to handle the application's bandwidth needs, which allows the PFS to be provisioned primarily for capacity and resiliency requirements. Interaction between DataWarp and the PFS can happen asynchronously to applications, which reduces the quantity of time that compute nodes need to be allocated to jobs while allowing them to spend more time being used for computation. DataWarp instances can be created on demand per batch job, or they can be shared amongst multiple batch jobs in order to further minimize PFS interaction.



(a) Read

(b) Write

Figure 9. Maximum sequential read rate (9a) and write rate (9b) from one compute node to one DataWarp server node for varying numbers of file streams.

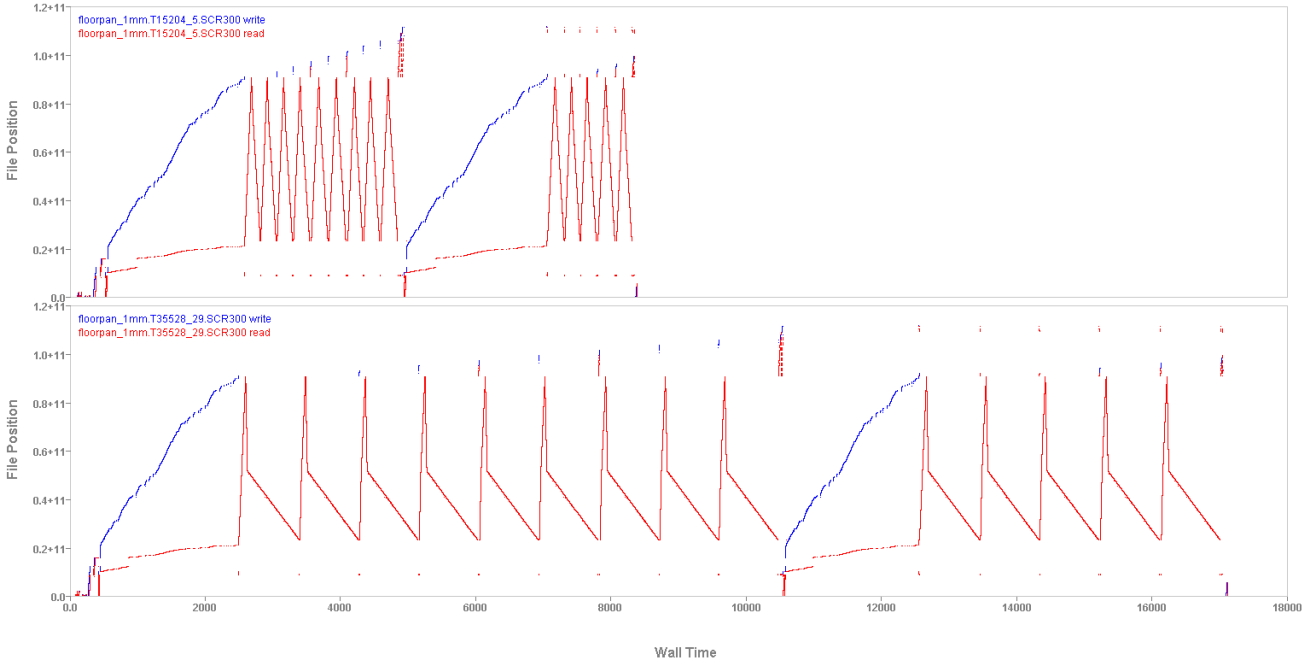


Figure 10. NASTRAN performs sequential reads forwards and backwards multiple times. Traditional spinning disks suffer while reading sequentially backwards due to expensive seeks while DataWarp SSD hardware does not suffer the same penalty. Image courtesy of I/O Doctors, LLC.

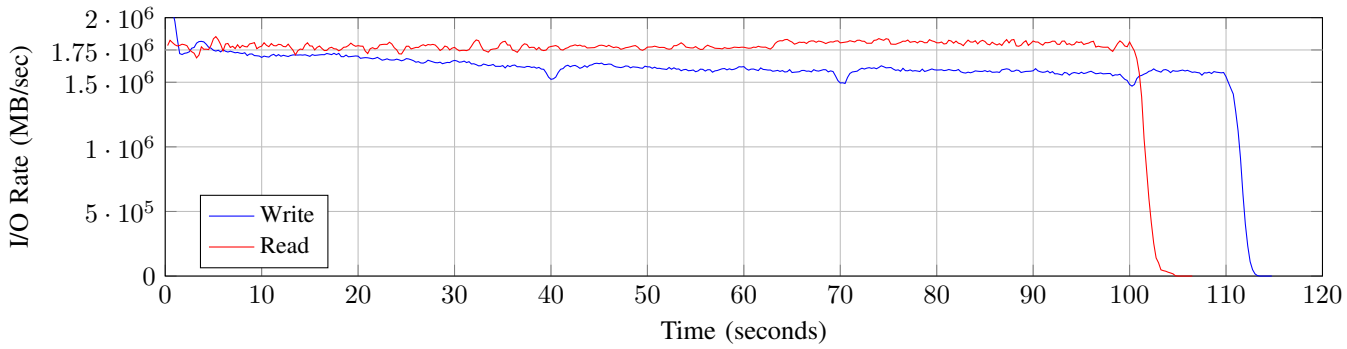


Figure 11. Results from running the IOR benchmark in POSIX file per process mode on 5280 compute nodes with 2 ranks per node against 264 DataWarp servers.

DataWarp accelerates many use cases. As a PFS cache, it accelerates applications that perform checkpoint/restart, have periodic output, or use lots of application library files. As a `/tmp` replacement, it allows for applications to dump state without ever performing PFS I/O or consuming valuable compute node memory. Users can use persistent instances to access frequently used shared data files or have multiple jobs manipulate data before eventually writing final results to a PFS. Using DataWarp for swap files on compute nodes enables applications needing more memory than is present on those nodes to complete successfully. If applications are run directly on DataWarp server nodes they have access to the high IOPS rate provided by the SSD hardware.

DataWarp is capable of providing superior bandwidth compared to a traditional PFS due to the use of high-performance SSDs and a shorter path between compute nodes and DataWarp servers. A single compute node is capable of saturating a single DataWarp server node, and DataWarp scales to hundreds of servers.

Acknowledgment

The authors would like to thank their teams and users at Cray, LANL, NERSC, and KAUST as well as those working in support of DataWarp at Adaptive, Altair, and SchedMD.

References

[1] (2016, February) Lustre Software Release 2.x Operations Manual. [Online]. Available: http://doc.lustre.org/lustre_manual.pdf

[2] (2014, October) GPFS Concepts, Planning, and Installation Guide. IBM. [Online]. Available: http://publib.boulder.ibm.com/epubs/pdf/a7604411.pdf

[3] Software Defined Storage | SNIA. [Online]. Available: http://www.snia.org/sds

[4] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn, "On the role of burst buffers in leadership-class storage systems," in *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, April 2012, pp. 1–11.

[5] (2015, September) DVS Administration Guide. Cray Inc. [Online]. Available: http://docs.cray.com/books/S-0005-5204/S-0005-5204.pdf

[6] (2015, November) Moab Workload Manager – Enterprise Edition. Adaptive Computing, Inc. [Online]. Available: http://docs.adaptivecomputing.com/8-1-2/enterprise/MWM/MoabEnterprise-8.1.2.pdf

[7] (2015, November) TORQUE Resource Manager – Administrator Guide. Adaptive Computing, Inc. [Online]. Available: http://docs.adaptivecomputing.com/torque/5-1-2/torqueAdminGuide-5.1.2.pdf

[8] (2015, June) PBS Professional®13.0 Administrator's Guide. Altair Engineering, Inc. [Online]. Available: http://www.pbsworks.com/pdfs/PBSAdminGuide13.0.pdf

[9] (2015, November) Slurm Burst Buffer Guide. SchedMD LLC. [Online]. Available: http://slurm.schedmd.com/burst_buffer.html

[10] (2015, September) DataWarp User Guide. Cray Inc. [Online]. Available: http://docs.cray.com/books/S-2558-5204/

[11] (2015, June) MUNGE Installation Guide. [Online]. Available: https://github.com/dun/munge/wiki/Installation-Guide

[12] (2016, March) LVM2 Resource Page. [Online]. Available: https://www.sourceware.org/lvm2/

[13] (2016, March) XFS: A high-performance journaling filesystem. [Online]. Available: http://oss.sgi.com/projects/xfs/

[14] A. Barry, "Resource Utilization Reporting," in *Proc. Cray Users' Group Technical Conference (CUG)*, 2013.

[15] (2016, March) MSC Nastran - Multidisciplinary Structural Analysis. MSC Software Corporation. [Online]. Available: http://www.mscsoftware.com/product/msc-nastran

[16] Intel®SSD DC P3608 Series. Intel Corporation. [Online]. Available: http://www.intel.com/content/www/us/en/solid-state-drives/solid-state-drives-dc-p3608-series.html