DataWarp Administration Guide S-2557-5204b

Contents

1 About the DataWarp Administration Guide

This publication covers administrative concepts and tasks for Cray XC™ series systems running CLE5.2.UP04 and installed with DataWarp SSD cards; it is intended for system administrators.

Release Information

This publication includes information, guidance, and procedures for DataWarp on Cray XC series systems running software release CLE5.2.UP04. It supports both implementations of DataWarp: DataWarp with DWS and Static DataWarp.

Record of Revision

Revision: `b` (03-02-16): Included source code for the `dwpoolhelp` command and modified the *DataWarp with DWS: Create a Storage Pool* on page 35 procedure to use `dwpoolhelp`.

Revision: `a` (11-06-15): Clean up errors and incorporate bugfixes.

Typographic Conventions

`Monospace`	Indicates program code, reserved words, library functions, command-line prompts, screen output, file/path names, key strokes (e.g., `Enter` and `Alt-Ctrl-F`), and other software constructs.
`Monospaced Bold`	Indicates commands that must be entered on a command line or in response to an interactive prompt.
`Oblique` or *`Italics`*	Indicates user-supplied values in commands or syntax definitions.
Proportional Bold	Indicates a graphical user interface window or element.
\ (backslash)	At the end of a command line, indicates the Linux® shell line continuation character (lines joined by a backslash are parsed as a single line). Do not type anything after the backslash or the continuation feature will not work correctly.

Feedback

Visit the Cray Publications Portal at *http://pubs.cray.com* and provide comments online using the *Contact Us* button in the upper-right corner or Email *pubs@cray.com*.

2 Important Information about this DataWarp Release

This release of DataWarp (hereinafter referred to as *DataWarp with DWS*) marks a significant change from the original statically-configured release (hereinafter referred to as *Static DataWarp*). DataWarp with DWS introduces the following features:

- DataWarp Service (DWS): dynamically allocates DataWarp capacity and bandwidth to jobs on request
- Administrator command line interface - `dwcli`: a DataWarp resource management tool
- DataWarp status command - `dwstat`: provides information about the various DataWarp resources
- User API - `libdatawarp`: provides compute applications with functions to control and query the staging of data

As a result of these major improvements and additional upcoming features, Static DataWarp is deprecated and will be removed in the next major release of the Cray Linux Environment (CLE). Cray encourages sites to switch from Static DataWarp to DataWarp with DWS to take advantage of the features mentioned above.

IMPORTANT: Because this release supports both types of DataWarp, it is important to note the following:

- There are procedural differences during installation. These differences are indicated in *DataWarp Installation and Configuration Guide* by the phrases **`Static DataWarp`** and **`DataWarp with DWS`**.
- The *DataWarp Administration Guide* supports DataWarp with DWS only.
- The *DataWarp User Guide* supports DataWarp with DWS only.

TIP: All DataWarp documentation describes units of bytes using the binary prefixes defined by the International Electrotechnical Commission (IEC), e.g., MiB, GiB, TiB. For further information, see *Prefixes for Binary and Decimal Multiples* on page 59.

3 About DataWarp

> **TIP:** All DataWarp documentation describes units of bytes using the binary prefixes defined by the International Electrotechnical Commission (IEC), e.g., MiB, GiB, TiB. For further information, see *Prefixes for Binary and Decimal Multiples* on page 59.

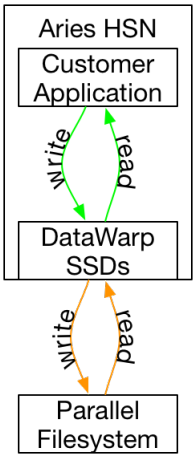
Cray DataWarp provides an intermediate layer of high bandwidth, file-based storage to applications running on compute nodes. It is comprised of commercial SSD hardware and software, Linux community software, and Cray system hardware and software. DataWarp storage is located on server nodes connected to the Cray system's high speed network (HSN). I/O operations to this storage completes faster than I/O to the attached parallel file system (PFS), allowing the application to resume computation more quickly and resulting in improved application performance. DataWarp storage is transparently available to applications via standard POSIX I/O operations and can be configured in multiple ways for different purposes. DataWarp capacity and bandwidth are dynamically allocated to jobs on request and can be scaled up by adding DataWarp server nodes to the system.

Each DataWarp server node can be configured either for use by the DataWarp infrastructure or for a site specific purpose such as a Hadoop distributed file system (HDFS).

> **IMPORTANT:** Keep in mind that DataWarp is focused on performance and not long-term storage. SSDs can and do fail.

The following diagram is a high level view of DataWarp. SSDs on the Cray high-speed network enable compute node applications to quickly read and write data to the SSDs, and the DataWarp file system handles staging data to and from a parallel filesystem.

Figure 1. DataWarp Overview



DataWarp Use Cases

There are four basic use cases for DataWarp:

Parallel file	DataWarp can be used to cache data between an application and the PFS. This allows PFS I/O to be overlapped with an application's computation. Initially, data movement (staging) between

system (PFS) cache

DataWarp and the PFS must be explicitly requested by a job and/or application and then performed by the DataWarp service. In a future release, data staging between DataWarp and the PFS can also be done implicitly (i.e., read ahead and write behind) by the DataWarp service without application intervention. Examples of PFS cache use cases include:

- **Checkpoint/Restart**: Writing periodic checkpoint files is a common fault tolerance practice for long running applications. Checkpoint files written to DataWarp benefit from the high bandwidth. These checkpoints either reside in DataWarp for fast restart in the event of a compute node failure, or are copied to the PFS to support restart in the event of a system failure.
- **Periodic output**: Output produced periodically by an application (e.g., time series data) is written to DataWarp faster than to the PFS. Then as the application resumes computation, the data is copied from DataWarp to the PFS asynchronously.
- **Application libraries**: Some applications reference a large number of libraries from every rank (e.g., Python applications). Those libraries are copied from the PFS to DataWarp once and then directly accessed by all ranks of the application.

Scratch storage

DataWarp can provide storage that functions like a `/tmp` file system for each compute node in a job. This data typically does not touch the PFS, but it can also be configured as PFS cache. Applications that use out-of-core algorithms, such as geographic information systems, can use DataWarp scratch storage to improve performance.

Shared storage

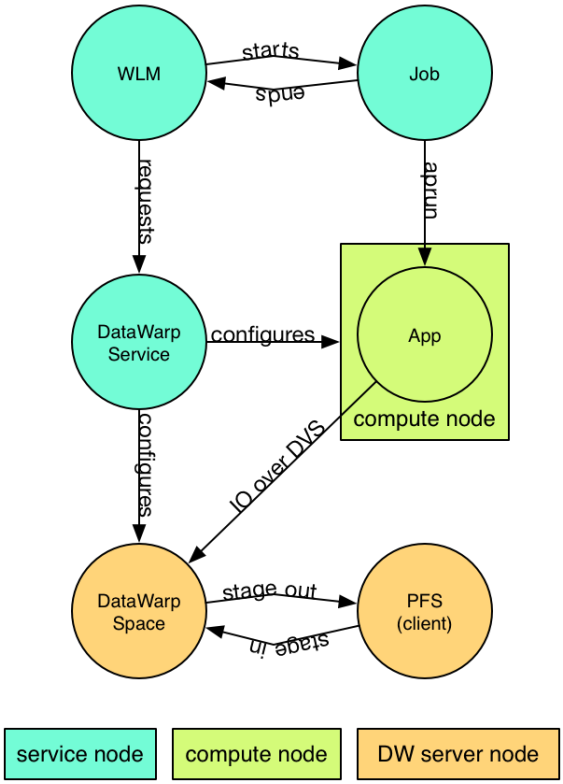
DataWarp storage can be shared by multiple jobs over a configurable period of time. The jobs may or may not be related and may run concurrently or serially. The shared data may be available before a job begins, extend after a job completes, and encompass multiple jobs. Shared data use cases include:

- **Shared input**: A read-only file or database (e.g., a bioinformatics database) used as input by multiple analysis jobs is copied from PFS to DataWarp and shared.
- **Ensemble analysis**: This is often a special case of the above **shared input** for a set of similar runs with different parameters on the same inputs, but can also allow for some minor modification of the input data across the runs in a set. Many simulation stategies use ensembles.
- **In-transit analysis**: This is when the results of one job are passed as the input of a subsequent job (typically using job dependencies). The data can reside only on DataWarp storage and may 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.

3.1 Overview of the DataWarp Process

Refer to Figures *DataWarp Component Interaction - bird's eye view* on page 7 and *DataWarp Component Interaction - detailed view* on page 8 for visual representation of the process.

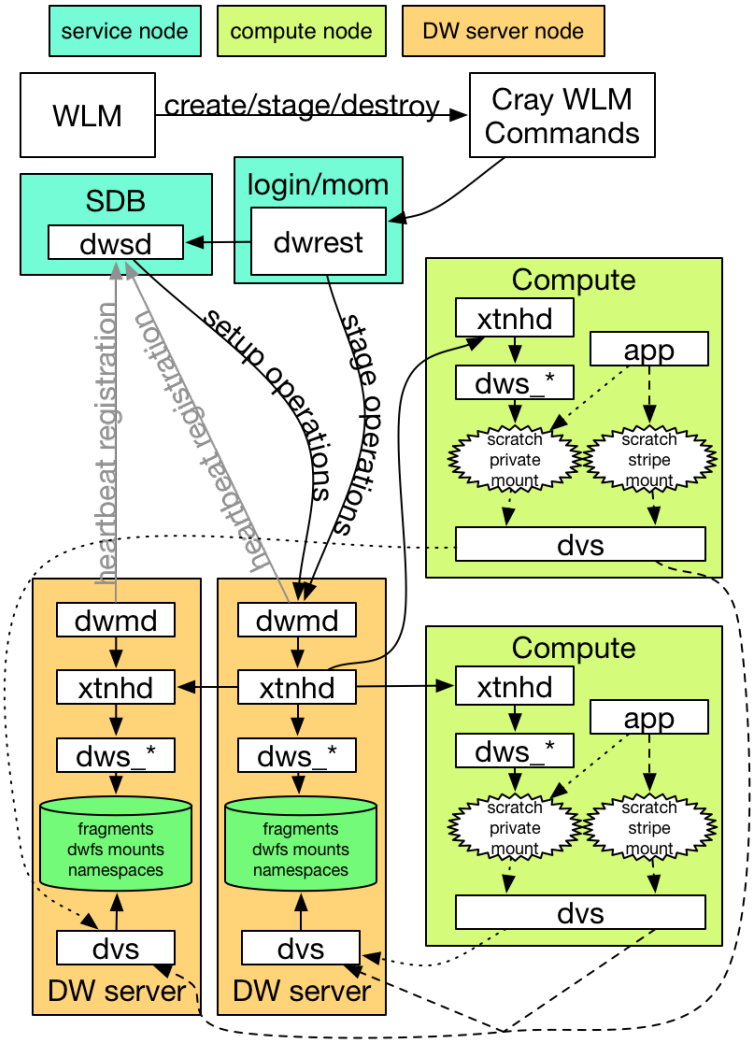
Figure 2. DataWarp Component Interaction - bird's eye view



1. A user submits a job to a workload manager. Within the job submission, the user must specify: the amount of DataWarp storage required, how the storage is to be configured, and whether files are to be staged from the PFS to DataWarp or from DataWarp to the PFS.
2. The workload manager provides queued access to DataWarp by first querying the DataWarp service for the total aggregate capacity. The requested capacity is used as a job scheduling constraint. When sufficient DataWarp capacity is available and other WLM requirements are satisfied, the workload manager requests the needed capacity and passes along other user-supplied configuration and staging requests.
3. The DataWarp service dynamically assigns the storage and initiates the stage in process.
4. After this completes, the workload manager acquires other resources needed for the batch job, such as compute nodes.
5. After the compute nodes are assigned, the workload manager and DataWarp service work together to make the configured DataWarp accessible to the job's compute nodes. This occurs prior to execution of the batch job script.
6. The batch job runs and any subsequent applications can interact with DataWarp as needed (e.g., stage additional files, read/write data).
7. When the batch job ends, the workload manager stages out files, if requested, and performs cleanup. First, the workload manager releases the compute resources and requests that the DataWarp service make the previously accessible DataWarp configuration inaccessible to the compute nodes. Next, the workload manager requests that additional files, if any, are staged out. When this completes, the workload manager tells the DataWarp service that the DataWarp storage is no longer needed.

The following diagram includes extra details regarding the interaction between a WLM and the DWS as well as the location of the various DWS daemons.

Figure 3. DataWarp Component Interaction - detailed view



3.2 DataWarp Concepts

For basic definitions, refer to *Terminology* on page 57.

Instances

DataWarp storage is assigned dynamically when requested, and that storage is referred to as an *instance*. The space is allocated on one or more DataWarp server nodes and is dedicated to the instance for the lifetime of the instance. A DataWarp instance has a lifetime that is specified when the instance is created, either *job instance* or *persistent instance*. A job instance is relevant to all previously described use cases except the shared data use case.

- **Job instance**: The lifetime of a job instance, as it sounds, is the lifetime of the job that created it, and is accessible only by the job that created it.

- **Persistent instance**: The lifetime of a persistent instance is not tied to the lifetime of any single job and is terminated by command. Access can be requested by any job, but file access is authenticated and authorized based on the POSIX file permissions of the individual files. Jobs request access to an existing persistent instance using a persistent instance name. A persistent instance is relevant only to the shared data use case.

  WARNING: New DataWarp software releases may require the re-creation of persistent instances.

When either type of instance is destroyed, DataWarp ensures that data needing to be written to the PFS is written before releasing the space for reuse. In the case of a job instance, this can delay the completion of the job.

Application I/O

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

For this release, a DataWarp instance can be configured as scratch. Additionally, all data staging between either type of instance and the PFS must be explicitly requested using the DataWarp job script staging commands or the application C library API (`libdatawarp`). In a future release, an instance will be configurable as cache, and all data staging between the cache instance and the PFS will occur implicitly.

A scratch configuration 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 reside on one DataWarp server node. For scratch instances the files are only accessible to the compute node that created them (e.g., /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.

  Private access mode assigns each compute node in a job to one of the DataWarp servers assigned to the job. The number of compute nodes assigned to a server is proportional to the amount of storage assigned to the job on that server. For example, if two servers are assigned and each provide half of the requested storage, then the compute nodes are distributed equally across the two servers. However, if one server provides 25% and the other 75% of the storage, then the compute nodes are distributed 25% and 75% as well.

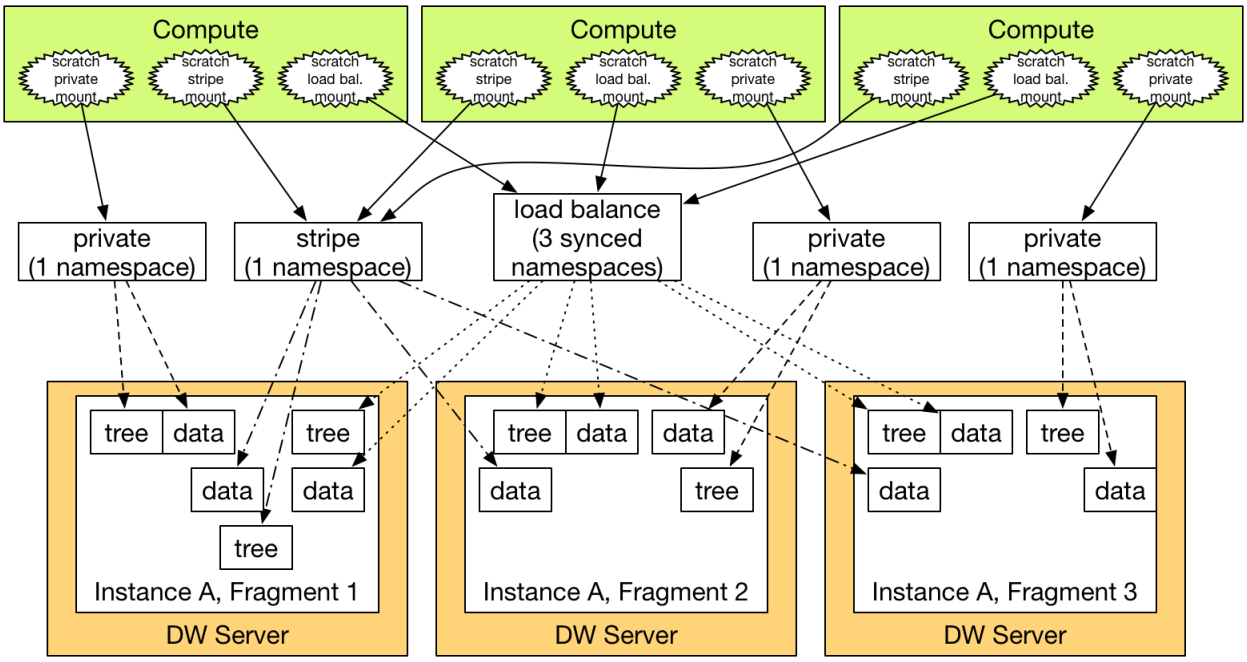
  Private access mode assumes that all compute nodes assigned to a job have similar capacity and bandwidth requirements.

There is a separate file namespace for every instance (job and persistent), type (scratch), and access mode (striped, private) except persistent/private is not supported. The file path prefix for each is provided to the job via environment variables.

- **Striped**: All compute nodes share one namespace; files stripe across all servers.
- **Private**: Each compute node gets its own namespace. Each namespace maps to one server node, therefore, files in a namespace are only on one server node.

The following diagram shows a scratch private, scratch stripe, and scratch load balance (deferred implementation) mount point on each of three compute (client) nodes. For scratch private, each compute node reads and writes to its own namespace that exists on one of the DataWarp server nodes. For scratch stripe, each compute node reads and writes to a common namespace, and that namespace spans all three DataWarp

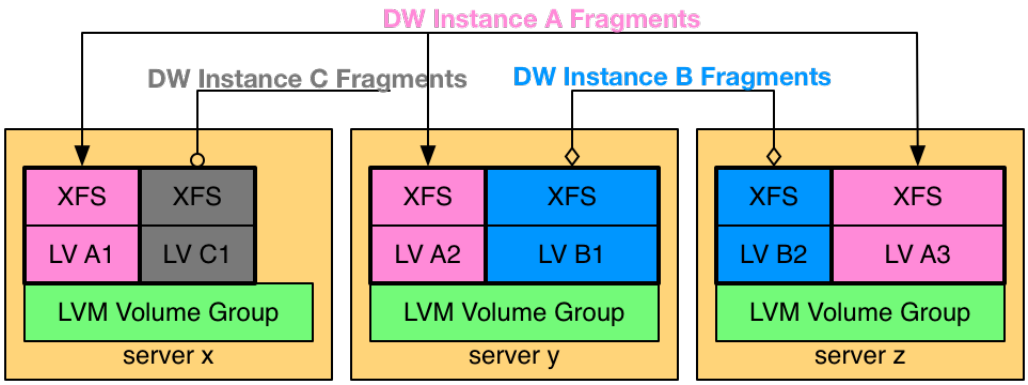
nodes. For scratch load balance (deferred implementation), each compute node reads from one of many synchronized namespaces (no writes allowed). The compute node - namespace mapping is based on a hashing algorithm.



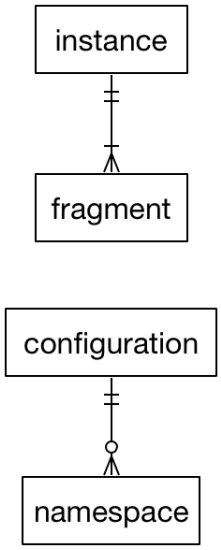
3.3 Instances and Fragments - a Detailed Look

The DataWarp Service (DWS) provides user access to subsets of storage space that exist between an arbitrary filesystem path (typically that of a Parallel File System) and a client (typically a compute node in a batch job). Storage space typically exists on multiple server nodes. On each server node, LVM combines block devices and presents them to the DWS as an LVM volume group. All of the LVM volume groups on all of the server nodes compose the aggregate storage space. A specific subset of the storage space is called a DataWarp *instance*, and typically spans multiple server nodes. Each piece of a DataWarp instance (as it exists on each server node) is called a DataWarp instance fragment. A DataWarp instance fragment is implemented as an LVM logical volume.

The following figure is an example of three DataWarp instances. DataWarp instance A consists of fragments that map to LVM logical volumes A1, A2, and A3 on servers x, y, z, respectively. DataWarp Instance B consists of fragments that map to LVM logical volumes y and z, respectively. DataWarp Instance C consists of a single fragment that maps to LVM logical volume C1 on server x.



The following diagram uses Crow's foot notation to illustrate the relationship between an instance-fragment and a configuration-namespace. One instance has one or more fragments; a fragment can belong to only one instance. A configuration has 0 or more namespaces; a namespace can belong to only one configuration.



3.4 Storage Pools

A storage pool groups nodes with storage together such that requests for space made against the pool are fufilled from the nodes associated with the pool with a common allocation granularity. Pools have either byte or node allocation granularity (*pool_AG*). This release of DWS only supports byte allocation granularity. There are tradeoffs in picking allocation granularities too small or too large.

> **TIP:** Throughout these procedures, units of bytes are described using the binary prefixes defined by the International Electrotechnical Commission (IEC). For further information, see *Prefixes for Binary and Decimal Multiples* on page 59.

The following are important considerations when creating a storage pool:

1. The byte-oriented allocation granularity for a pool must be at least 16MiB.
2. Each node's volume group (`dwcache`, configured in *Initialize an SSD* on page 45) has a Physical Extent size (*PE_size*) and Physical Volume count (*PV_count*). The default *PE_size* is 4MiB, and *PV_count* is

equal to the number of Physical Volumes specified during volume group creation. DWS places the following restriction on nodes associated with a pool:

- A node can only be associated with a storage pool if the node's granularity (*PE_size* * *PV_count*) is a factor of the pool's allocation granularity (*pool_AG*). The `dwstat nodes` command lists the node's granularity in the `gran` column.

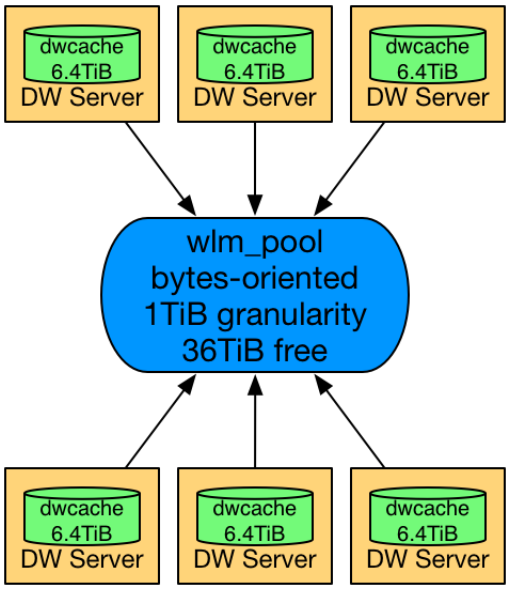
3. The more nodes in the pool, the higher the granularity.
4. Ideally, a pool's allocation granularity is defined as a factor of the aggregate space of each node within the pool; otherwise, some space is not usable and, therefore, is wasted.

The most important considerations are #1 and #2. On all Cray systems, picking a pool granularity of at least 16MiB (16,777,216 bytes) and is a multiple of 16MiB (16,777,216, 33,554,432, 50,331,648, ...) will define a functioning, but possibly sub-optimal, configuration. The following recommendation table does not take #3 and #4 into consideration but will be a good starting point for all Cray system configurations:

Table 1. Nodes

SSD Nodes in Pool	Granularity (bytes)
1	16,777,216
<20	214,748,364,800
<100	429,496,729,600
<200	858,993,459,200
>=200	1,073,741,824,000

The following diagram shows six different DataWarp nodes belonging to a storage pool `wlm_pool` with a 1TiB allocation granularity. Each DataWarp node has 6.4TiB of space, which means that 0.4TiB are wasted per node because only 6 allocation granularities fit on any one node.



3.5 Registrations

A configuration represents a way to use the DataWarp space. Configurations are used in one of two ways:

- configurations are activated
- data is staged into or out of configurations

When either of these actions are performed, the action must supply a DataWarp session identifier in addition to other action-specific data such as a mount point. The session identifier is required because the DataWarp Service (DWS) keeps track of whether a configuration is used and which sessions used it. Then, when requested to remove either the configuration or session, the DWS cleans things up correctly.

The first time a configuration is used by a session, the DWS automatically creates a *registration* entry that binds together the session and configuration. The registration is automatically requested to be removed when either the linked configuration or session is requested to be removed. The actions performed at registration removal time depend on two things:

1. The type of configuration linked with the registration
2. The value of the registration's `wait` attribute
    - By default, `wait=true`, resulting in the execution of some configuration-specific actions prior to the complete removal of the registration.

For this release, the only type of configuration is `scratch`. The DWS carries out the following operations for registrations that are linked with a scratch configuration:

1. Files marked for stage out by a user application (using `libdatawarp`, see *libdatawarp - the DataWarp API*) in a batch job with the `DW_STAGE_AT_JOB_END` stage type are transitioned to being immediately staged out.
2. All existing stage out activity, including the stage out from the previous step, is allowed to fully complete.

If the above process is interrupted, e.g., a DataWarp server node crashes, the DWS attempts to restore everything associated with the node and restart the process after the node reboots. This includes restoring any logical volumes or mount points that are associated with the configuration.

There are times when the previous behavior is not desired. Consider either of the following:

- A DWS or underlying software bug exists that prevents the restoration of the DataWarp state on a crashed server node
- Hardware fails such that data on the SSD is permanently lost

In situations like this, set `wait=false` for the registration in order to tell the DWS to abort the normal cleanup process. For example, the following registration is in the process of being destroyed but cannot finish because a linked SSD has failed:

```
user> dwstat registrations
reg state sess conf wait
  2 D----    5   11 true
```

Instruct the DWS to abort the normal registration removal tasks by setting the `wait=false` with the following `dwcli` command:

```
user> dwcli update registration --id 2 --no-wait
```

WARNING: Use of `--no-wait` can lead to data loss because some data may not have been staged out to the PFS.

WLM Interation with Registrations

Registration removal blocks batch job removal because the registration belongs to a session, which in turn belongs to a batch job. Each WLM provides its own way to force the removal of a batch job. Each of the DataWarp-integrated WLMs have been modified to automatically set the `wait` attribute of registrations to `false` when the WLM-specific job removal force option is used. It is only necessary to set `wait=false` using `dwcli` for registrations without a corresponding WLM batch job to force remove.

4 DataWarp Administrator Tools

4.1 dwcli(8)

NAME

`dwcli` - Command line interface for DataWarp

SYNOPSIS

`dwcli` [*common_options*] [*ACTION RESOURCE* [*resource_attributes*]]

DESCRIPTION

The `dwcli` command provides a command line interface to act upon DataWarp resources. This is primarily an administration command, although a user can initiate some actions using it. With full WLM support, a user does not have a need for this command.

> **IMPORTANT:** The `dws` module must be loaded to use this command.
>
> ```
> $ module load dws
> ```

COMMON OPTIONS

`dwcli` accepts the following common options:

`--debug`

Enable debug mode

`-h | --help`

Display usage information for the command, actions, and resources:

- `dwcli -h`
- `dwcli` *`action`* -h
- `dwcli` *`action resource`* -h

`-j | --json`

Display debug output as json if applicable (not valid with `--debug`)

`-r` *`ROLE`*

Request a role outside the user's level

`-s | --short`

Display abbreviated `create` output

-v|--version

Display `dwcli` version information

ACTIONS

The following actions are available:

create Create resource

Valid for: `activation`, `configuration`, `instance`, `pool`, and `session`.

ls Display information about a resource

Valid for: `activation`, `configuration`, `instance`, `fragment`, `namespace`, `node`, `pool`, `registration`, and `session`.

rm Remove a resource

Valid for: `activation`, `configuration`, `instance`, `pool`, `registration`, and `session`.

stage Stage files and directories in or out

Valid for options: `in`, `out`, `query`, and `terminate`.

update Update the attributes of a resource

Valid for: `activation`, `configuration`, `instance`, `node`, `registration`, and `session`.

RESOURCES

`dwcli` accepts the following resources:

- **activation**

  A DataWarp activation is an object that represents an available instance configuration on a set of nodes. The activation resource has the following attributes:

  --configuration *CONFIGURATION*

  Numeric configuration ID for activation

  --hosts *CLIENT_NODES*

  Hostnames on which the referenced datawarp instance configuration may be activated. If not defined, the hostnames associated with *`SESSION`* are used. If defined, the hostnames must be a subset of those associated with *`SESSION`*.

  --id *ID*

  Numeric activation ID

  --mount *MOUNT*

  Client mount directory for scratch configurations

  --replace-fuse

  Directs DWS to replace the activation's fuse and retry activating it

  --session *SESSION*

  Numeric ID of session with which the datawarp activation is associated

- **configuration**

A DataWarp configuration represents a specific way in which a DataWarp instance is used. The configuration resource has the following attributes:

--access-type *ACCESS_TYPE*

Type of access, either `stripe` or `private`

--group *GROUP_ID*

Numeric group ID for the root directory of the storage

-i | --id *ID*

Numeric configuration ID

--instance *INSTANCE*

Numeric ID of instance in which this configuration exists

--max-files-created *MAX_FILES_CREATED*

Maximum number of files allowed to be created in a single configuration namespace

--max-file-size *MAX_FILE_SIZE*

Maximum file size, in bytes, for any file in the configuration namespace

--replace-fuse

Directs DWS to replace the configuration's fuse and retry configuration tasks

--root-permissions *ROOT_PERMISSIONS*

File system permissions set on the root directory for storage of type scratch, in octal format (e.g., 0777)

--type *TYPE*

Type of configuration; currently only `scratch` is valid

- **fragment**

  A DataWarp fragment is a subset of managed space found on a DataWarp node. The fragment resource has no attributes available for this command.

- **instance**

  A DataWarp instance is a collection of DataWarp fragments, where no two fragments in a DataWarp instance exist on the same node. DataWarp instances are essentially unusable raw space until at least one DataWarp instance configuration is created, specifying how the space is used and accessed. A DataWarp instance may not be created unless a DataWarp session is supplied at creation time. The instance resource has the following attributes:

  --capacity *size*

  Instance capacity in bytes

  --expiration *epoch*

  Expiration time in Unix, or epoch, time

  -i | --id *ID*

  Numeric instance ID

  --label *LABEL*

  Instance label name

  --optimization *OPTIMIZATION*

  Requested optimization strategy; options are `bandwidth`, `interference`, and `wear`. Specifying `bandwidth` optimization results in the DWS picking as many server nodes as

possible while satisfying the capacity request. Specifying `interference` optimization results in the DWS picking as few server nodes as possible when satisfying the capacity request. Specifying `wear` optimization results in the DWS picking server nodes primarily on the basis of the health of the SSDs on the server nodes. Both `bandwidth` and `interference` make use of SSD health data as a second-level optimization.

`--pool` *`pname`*

Name of pool with which the instance is associated

`--private`

Controls the visibility of the instance being created; private is visible only to administrators and the user listed in *`SESSION`*

`--public`

Controls the visibility of the instance being created; public is visible to all users. Persistent datawarp instances, which are meant to be shared by multiple users, are required to be public.

`--replace-fuse`

Directs DWS to replace the instance's fuse and retry instance tasks

`--session` *`SESSION`*

Numeric ID of session with which the instance is associated

`--write-window-length` *`WW_LENGTH`*

Write window duration in seconds; used with `--write-window-multiplier`

`--write-window-multiplier` *`WW_MULTIPLIER`*

Used with `--write-window-length`, for each fragment comprising the instance, the size of the fragment is multiplied by *`WW_MULTIPLIER`* and the user is allowed to write that much data to the fragment in a moving window of *`WW_LENGTH`*. When the limit is exceeded, the `scratch_limit_action` specified by the system administrator in `dwsd.yaml`, is performed. This can aid in the detection of anomalous usage of a DataWarp instance.

- **node**

  A DataWarp node can host DataWarp capacity (server node), have DataWarp configurations activated on it (client node), or both. The node resource has the following attributes that are only valid with `update`:

  `--drain`

  Set `drain=true`; do not use for future instance requests

  `-n` | `--name` *`NAME`*

  Hostname of node

  `--no-drain`

  Set `drain=false`; node is available for requests

  `--pool` *`POOL`*

  Name of pool to which this node belongs

  `--rm-pool`

  Disassociate the node from a pool

- **pool**

  A DataWarp pool represents an aggregate DataWarp capacity. The pool resource has the following attributes:

  `--granularity` *`GRANULARITY`*

Pool allocation granularity in bytes

-n | --name *NAME*

Pool name

- **registration**

  A DataWarp registration represents a known use of a configuration by a session. The registration resource has the following attributes:

  -i | --id *ID*

  Numeric registration ID

  --no-wait

  Set `wait=false`; do not wait for associated configurations to finish asynchronous activities such as waiting for all staged out data to finish staging out to the PFS

  --replace-fuse

  Directs DWS to replace the registration's fuse and begin retrying registration tasks

  --wait

  Set `wait=true`; wait for associated configurations to finish asynchronous activities

- **session**

  A DataWarp session is an object used to map events between a client context (e.g., a WLM batch job) and a DataWarp service context. It establishes node authorization rights for activation purposes, and actions performed through the session are undone when the session is removed. The session resource has the following attributes:

  --creator *CREATOR*

  Name of session creator

  --expiration *EXPIRATION*

  Expiration time in Unix, or epoch, time. If `0`, the session never expires.

  --hosts *CLIENT_NODE* [*CLIENT_NODE*...]

  List of hostnames to which the session is authorized access

  -i | --id *ID*

  Numeric session ID

  --owner *OWNER*

  Userid of session owner

  --replace-fuse

  Directs DWS to replace the session's fuse and retry session tasks

  --token *TOKEN*

  Session label

STAGE OPTIONS

The `stage` action stages files/directories and accepts the following options:

- **in**: stage a file or directory from a PFS into DataWarp. The following arguments are accepted:

  -b | --backing-path *BACKING_PATH*

  Path of file/directory to stage into the DataWarp file system

  -c | --configuration *CONFIGURATION_ID*

Numeric configuration ID

-d | --dir *DIRNAME*

Name of directory to stage into the DataWarp file system

-f | --file *FILENAME*

Name of file to stage into the DataWarp file system

-s | --session *SESSION*

Numeric session ID

- **list**: provides a recursive listing of all files with stage attributes for a staging session/configuration. The following arguments are accepted:

  -c | --configuration *CONFIGURATION_ID*

  Numeric configuration ID

  -s | --session *SESSION*

  Numeric session ID

- **out**: stage a file or directory out of the DataWarp file system to a PFS. The following arguments are accepted:

  -b | --backing-path *BACKING_PATH*

  PFS path to where file/directory is staged out

  -c | --configuration *CONFIGURATION_ID*

  Numeric configuration ID

  -d | --dir *DIRNAME*

  Name of directory to stage out to the PFS

  -f | --file *FILENAME*

  Name of file to stage out to the PFS

  -s | --session *SESSION*

  Numeric session ID

- **query**: query staging status for a file or directory. The following arguments are accepted:

  -c | --configuration *CONFIGURATION_ID*

  Numeric configuration ID

  -d | --dir *DIRNAME*

  Name of a DataWarp directory to query (optional)

  -f | --file *FILENAME*

  Name of a DataWarp file to query (optional)

  -s | --session *SESSION*

  Numeric session ID

- **terminate**: terminate a current stage operation. The following arguments are accepted:

  -c | --configuration *CONFIGURATION_ID*

  Numeric configuration ID

  -d | --dir *DIRNAME*

  Name of directory for which staging is terminated

  -f | --file *FILENAME*

Name of file for which staging is terminated

-s | --session ***SESSION***

Numeric session ID

EXAMPLE: Create a pool

Only an administrator can execute this command.

```
smw# dwcli create pool --name example-pool --granularity 16777216
created pool name example-pool
```

EXAMPLE: Assign a node to the pool

Only an administrator can execute this command.

```
smw# dwcli update node --name example-node --pool example-pool
```

EXAMPLE: Create a session

Only an administrator can execute this command.

```
$ dwcli create session --expiration 4000000000 --creator $(id -un) --token example-
session --owner $(id -u) --hosts example-node
created session id 10
```

EXAMPLE: Create an instance

Only an administrator can execute this command.

```
$ dwcli create instance --expiration 4000000000 --public --session 10 --pool
example-poolname --capacity 1099511627776 --label example-instance --optimization
bandwidth
created instance id 8
```

EXAMPLE: Create a configuration

```
$ dwcli create configuration --type scratch --access-type stripe --root-
permissions 0755 --instance 8 --group 513
created configuration id 7
```

EXAMPLE: Create an activation

```
$ create activation --mount /some/pfs/mount/directory --configuration 7 --session
10
created activation id 7
```

EXAMPLE: Set a registration to `--no-wait`

Directs DWS to not wait for associated configurations to finish asynchronous activities such as waiting for all staged out data to finish staging out to the PFS. Note that no output after this command indicates success.

```
$ dwcli update registration --id 1 --no-wait
$
```

EXAMPLE: Remove a pool

Only an administrator can execute this command.

```
$ dwstat pools
  pool units quantity    free  gran
canary bytes  3.98GiB 3.97GiB 16MiB
$ dwcli rm pool --name canary
$ dwstat pools
no pools
```

EXAMPLE: Remove a session

Only an administrator can execute this command.

```
$ dwstat sessions
sess state token creator owner             created expiration nodes
   1 CA---    ok    test 12345 2015-09-18T16:31:24    expired     1

$ dwcli rm session --id 1
sess state token creator owner             created expiration nodes
   1 D----    ok    test 12345 2015-09-18T16:31:24    expired     0

After some time...
$ dwstat sessions
no sessions
```

EXAMPLE: Fuse replacement

```
$ dwstat instances
inst state sess bytes nodes            created expiration intact           label public confs
   1 D-F-M    1 16MiB     1 2015-09-18T17:47:57    expired  false canary-instance   true     1
$ dwcli update instance --replace-fuse --id 1
$ dwstat instances
inst state sess bytes nodes            created expiration intact           label public confs
   1 D---M    1 16MiB     1 2015-09-18T17:47:57    expired  false canary-instance   true     1
```

EXAMPLE: Stage in a directory, query immediately, then stage list

```
$ dwcli stage in --session $session --configuration $configuration --dir=/tld/. --backing-path=/tmp/
demo/
path   backing-path nss ->c ->q ->f <-c <-q <-f <-m
/tld/. -            1   3   1   -   -   -   -   -

$ dwcli stage query --session $session --configuration $configuration
path  backing-path nss ->c ->q ->f <-c <-q <-f <-m
/.    -            1   4   -   -   -   -   -   -
/tld/ -            1   4   -   -   -   -   -   -

$ dwcli stage list --session $session --configuration $configuration
path                 backing-path               nss ->c ->q ->f <-c <-q <-f <-m
/tld/filea           /tmp/demo/filea            1   1   -   -   -   -   -   -
/tld/fileb           /tmp/demo/fileb            1   1   -   -   -   -   -   -
/tld/subdir/subdirfile /tmp/demo/subdir/subdirfile 1   1   -   -   -   -   -   -
/tld/subdir/subfile  /tmp/demo/subdir/subfile   1   1   -   -   -   -   -   -
```

EXAMPLE: Stage a file in afterwards, stage list, then query

Note the difference in the `stage query` output.

```
$ dwcli stage in --session $session --configuration $configuration --file /dwfsfile --backing-path /tmp/
demo/filea
```

```
path      backing-path    nss ->c ->q ->f <-c <-q <-f <-m
/dwfsfile /tmp/demo/filea 1   1   -   -   -   -   -   -

$ dwcli stage list --session $session --configuration $configuration
path                  backing-path               nss ->c ->q ->f <-c <-q <-f <-m
/dwfsfile             /tmp/demo/filea            1   1   -   -   -   -   -   -
/tld/filea            /tmp/demo/filea            1   1   -   -   -   -   -   -
/tld/fileb            /tmp/demo/fileb            1   1   -   -   -   -   -   -
/tld/subdir/subdirfile /tmp/demo/subdir/subdirfile 1   1   -   -   -   -   -   -
/tld/subdir/subfile   /tmp/demo/subdir/subfile   1   1   -   -   -   -   -   -


$ dwcli stage query --session $session --configuration $configuration
path      backing-path    nss ->c ->q ->f <-c <-q <-f <-m
/.        -               1   5   -   -   -   -   -   -
/tld/     -               1   4   -   -   -   -   -   -
/dwfsfile /tmp/demo/filea 1   1   -   -   -   -   -   -
```

4.2 dwstat(1)

NAME

`dwstat` - Provides status information about DataWarp resources

SYNOPSIS

```
dwstat [-h]
dwstat [--all] [-b | -e | -E | -g | -G | -H | -k | -K | -m | -M |
   -p | -P | -t | -T | -y | -Y | -z | -Z]
   [--role ROLE]
   [RESOURCE [RESOURCE]...]
```

DESCRIPTION

The `dwstat` command provides status information about DataWarp resources in tabular format.

IMPORTANT: The `dws` module must be loaded to use this command.

```
$ module load dws
```

The `dwstat` commands accepts the following options:

-h | --help	Displays usage information.
--all	Used with `nodes` resource; displays all nodes. Default display includes nodes with `capacity>0` only.
-b	Displays output in bytes
-e	Displays output in IEC Exbibyte (EiB) units; for further information, see *Prefixes for Binary and Decimal Multiples* on page 59
-E	Displays output in SI Exabyte (EB) units
-g	Displays output in IEC gibibyte (GiB) units
-G	Displays output in SI kigabyte (GB) units

-H	Displays output in SI units (IEC is default)
-k	Displays output in IEC kibibyte (KiB) units
-K	Displays output in SI kilobyte (KB) units
-m	Displays output in IEC mebibyte (MiB) units
-M	Displays output in SI megabyte (MB) units
-p	Displays output in IEC Pebibyte (PiB) units
-P	Displays output in SI Petabyte (PB) units
--role *ROLE*	Requests a role outside of user's level
-t	Displays output in IEC Tebibyte (TiB) units
-T	Displays output in SI Terabyte (TB) units
-y	Displays output in IEC Yobibyte (YiB) units
-Y	Displays output in SI Yottabyte (YB) units
-z	Displays output in IEC Zebibyte (ZiB) units
-Z	Displays output in SI Zettabyte (ZB) units

Resources

The `dwstat` command accepts the following resources:

activations	Displays a table of current activations; a DataWarp activation is an object that represents an available instance configuration on a set of nodes.
all	Displays the tables for all resource types.
configurations	Displays a table of current configurations; a DataWarp configuration represents a specific way in which a DataWarp instance will be used.
fragments	Displays a table of current fragments; a DataWarp fragment is a subset of managed space on a DataWarp node.
instances	Displays a table of current instances; a DataWarp instance is a collection of DataWarp fragments, where no two fragments in the instance exist on the same node.
most	Displays tables for `pools`, `sessions`, `instances`, `configurations`, `registrations`, and `activations`
namespaces	Displays a table of current namespaces; a DataWarp namespace represents a partitioning of a DataWarp scratch configuration.
nodes	Displays a table of current nodes; a DataWarp node can host DataWarp capacity, have DataWarp configurations activated on it, or both. By default, displays nodes with `capacity>0` only.
pools	Displays a table of current pools; a DataWarp pool represents an aggregate DataWarp capacity. (Default output)
registrations	Displays a table of current registrations; a DataWarp registration represents a known use of a configuration by a session.
sessions	Displays a table of current sessions; a DataWarp session is an object used to map events between a client context and a DataWarp service context. A WLM typically creates a DataWarp session for each batch job that uses the DataWarp service.

EXAMPLE: dwstat pools

```
$ dwstat pools
    pool units quantity    free   gran
wlm_pool bytes        0       0   1GiB
   space bytes  7.12TiB 2.88TiB 128GiB
testpool bytes        0       0  16MiB
```

The column headings are defined as:

pool	Pool name
units	Pool units; currently only bytes are supported
quantity	Maximum configured space
free	Currently available space
gran	Granularity - pool stripe size

EXAMPLE: dwstat sessions

```
$ dwstat sessions
 sess state      token creator owner             created expiration nodes
  832 CA---  783000000  tester 12345 2015-09-08T16:20:36      never    20
  833 CA---  784100000  tester 12345 2015-09-08T16:21:36      never     1
  903 D---- 1875700000  tester 12345 2015-09-08T17:26:05      never     0
```

The column headings are defined as:

sess	Numeric session ID
state	Five-character code representing a session's state as follows (left to right): **1.** Goal: `C` = Create; `D` = Destroy **2.** Setup: `A` = Actualized; `-` = non-actualized **3.** Condition: `F` = Fuse blown (an error exists); `-` = fuse intact **4.** Status: `T` = Transitioning; `-` = idling or blocked from transitioning **5.** Spectrum: `M` = Mixed (goal delayed by registration); `-` = not delayed
token	Unique identifier (typically the batch job id) created by WLM
creator	Typically an identifier for the WLM software
owner	UID of job
created	Creation timestamp
expiration	*`date`* = expiration date; `never` = no expiration date
nodes	Number of nodes at session set up

EXAMPLE: dwstat instances

```
$ dwstat instances
 inst state sess  bytes nodes            created  expiration intact  label public confs
  753 CA---  832 128GiB     1 2015-09-08T16:20:36       never   true I832-0  false     1
  754 CA---  833 128GiB     1 2015-09-08T16:21:36       never   true I833-0  false     1
```

```
  807 D----   903 128GiB    1 2015-09-08T17:26:05       never  false I903-0  false     1
  808 CA---   904 128GiB    1 2015-09-08T17:26:08       never   true I904-0  false     1
  810 CA---   906 128GiB    1 2015-09-08T17:26:10       never   true I906-0  false     1
```

The column headings are defined as:

inst Numeric instance ID

state Five-character code representing an instance's state as follows (left-to-right):

1. Goal: `C` = Create; `D` = Destroy
2. Setup: `A` = Actualized; `-` = non-actualized
3. Condition: `F` = Fuse blown (an error exists); `-` = fuse intact
4. Status: `T` = Transitioning; `-` = idling or blocked from transitioning
5. Spectrum: `M` = Mixed (goal delayed by registration); `-` = not delayed

sess Numeric session ID

bytes Instance size

nodes Number of nodes on which this instance is active

created Creation timestamp

expiration *`date`* = expiration date; `never` = no expiration date

intact True, if `Goal=C` (create), and all fragments associated with this instance are themselves associated with a node

label User-defined label (name)

public `true` = shared resource (visible to all users)

confs Number of configurations to which an instance belongs

EXAMPLE: `dwstat configurations`

```
$ dwstat configurations
 conf state inst    type access_type activs
  715 CA---  753 scratch      stripe      1
  716 CA---  754 scratch      stripe      1
  759 D--T-  807 scratch      stripe      0
  760 CA---  808 scratch      stripe      1
```

The column headings are defined as:

conf Number configuration ID

state Five-character code representing a configuration's state as follows (left-to-right):

1. Goal: `C` = Create; `D` = Destroy
2. Setup: `A` = Actualized; `-` = non-actualized
3. Condition: `F` = Fuse blown (an error exists); `-` = fuse intact
4. Status: `T` = Transitioning; `-` = idling or blocked from transitioning
5. Spectrum: `M` = Mixed (goal delayed by registration); `-` = not delayed

inst Numeric instance ID

type	Configuration type - `scratch` or `cache`
access_type	Access mode - `stripe` or `private`
activs	Number of activations to which a configuration belongs

EXAMPLE: `dwstat registrations`

```
$ dwstat registrations
  reg state sess conf wait
  648 CA---  832  715 true
  649 CA---  833  716 true
  674 CA---  904  760 true
```

The column headings are defined as:

reg Numeric registration ID

state Five-character code representing a registration's state as follows (left-to-right):

1. Goal: `C` = Create; `D` = Destroy
2. Setup: `A` = Actualized; `-` = non-actualized
3. Condition: `F` = Fuse blown (an error exists); `-` = fuse intact
4. Status: `T` = Transitioning; `-` = idling or blocked from transitioning
5. Spectrum: `M` = Mixed (goal delayed by registration); `-` = not delayed

sess Numeric session ID

conf Numeric configuration ID

wait If `true`, then on registration teardown any data in the associated configuration will first finish asynchronous activities

EXAMPLE: `dwstat activations`

```
$ dwstat activations
activ state sess conf nodes      mount
  622 CA---  832  715    20  /tmp/tst1
  623 CA---  833  716     1  /tmp/tst2
  648 CA---  904  760     1  /tmp/tst3
  650 CA---  906  762     1  /tmp/tst4
```

The column headings are defined as:

activ Numeric activation ID

state Five-character code representing an activation's state as follows (left-to-right):

1. Goal: `C` = Create; `D` = Destroy
2. Setup: `A` = Actualized; `-` = non-actualized
3. Condition: `F` = Fuse blown (an error exists); `-` = fuse intact
4. Status: `T` = Transitioning; `-` = idling or blocked from transitioning
5. Spectrum: `M` = Mixed (goal delayed by registration); `-` = not delayed

sess	Numeric session ID
conf	Numeric configuration ID
nodes	Number of nodes on which an activation is present
mount	Mount point for the activation

EXAMPLE: `dwstat fragments`

```
$ dwstat fragments
 frag state inst capacity     node
  780  CA--  753   128GiB nid00066
  781  CA--  754   128GiB nid00069
  842  D---  807   128GiB nid00022
  843  CA--  808   128GiB nid00065
```

The column headings are defined as:

frag	Numeric fragment ID
state	Four-character code representing a fragment's state as follows (left-to-right): **1.** Goal: `C` = Create; `D` = Destroy **2.** Setup: `A` = Actualized; `-` = non-actualized **3.** Condition: `F` = Fuse blown (an error exists); `-` = fuse intact **4.** Status: `T` = Transitioning; `-` = idling or blocked from transitioning
inst	Numeric instance ID
capacity	Total capacity of a fragment
node	Hostname of node on which a fragment is located

EXAMPLE: `dwstat namespaces`

```
$ dwstat namespaces
   ns state conf frag span
  758  CA--  715  780    1
  759  CA--  716  781    1
  818  CA--  760  843    1
```

The column headings are defined as:

ns	Numeric namespace ID
state	Four-character code representing a namespace's state as follows (left-to-right): **1.** Goal: `C` = Create; `D` = Destroy **2.** Setup: `A` = Actualized; `-` = non-actualized **3.** Condition: `F` = Fuse blown (an error exists); `-` = fuse intact **4.** Status: `T` = Transitioning; `-` = idling or blocked from transitioning
conf	Numeric configuration ID
frag	Numeric fragment ID

span Number of fragments across which a namespace reads and writes

EXAMPLE: `dwstat nodes`

```
$ dwstat nodes
    node  pool online drain  gran   capacity insts activs
nid00022 space   true false  8MiB    3.64TiB     7      0
nid00065 space   true false 16MiB 1023.98GiB     7      0
nid00066 space   true false 16MiB 1023.98GiB     7      0
nid00069 space   true false 16MiB 1023.98GiB     7      0
nid00070 space   true false 16MiB 1023.98GiB     6      0
nid00004     -   true false     0           0     0      3
```

The column headings are defined as:

node	Node hostname
pool	Name of pool to which node is assigned
online	The node is available
drain	`true` = resource is draining
gran	Node granularity
capacity	Total capacity of a node
insts	Number of instances on a node
activs	Number of activations on a node

4.3 xtcheckssd(8)

NAME

`xtcheckssd` - Report SSD health

SYNOPSIS

```
xtcheckssd [-h]
xtcheckssd [-d [-i TIME]] [-p PATH] [-r ] [-s] [MountPoint]
```

DESCRIPTION

The `xtcheckssd` command queries the health of one or all SSDs (both FusionIO and NVMe). It is located in `/opt/cray/diag/default/bin`, and must be run as `root` on an SSD service node, either as a daemon or as a one-time command. `xtcheckssd` reports output to: the console (when not run as a daemon); the SMW, via the `/dev/console` log; and the CLE system log (`syslog`) via the RCA event `ec_rca_diag`.

OPTIONS

`xtcheckssd` accepts the following options:

-d

Run as a daemon (default 24-hour wakeup)

-h

Displays usage information

-i *TIME*

Used with `-d`, this option sets the reporting interval (*`TIME`*) and the interval for generating the `ec_rca_diag` event. *`TIME`* is defined in seconds (`s`), minutes (`m`), hours (`h`), or days (`d`). Default is `1d`; minimum is `1m`, and maximum is `7d`.

-p *PATH*

Specifies the path to the NVMe utilities

-r

Do not generate `ec_rca_diag` RCA event

-s

Silent mode; no console output. This has no affect when running as a daemon.

MountPoint

The SSD mount point; if not specified, reports on all SSDs

FILES

/dev/console	SMW console log
/what/is/the/path/to/syslog	CLE system log

RETURN VALUES

Normal operation:

```
PCIe slot#:SLOT,Name:VENDOR MODEL,SN:SERIAL#,Size:SIZEGB,Remaining life:RLIFE
%,Temperature:TEMP(c)
```

Abnormal operation:

```
Abnormal Operation - No SSds are installed:
 xtcheckssd: 3 No SSDs found
```

```
Abnormal Operation - NVMe Utility not found:
 xtcheckssd-error: 4 Path to NVMe-CLI User Utility does not exist
```

LIMITATIONS

When only a single SSD is attached to a node, `xtcheckssd` identifies and reports the health of the SSD but is unable to determine the front panel PCIe slot number.

EXAMPLES

Example 1: Report on all SSDs for a node:

```
nid00350:# /opt/cray/diag/default/bin/xtcheckssd
PCIe slot#:1,Name:INTEL SSDPECME040T4,SN:CVF8515300094P0DGN-1,Size:
4000GB,Remaining life:100%,Temperature:22(c)
```

```
PCIe slot#:1,Name:INTEL SSDPECME040T4,SN:CVF8515300094P0DGN-2,Size:
4000GB,Remaining life:100%,Temperature:24(c)
PCIe slot#:0,Name:INTEL SSDPECME040T4,SN:CVF85153001V4P0DGN-1,Size:
4000GB,Remaining life:100%,Temperature:22(c)
PCIe slot#:0,Name:INTEL SSDPECME040T4,SN:CVF85153001V4P0DGN-2,Size:
4000GB,Remaining life:100%,Temperature:24(c)
xtcheckssd: 0 Normal program termination
```

Example 2: Run `xtcheckssd` as a deamon generating a report every 24 hours:

```
DWnode:# /opt/cray/diag/default/bin/xtcheckssd -d
```

4.4 xtiossdflash(8)

NAME

`xtiossdflash` - Updates the firmware on Intel P3608 SSD cards

SYNOPSIS

`xtiossdflash [ -vfFh] [ -i F3608_FW_IMAGE ]` *target*

DESCRIPTION

The `xtiossdflash` command, which must be run by `root` on the boot node, updates the firmware on Intel P3608 SSD cards.

`xtiossdflash` accepts the following options:

-f	Flash a specified firmware
-F	Force a flash even if the drive is already flashed to the current version
-h	Display this help dialog
-i *path*	Specifies path to Intel P3608 flash image
-v	Display current firmware version

`xtiossdflash` accepts the following argument:

target May be a single node, a comma-separated list of nodes, of the keyword `all_service`, which includes all nodes with Intel P3608 SSD cards.

`xtiossdflash` compares the current flash version to the image flash file and flashes the device only if the two are different (up or down). This can be overridden by specifying the `-F` (force) flag.

Service node needs to be rebooted for the new firmware to be loaded .

EXAMPLES

To report the model and firmware version of SSDs:

```
boot:# xtiossdflash -v all_service
```

An example of a successful execution:

```
boot:# xtiossdflash -f -i /tmp/8DV10151_8B1B0130_signed.bin all_service
c0-0c0s5n2: <nvme_flash>: The firmware for /dev/nvme0 is up to date (8DV10151).
c0-0c0s5n2: <nvme_flash>: The firmware for /dev/nvme1 is up to date (8DV10151).
c0-0c0s5n2: <nvme_flash>: The firmware for /dev/nvme2 is up to date (8DV10151).
c0-0c0s5n2: <nvme_flash>: The firmware for /dev/nvme3 is up to date (8DV10151).
c0-0c0s3n1: <nvme_flash>: The firmware for /dev/nvme0 is up to date (8DV10151).
c0-0c0s3n2: <nvme_flash>: The firmware for /dev/nvme0 is up to date (8DV10151).
c0-0c0s5n1: <nvme_flash>: Flashing /dev/nvme0 using file /tmp/
8DV10151_8B1B0130_signed.bin
```

An example of an unsuccessful execution:

```
boot:# xtiossdflash -f -i /tmp/8DV10151_8B1B0130_signed.bin all_service
c0-0c0s5n1: <nvme_flash>: Flashing /dev/nvme0 using file /tmp/
8DV10151_8B1B0130_signed.bin
c0-0c0s5n1: NVME Admin command error:263
c0-0c0s5n1: <nvme_flash>: Firmware activation on /dev/nvme0 failed!
c0-0c0s5n1: <nvme_flash>: Flash failure detected. Exiting.
pdsh@boot: c0-0c0s5n1: ssh exited with exit code 1
c0-0c0s5n2: <nvme_flash>: The firmware for /dev/nvme0 is up to date (8DV10151).
```

4.5 xtssdconfig(8)

NAME

`xtssdconfig` - Displays SSD configuration information

SYNOPSIS

```
xtssdconfig [-h] [-v]
xtssdconfig [-j] [-m] [-t TIMEOUT] [id,[id,...]
```

DESCRIPTION

The `xtssdconfig` command runs `xthwinv` and parses the output to display SSD information.

OPTIONS

`xtssdconfig` accepts the following options:

-h|--help

Displays usage information

-j|--json

Format the output as JSON

-m|--mini

Used with `-j`, displays a brief version of the output

-t|--timeout *TIMEOUT*

Defines the response timeout (secs); default is the `xthwinv` default

-v|--VERSION

Displays tool version

id

Optional; one or more comma-separated IDs (cname). Valid cnames are partitions, cabinets, cages, or blades.

If a provided cname is the parent of components with SSDs, those child components are matched. For example, if nodes c0-0c0s1n3 and c0-0c0s7n0 have associated SSDs, then specifying the cname c0-0c0 matches both nodes because it is the parent of those nodes.

EXAMPLES

Report on all SSD nodes:

```
crayadm@smw:> xtssdconfig
   node_id     ssd_id     sub_id  bus device func size serial_num
-------------------------------------------------------------------------
c1-0c2s0n1  0x8086953 0x80863709 0x04   0x00  0x0      CVF85153001S4P0DGN-1
c1-0c2s0n1  0x8086953 0x80863709 0x05   0x00  0x0      CVF85153001S4P0DGN-2
c1-0c2s0n1  0x8086953 0x80863709 0x08   0x00  0x0      CVF85156006B4P0DGN-1
c1-0c2s0n1  0x8086953 0x80863709 0x09   0x00  0x0      CVF85156006B4P0DGN-2
c1-0c2s0n2  0x8086953 0x80863709 0x04   0x00  0x0      CVF85153000Z4P0DGN-1
c1-0c2s0n2  0x8086953 0x80863709 0x05   0x00  0x0      CVF85153000Z4P0DGN-2
c1-0c2s0n2  0x8086953 0x80863709 0x08   0x00  0x0      CVF85153001E4P0DGN-1
c1-0c2s0n2  0x8086953 0x80863709 0x09   0x00  0x0      CVF85153001E4P0DGN-2
```

5 DataWarp Administrator Tasks

5.1 Update DWS Configuration Files

There are three DataWarp Service (DWS) configuration files:

1. The scheduler configuration file: `/etc/opt/cray/dws/dwsd.yaml`
2. The manager daemon configuration file: `/etc/opt/cray/dws/dwmd.yaml`
3. The API gateway configuration file: `/etc/opt/cray/dws/dwrest.yaml`

I believe this procedure has to change for CLE6.0, doesn't it?

Use `xtopview` to modify any of these files from within the shared root. For the changes to take affect, exit `xtopview` and either restart or send `SIGHUP` to the corresponding daemon(s).

The DataWarp Scheduler Daemon (`dwsd`)

The `dwsd`, which runs on the sdb node, reads /`etc/opt/cray/dws/dwsd.yaml` at startup and when it receives the `SIGHUP` signal. For example, if a change is made to `dwsd.yaml`:

```
boot:# ssh sdb
sdb:# kill -HUP $(</var/opt/cray/dws/dwsd.pid)
sdb:# tail -5 /var/opt/cray/dws/log/dwsd.log
2015-09-17 14:15:26 ========== Event on fd 4
2015-09-17 14:15:26 Caught signal Hangup
2015-09-17 14:15:26 vvvvvvvvvv Configuration Delta Summary vvvvvvv
2015-09-17 14:15:26 log_mask: 0x7 -> 0x587
2015-09-17 14:15:26 ^^^^^^^^^^ End Configuration Delta Summary ^^^
```

The DataWarp Management Daemon (`dwmd`)

The `dwmd`, which runs on each SSD-endowed node, reads `/etc/opt/cray/dws/dwmd.yaml` at startup and when it receives the `SIGHUP` signal. For example, if a change is made to `dwmd.yaml`:

```
boot:# ssh nid00777
nid00777:# kill -HUP $(</var/opt/cray/dws/dwmd.pid)
nid00777:# tail -4 /var/opt/cray/dws/log/dwmd.log
2015-09-17 14:21:54 (31678) Caught signal Hangup
2015-09-17 14:21:54 (31678) vvvvvvvvvv Configuration Delta Summary vvvvvvv
2015-09-17 14:21:54 (31678) log_mask: 0xfffff -> 0xf
2015-09-17 14:21:54 (31678) ^^^^^^^^^^ End Configuration Delta Summary ^^^
```

On systems with many SSDs, it may be necessary to send `SIGHUP` to `dwmd` daemons on many nodes. The following command generates a file that contains identifiers that can be used with `pcmd` to perform the `SIGHUP` in parallel:

```
dwstat nodes | tail -n +2 | cut -d ' ' -f 1 | sed -e 's/[^0-9]//g' -e '/^$/d' |
head -c -1 | tr '\n' ',' >/tmp/dws_servers.nids
```

```
sdb:# module load dws
sdb:# dwstat nodes | tail -n +2 | cut -d ' ' -f 1 | sed -e 's/[^0-9]//g' -e '/^$/
d' | head -c -1 | tr '\n' ',' >/tmp/dws_servers.nids
sdb:# module load nodehealth
sdb:# pcmd -f /tmp/dws_servers.nids 'kill -HUP $(</var/opt/cray/dws/dwmd.pid)'
Reply (complete) from nid00065 exit code: 0
Reply (complete) from nid00066 exit code: 0
Reply (complete) from nid00069 exit code: 0
Reply (complete) from nid00070 exit code: 0
```

The DataWarp RESTful Service (`dwrest`)

The `dwrest` component, which typically runs on a single login node, reads `/etc/opt/cray/dws/dwrest.yaml` at startup and when it receives the `SIGHUP` signal. The `dwrest` component is displayed in `ps` ouput as the `gunicorn` process. Therefore, if a change is made to `dwrest.yaml`:

```
boot:# ssh login
login:# kill -HUP $(</var/opt/cray/dws/gunicorn.pid)
```

5.2 DataWarp with DWS: Create a Storage Pool

Prerequisites

- Access to DataWarp administrator privileges (`root`, `crayadm`, or other UID defined in `CLEinstall.conf`) is available:
  - Consult the `admin` entry of the shared root file `/etc/opt/cray/dws/dwrest.yaml` for a list of DataWarp administrator UIDs.
- Recommended: Completion of *Initialize an SSD* on page 45.

About this task

A storage pool groups nodes with storage together such that requests for space made against the pool are fufilled from the nodes associated with the pool with a common allocation granularity. Pools have either byte or node allocation granularity (*`pool_AG`*). This release of DWS only supports byte allocation granularity. There are tradeoffs in picking allocation granularities too small or too large.

TIP:

Use `dwpoolhelp`: Determining an optimal pool allocation granularity for a system is a function of several factors, including the number of SSD nodes, the number of SSD cards per node, the size of the SSDs, as well as software requirements, limitiations, and bugs. Therefore, the best value is site specific and likely to change over time. For this reason, Cray developed the `dwpoolhelp` command to automate this process.

Because the command is not yet released, the source is provided in *The dwpoolhelp Command Source Code* on page 38.

The `dwpoolhelp` command calculates and displays pool allocation granularity values for a range of node granularity units along with waste per node and waste per pool values in bytes. The `dwpoolhelp` command accepts the following options:

`-c` *`capacity`*

Number of bytes available on each node; default = 6401262878720 bytes

`-h`

Displays usage information

`-g` *`granularity`*

Allocation granularity of each node; default = 16777216 bytes

`-m` *`stripes`*

Maximum number of DVS stripes

`-n` *`nodes`*

Number of nodes in the pool

`-s`

Suggests only the smallest viable granularity

`-v`

Displays version information for `dwpoolhelp`

Sites intending to use `dwpoolhelp` can jump to step *1* on page 37 of the procedure.

Not using `dwpoolhelp`: If a site chooses not to build and use the `dwpoolhelp` command to determine the pool allocation granularity, the following guidelines are provided as important considerations when creating a storage pool:

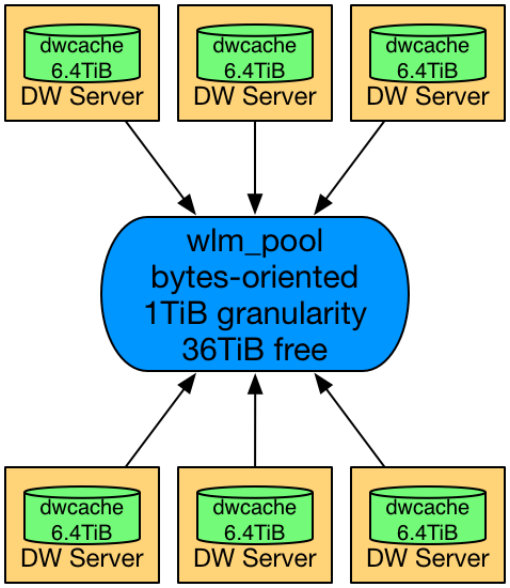
1. The byte-oriented allocation granularity for a pool must be at least 16MiB.
2. Each node's volume group (`dwcache`, configured in *Initialize an SSD* on page 45) has a Physical Extent size (*`PE_size`*) and Physical Volume count (*`PV_count`*). The default *`PE_size`* is 4MiB, and *`PV_count`* is equal to the number of Physical Volumes specified during volume group creation. DWS places the following restriction on nodes associated with a pool:
   - A node can only be associated with a storage pool if the node's granularity (*`PE_size`* * *`PV_count`*) is a factor of the pool's allocation granularity (*`pool_AG`*). The `dwstat nodes` command lists the node's granularity in the `gran` column.
3. The more nodes in the pool, the higher the granularity.
4. Ideally, a pool's allocation granularity is defined as a factor of the aggregate space of each node within the pool; otherwise, some space is not usable and, therefore, is wasted.

The most important considerations are #1 and #2. On all Cray systems, picking a pool granularity of at least 16MiB (16,777,216 bytes) and is a multiple of 16MiB (16,777,216, 33,554,432, 50,331,648, ...) will define a functioning, but possibly sub-optimal, configuration. The following recommendation table does not take #3 and #4 into consideration but will be a good starting point for all Cray system configurations:

Table 2. Nodes

SSD Nodes in Pool	Granularity (bytes)
1	16,777,216
<20	214,748,364,800
<100	429,496,729,600
<200	858,993,459,200
>=200	1,073,741,824,000

The following diagram shows six different DataWarp nodes belonging to a storage pool `wlm_pool` with a 1TiB allocation granularity. Each DataWarp node has 6.4TiB of space, which means that 0.4TiB are wasted per node because only 6 allocation granularities fit on any one node.



Sites not using `dwpoolhelp` should jump to step *3* on page 38 of the procedure.

Procedure

1. Build the `dwpoolhelp` command from the source in *The dwpoolhelp Command Source Code* on page 38.

```
$ gcc -o dwpoolhelp dwpoolhelp.c -DPACKAGE_VERSION=\"pubs-copy\"
```

2. Execute `dwpoolhelp` with site-specific values.

   For example:

```
$ ./dwpoolhelp -n 10
== Starting Values ==
Number of nodes: 10
Node capacity: 6401262878720
Allocation granularity on nodes: 16777216
```

```
== Calculating maximum granules per node ==
Max number of granules in an instance while still being able to access all
capacity is 4096
floor(max_stripes / nodes) -> floor(4096 / 10) = 409
Bug 830114 limits to maximum of 35 granules per node!
Maximum granules per node: 35

== Optimal pool granularities per granules per node ==
Gran / node     Pool granularity  Waste per node       Waste per pool
          1        6401262878720               0                    0
          2        3200623050752        16777216            167772160
          3        2133743108096        33554432            335544320
          4        1600311525376        16777216            167772160
          5        1280252575744               0                    0
...
         30         213372633088        83886080            838860800
         31         206477197312       469762048           4697620480
         32         200034746368       150994944           1509949440
         33         193961394176       536870912           5368709120
         34         188257140736       520093696           5200936960
         35         182888431616       167772160           1677721600
```

3. Log in to a booted CLE service node as a DWS administrator.

4. Load the DataWarp Service module.

```
login:# module load dws
```

5. Create a storage pool.

```
login:# dwcli create pool --name pool_name --granularity alloc_gran
```

Example 1: to create a pool `wlm_pool` (Cray recommended name) with an allocation granularity of 200 GiB:

```
login:# dwcli create pool --name wlm_pool --granularity 214748364800
created pool id wlm_pool
```

Example 2: to create a pool `wlm_pool` with 30 granularities per node (using the `dwpoolhelp` output from above):

```
login:# dwcli create pool --name wlm_pool2 --granularity 213372633088
created pool id wlm_pool2
```

6. Verify the pool was created.

```
login:# dwstat pools
     pool  unit quantity    free       gran
 wlm_pool bytes        0       0     200GiB
wlm_pool2 bytes        0       0  198.72GiB
```

5.3 The dwpoolhelp Command Source Code

The `dwpoolhelp` command calculates and displays pool allocation granularity values for a range of node granularity units along with waste per node and waste per pool values in bytes. This command is not included in

any current releases, and source is provided here to aide administrators when creating DataWarp storage pools (see *DataWarp with DWS: Create a Storage Pool* on page 35).

To build the command, execute the following:

```
$ gcc -o dwpoolhelp dwpoolhelp.c -DPACKAGE_VERSION=\"pubs-copy\"
```

```
/*
 * (c) 2015 Cray Inc.  All Rights Reserved.  Unpublished Proprietary
 * Information.  This unpublished work is protected to trade secret,
 * copyright and other laws.  Except as permitted by contract or
 * express written permission of Cray Inc., no part of this work or
 * its content may be used, reproduced or disclosed in any form.
 */

#ifdef HAVE_CONFIG_H
# include <config.h>
#endif

#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <inttypes.h>
#include <limits.h>
#include <getopt.h>
#include <string.h>
#include <errno.h>
#include <stdbool.h>

#define GETOPT_OPTS "hvsn:c:g:m:"
enum {
    OPTION_HELP = 'h',
    OPTION_VERSION = 'v',
    OPTION_NODES = 'n',
    OPTION_NODE_CAPACITY = 'c',
    OPTION_NODE_GRANULARITY = 'g',
    OPTION_SMALLEST = 's',
    OPTION_MAX_STRIPES = 'm'
};

static const struct option long_opts[] = {
    {"nodes", required_argument, NULL, OPTION_NODES},
    {"node-capacity", required_argument, NULL, OPTION_NODE_CAPACITY},
    {"node-granularity", required_argument, NULL, OPTION_NODE_GRANULARITY},
    {"max-stripes", required_argument, NULL, OPTION_MAX_STRIPES},
    {"smallest", no_argument, NULL, OPTION_SMALLEST},
    {"help", no_argument, NULL, OPTION_HELP},
    {"version", no_argument, NULL, OPTION_VERSION},
    {0, 0, 0, 0}
};

static void print_table(int nodes, int gran_per_node,
            int64_t capacity, int64_t node_gran);
static void print_smallest(int nodes, int gran_per_node,
               int64_t capacity, int64_t node_gran);
static int parse_args(int argc, char *const* argv, int *ret_nodes,
              int64_t *ret_capacity, int64_t *ret_node_gran,
              bool *ret_smallest, int *ret_max_stripes);
static void print_version(FILE *fp, const char *cmd_name);
static void print_usage(FILE *fp, const char *cmd_name);
```

```
static int strtoi_helper(const char *str, int *val);
static int strtoll_helper(const char *str, int64_t *val);

int
main(int argc, char *const*argv)
{
    int ret = EXIT_FAILURE;
    int err;
    int max_stripes = 4096; /* Hard-coded DVS limit */
    bool smallest = false;
    int nodes = 300;
    int64_t capacity = 6401262878720;
    int64_t node_gran = 16777216;
    int64_t node_alloc_gran;
    int gran_per_node;

    do {
        if ((err = parse_args(argc, argv,
                    &nodes, &capacity, &node_gran,
                    &smallest, &max_stripes))) {
            ret = err;
            break;
        }
        /* Output givens */
        printf("== Starting Values ==\n");
        printf("Number of nodes: %d\n", nodes);
        printf("Node capacity: %"PRId64"\n", capacity);
        printf("Allocation granularity on nodes: %"PRId64"\n\n",
               node_gran);

        /* Granules per node */
        printf("== Calculating maximum granules per node ==\n");
        printf("Max number of granules in an instance while still "
               "being able to access all capacity is %d\n",
               max_stripes);
        gran_per_node = max_stripes / nodes;
        printf("floor(max_stripes / nodes) -> floor(%d / %d) = %d\n",
               max_stripes, nodes, gran_per_node);
        if (gran_per_node > 35) {
            printf("Bug 830114 limits to maximum of 35 granules per node!\n");
            gran_per_node = 35;
        } else {
            printf("No further downward adjustment needed\n");
        }
        printf("Maximum granules per node: %d\n", gran_per_node);
        if (node_gran < 16777216) {
            /* Handle XFS minimum size */
            node_alloc_gran = ((16777216 + (node_gran - 1)) / node_gran) *
node_gran;
            printf("Using %"PRId64" bytes for actual allocation "
                   "granularity on nodes to satisfy "
                   "XFS requirements\n", node_alloc_gran);
        } else {
            node_alloc_gran = node_gran;
        }
        printf("\n");

        if (smallest) {
            print_smallest(nodes, gran_per_node, capacity,
                        node_alloc_gran);
        } else {
```

```
            print_table(nodes, gran_per_node, capacity,
                    node_alloc_gran);
        }

        ret = EXIT_SUCCESS;
    } while (0);

    return ret;
}

static void
print_table(int nodes, int gran_per_node, int64_t capacity, int64_t node_gran)
{
    int i;
    int64_t granularity;

    /* Results table */
    printf("== Optimal pool granularities per granules per node ==\n");
    printf("%11s %20s %15s %20s\n",
           "Gran / node", "Pool granularity",
           "Waste per node", "Waste per pool");
    for (i = 1; i <= gran_per_node; i++) {
        granularity = capacity / i;
        granularity -= granularity % node_gran;
        printf("%11d %20"PRId64" %15"PRId64" %20"PRId64"\n",
               i, granularity, capacity % granularity,
               capacity % granularity * nodes);
    }
    return;
}

static void
print_smallest(int nodes, int gran_per_node, int64_t capacity, int64_t node_gran)
{
    int64_t granularity;

    /* Granule size */
    printf("== Calculating granule size that wastes the least amount of space ==
\n");
    granularity = capacity / gran_per_node;
    printf("Starting point for granularity is "
           "floor(capacity / gran_per_node) "
           "-> %"PRId64" / %d = %"PRId64"\n",
           capacity, gran_per_node, granularity);
    if (granularity % node_gran) {
        printf("Adjusting granularity downward to be a multiple "
               "of the node granularity\n");
        printf("granularity - granularity %% node_gran "
               "-> %"PRId64" - %"PRId64" %% %"PRId64" "
               "= %"PRId64"\n",
               granularity, granularity, node_gran,
               granularity - granularity % node_gran);
        granularity -= granularity % node_gran;
    } else {
        printf("Starting point is a multiple of the "
               "node granularity (%"PRId64")\n", node_gran);
    }

    /* Results */
    printf("\n== Results ==\n");
    printf("RECOMMENDED POOL GRANULARITY: %"PRId64"\n",
```

```
            granularity);
    printf("BYTES LOST PER NODE: %"PRId64"\n",
            capacity % granularity);
    printf("BYTES LOST ACROSS %d NODES: %"PRId64"\n",
            nodes, capacity % granularity * nodes);

    return;
}

static int
parse_args(int argc, char *const* argv, int *ret_nodes, int64_t *ret_capacity,
       int64_t *ret_node_gran, bool *ret_smallest, int *ret_max_stripes)
{
    const char *prog_name = strrchr(argv[0], '/');
    int opt = -1;

    if (prog_name == NULL) {
        prog_name = argv[0];
    } else {
        prog_name += 1;
    }

    if (argc > 1) {
        while ((opt = getopt_long(argc, argv, GETOPT_OPTS, long_opts, NULL))
               != EOF) {
            switch (opt) {
            case OPTION_HELP:
                print_usage(stdout, prog_name);
                exit(EXIT_SUCCESS);
            case OPTION_VERSION:
                print_version(stdout, prog_name);
                exit(EXIT_SUCCESS);
            case OPTION_NODES:
                if (strtoi_helper(optarg, ret_nodes)) {
                    exit(EXIT_FAILURE);
                }
                if (*ret_nodes < 1) {
                    fprintf(stderr, "Must supply at least "
                        "one node\n");
                    exit(EXIT_FAILURE);
                }
                break;
            case OPTION_MAX_STRIPES:
                if (strtoi_helper(optarg, ret_max_stripes)) {
                    exit(EXIT_FAILURE);
                }
                if (*ret_max_stripes < 1) {
                    fprintf(stderr, "Must supply at least "
                        "one stripe\n");
                    exit(EXIT_FAILURE);
                }
                break;
            case OPTION_NODE_CAPACITY:
                if (strtoll_helper(optarg, ret_capacity)) {
                    exit(EXIT_FAILURE);
                }
                if (*ret_capacity < 0) {
                    fprintf(stderr, "Must have at least "
                        "16777216 bytes of capacity\n");
                    exit(EXIT_FAILURE);
                }
```

```
                break;
            case OPTION_NODE_GRANULARITY:
                if (strtoll_helper(optarg, ret_node_gran)) {
                    exit(EXIT_FAILURE);
                }
                if (*ret_node_gran < 1) {
                    fprintf(stderr, "Node granularity must "
                        "be at least 1 byte\n");
                    exit(EXIT_FAILURE);
                }
                break;
            case OPTION_SMALLEST:
                *ret_smallest = true;
                break;
            default:
                print_usage(stderr, prog_name);
                exit(EXIT_FAILURE);
            }
        }
        if (*ret_capacity < *ret_node_gran) {
            fprintf(stderr, "Node capacity must be larger than "
                "node granularity\n");
            exit(EXIT_FAILURE);
        }
        if (*ret_nodes > *ret_max_stripes) {
            fprintf(stderr, "Nodes most not exceed max stripes\n");
            exit(EXIT_FAILURE);
        }
    }

    return 0;
}

static void
print_usage(FILE *fp, const char *cmd_name)
{
    fprintf(fp, "Usage: %s [OPTIONS]\n",
        cmd_name);
    fprintf(fp,
        " -h,   --help     Print this help message and exit\n"
        " -v,   --version  Print %s version information and exit\n"
        " -s,   --smallest Suggest only the smallest viable granularity\n"
        " -n N, --nodes=N  Number of nodes that will be in the pool\n"
        " -c C, --node-capacity=C    Number of bytes available on each node\n"
        " -g G, --node-granularity=G Allocation granularity of each node\n"
        " -m M, --max-stripes=M      Maximum number of DVS stripes\n",
        cmd_name);
}

static void
print_version(FILE *fp, const char *cmd_name)
{
    fprintf(fp, "%s (DWS) Version %s\n", cmd_name, PACKAGE_VERSION);
#ifdef EXTRA_BUILD_INFO
    fprintf(fp, "%s\n", EXTRA_BUILD_INFO);
#endif
}

static int
strtoi_helper(const char *str, int *val)
{
```

```
    char *endp;
    long strval;

    if (str == NULL) {
        fprintf(stderr, "NULL token\n");
        return -1;
    }

    errno = 0;
    strval = strtol(str, &endp, 0);

    if (errno == EINVAL) {
        /* Shouldn't ever see this... */
        fprintf(stderr, "Invalid base\n");
        return -1;
    } else if (errno == ERANGE || strval < INT_MIN || strval > INT_MAX) {
        fprintf(stderr, "'%s' not in suitable range for integers\n", str);
        return -1;
    } else if (endp == str) {
        fprintf(stderr, "'%s' is not an integer\n", str);
        return -1;
    } else if (*endp != 0) {
        fprintf(stderr, "'%s' has trailing non-integer junk\n", str);
        return -1;
    }

    *val = strval;
    return 0;
}

static int
strtoll_helper(const char *str, int64_t *val)
{
    char *endp;
    int64_t strval;

    if (str == NULL) {
        fprintf(stderr, "NULL token\n");
        return -1;
    }

    errno = 0;
    strval = strtoll(str, &endp, 0);

    if (errno == EINVAL) {
        /* Shouldn't ever see this... */
        fprintf(stderr, "Invalid base\n");
        return -1;
    } else if (errno == ERANGE || strval < INT64_MIN || strval > INT64_MAX) {
        fprintf(stderr, "'%s' not in suitable range for int64_t\n", str);
        return -1;
    } else if (endp == str) {
        fprintf(stderr, "'%s' is not an integer\n", str);
        return -1;
    } else if (*endp != 0) {
        fprintf(stderr, "'%s' has trailing non-integer junk\n", str);
        return -1;
    }

    *val = strval;
```

```
    return 0;
}
```

5.4 Initialize an SSD

Prerequisites

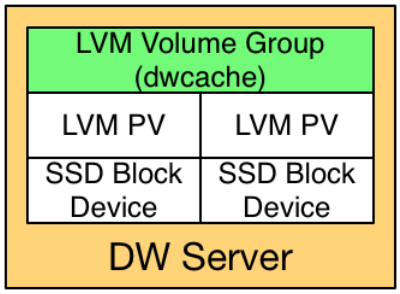
- Ability to log in as `root`

About this task

During the CLE installation process, the system administrator selects SSD-endowed nodes whose space the DWS will manage. This step ensures that the correct DWS daemon, `dwmd`, is started at boot time on these nodes. It **does not** prepare the SSDs for use with the DWS; this is performed manually using the following instructions.

After CLE boots, the following one-time manual device configuration must be performed **for each node** specified in `datawarp_manager_nodes` in `CLEinstall.conf`.

The diagram below shows how the logical volume manager (LVM) volume group `dwcache` is constructed on each DW node. In this diagram, two SSD block devices have been converted to LVM physical devices with the `pvcreate` command. These two LVM physical volumes were combined into the LVM volume group `dwcache` with the `vgcreate` command.



> **TIP:** Throughout these procedures, units of bytes are described using the binary prefixes defined by the International Electrotechnical Commission (IEC). For further information, see *Prefixes for Binary and Decimal Multiples* on page 59.

Procedure

1. Log in to an SSD-endowed node as `root`.

   This example uses `nid00349`.

2. Identify the SSD block devices.

```
nid00349:~ # lsblk
NAME    MAJ:MIN RM   SIZE RO MOUNTPOINT
nvme0n1 254:0    0   1.8T  0
nvme1n1 254:64   0   1.8T  0
```

3. Clean up existing uses of the SSDs (required only if the SSDs were previously used).

   This may include:

   - Unmounting any file systems on the SSDs
   - Stopping swap services
   - Using LVM to remove the SSDs from an existing volume group
   - Removing any existing partitioning scheme on the SSDs with:

     ```
     # dd if=/dev/zero of=phys_vol bs=512 count=1
     ```

     WARNING: This operation destroys any existing data on an SSD.

4. Initialize each physical device for later use by LVM. Note that Cray currently sells systems with 1, 2, or 4 physical devices on a node.

   WARNING: This operation destroys any existing data on an SSD. Back up any existing data before proceeding.

   ```
   nid00349:# pvcreate phys_vol [phys_vol...]
   ```

   For example:

   ```
   nid00349:# pvcreate /dev/nvme0n1 /dev/nvme1n1 /dev/nvme2n1 /dev/nvme3n1
   Physical volume "/dev/nvme0n1" successfully created
   Physical volume "/dev/nvme1n1" successfully created
   Physical volume "/dev/nvme2n1" successfully created
   Physical volume "/dev/nvme3n1" successfully created
   ```

5. Create an LVM volume group called `dwcache` that uses these physical devices.

   Requirements for the LVM physical volumes specified are:

   - Any number of physical devices may be specified.
   - Each physical volume specified **must be** the exact same size.
     - To verify physical volume size, execute the command: `pvs --units b` and examine the `PSize` column of the output.

   ```
   nid00349:# vgcreate dwcache phys_vol [phys_vol...]
   ```

   ```
   nid00349:# vgcreate dwcache /dev/nvme0n1 /dev/nvme1n1 /dev/nvme2n1 /dev/nvme3n1
   Volume group "dwcache" successfully created
   ```

6. Restart the `dwmd` service.

   For example:

   ```
   nid00349:# service dwmd restart
   Shutting down
   dwmd                                                       done
   Starting dwmd 2015-08-06 14:13:55 (10924) Beginning dwmd initialization
   2015-08-06 14:13:55 (10924) Daemonizing...
   2015-08-06 14:13:55 (10925) Redirecting stdout and stderr to /var//opt/cray/dws/
   log/dwmd.log. If this step fails, there's no way to log an error.
   ```

7. Verify that DWS recognizes the node with storage.

```
nid00349:# module load dws
nid00349:# dwstat nodes
    node  pool online drain  gran capacity insts activs
nid00349    -    true false  8MiB  3.64TiB     0      0
```

5.5 Assign a Node to a Storage Pool

Prerequisites

- At least one storage pool exists, see *DataWarp with DWS: Create a Storage Pool* on page 35.
- At least one SSD is initialized for use with the DWS, see *Initialize an SSD* on page 45.
- Access to DataWarp administrator privileges (`root`, `crayadm`, or other UID defined in `CLEinstall.conf`) is available.

About this task

Follow this procedure to associate an SSD-endowed node with an existing storage pool.

Procedure

1. Log in to a booted CLE service node as a DWS administrator.
2. Load the DataWarp Service module.

   ```
   login:# module load dws
   ```

3. Associate an SSD-endowed node with a storage pool.

   ```
   login:# dwcli update node --name hostname --pool pool_name
   ```

   For example, to associate a node (hostname `nid00349`) with a storage pool called `wlm_pool`:

   ```
   login:# dwcli update node --name nid00349 --pool wlm_pool
   ```

   The association may fail. If it does, ensure that the pool exists (`dwstat pools`) and that the node's granularity (`dwstat nodes -b`) is a factor of the pool's granularity (`dwstat pools -b`).

4. Verify the node is associated with the pool.

   ```
   login:# dwstat pools nodes
       pool units quantity    free    gran
   wlm_pool bytes  3.64TiB 3.64TiB  910GiB

       node     pool online  drain  gran   capacity insts activs
   nid00349 wlm_pool   true  false  8MiB    3.64TiB     0      0
   ```

5.6 Verify the DataWarp Configuration

Prerequisites

- At least one storage node is assigned to a storage pool, see *Assign a Node to a Storage Pool* on page 47.
- Access to DataWarp administrator privileges (`root`, `crayadm`, or other UID defined in CLEinstall.conf) is available.

About this task

There are a few ways to verify that the DataWarp configuration is as desired.

> **TIP:** Throughout these procedures, units of bytes are described using the binary prefixes defined by the International Electrotechnical Commission (IEC). For further information, see *Prefixes for Binary and Decimal Multiples* on page 59.

Procedure

1. Log in to a booted service node and load the DataWarp Service module.

```
login:# module load dws
```

2. Request status information about DataWarp resources.

```
login:# dwstat pools nodes
   pool units quantity    free    gran
  space bytes   3.5TiB 3.38TiB  128GiB

    node  pool online drain  gran   capacity  insts  activs
nid00065 space   true false 16MiB 1023.98GiB      1       0
nid00066 space   true false 16MiB 1023.98GiB      0       0
nid00070 space   true false 16MiB 1023.98GiB      0       0
nid00069 space   true false 16MiB 1023.98GiB      0       0
nid00022     -   true false  8MiB    3.64TiB      0       0
nid00004     -   true false     0          0      0       0
nid00005     -   true false     0          0      0       0
```

3. Check the following combinations for each row.
   - If **pool** is – and **capacity** ≠ `0`: This is a server node that has not yet been associated with a storage pool. See *Assign a Node to a Storage Pool* on page 47.
   - If **pool** is – and **capacity** is `0`: This is a non-server node (e.g., client/compute) and does not need to be associated with a storage pool.
   - If **pool** is *`something`* and **capacity** ≠ `0`: This is a server node that belongs to the pool called <something>.
   - If **pool** is *`something`* and **capacity** is `0`: This is a non-server node that belongs to a pool. Since the non-server node contributes no space to the pool, this association is not necessary but harmless.

This completes the process to configure DataWarp with DWS as outlined in *DataWarp with DWS: Post-boot Configuration*. Refer to the site-specific Workload Manager documentation for further configuration steps to integrate the WLM with Cray DataWarp.

5.7 Enable the Node Health Checker DataWarp Test

Prerequisites

- Ability to log in as `root`

About this task

The Node Health Checker (NHC) is automatically invoked by ALPS upon the termination of an application. ALPS passes a list of compute nodes associated with the terminated application to NHC. NHC performs specified tests to determine if compute nodes allocated to the application are healthy enough to support running subsequent applications. The `DataWarp` test is a plugin script to check that any reservation-affiliated DataWarp mount points have been removed. The plugin can only detect a problem once the last reservation on a node completes.

The configuration file that controls NHC behavior after a job has terminated is `/etc/opt/cray/nodehealth/nodehealth.conf`, located in the shared root. The CLE installation and upgrade processes automatically install this file and enable NHC software. By default, the `DataWarp` test is disabled.

For further information about NHC, see the `intro_NHC(8)` man page and *CLE XC™ System Administration Guide (S-2393)*.

Procedure

1. Log on to the boot node and invoke `xtopview`.

   ```
   smw# ssh root@boot
   boot:# xtopview
   ```

2. Edit the NHC configuration file on the shared root.

   ```
   default/:/# vi /etc/opt/cray/nodehealth/nodehealth.conf
   ```

3. Search for the DataWarp test entry `datawarp.sh`.

4. Enable the test by uncommenting the entire `[plugin]` entry.

   ```
   [Plugin]
   Command: datawarp.sh
   Action: Admindown
   WarnTime: 30
   Timeout: 360
   RestartDelay: 65
   Uid: 0
   Gid: 0
   Sets: Reservation
   ```

   For information regarding the standard variables used with the DataWarp test, see *CLE XC™ System Administration Guide (S-2393)*.

5. Save the changes and exit back to the SMW. Changes made to the NHC configuration file are reflected in the behavior of NHC the next time that it runs.

5.8 Manage Log Files

The DataWarp scheduler daemon (`dwsd`), manager daemon (`dwmd`), and RESTful service (`dwrest`) write to log files within `/var/opt/cray/dws/log`. Cray recommends using `logrotate` to control the size of these log files by activating it on the following:

- sdb node
- `dw_api_gateway` node
- all nodes in the `datawarp_manager_nodes` list (see *DataWarp with DWS: Set CLEinstall.conf Parameters for the DataWarp Service*)

Normally, `logrotate` is run as a daily cron job. For further information, see the `logrotate(8)` and `cron(8)` man pages, CLE Installation and Configuration Guide (S-2444), and the *Use `logrotate` to Control File Size* procedure in Cray XC Native Slurm Installation Guide (S-2538).

5.9 Drain a Storage Node

About this task

After an administrator assigns a node to a pool, any capacity on the node may be used when satisfying instance requests. There are times when a site does not want to allow new instances to be placed on a node and also does not want to disassociate the node from a pool. The `drain` attribute on a node controls this behavior. If a node is in a drain state, the DWS will not place new instances on the node and will also remove that node's free capacity contribution to a pool. The `dwstat nodes pools` command displays this information.

Procedure

1. Check the node and pool information.

```
crayadm@sys:> dwstat nodes pools
     node pool online drain  gran   capacity insts activs
nid00022 space   true false  8MiB    3.64TiB     0      0

 pool units quantity    free   gran
space bytes  7.12TiB 7.12TiB 128GiB
```

2. Drain the storage node.

```
smw:# dwcli update node --name hostname --drain
```

   where *`hostname`* is the hostname of the node to be drained

   For example:

```
smw:# dwcli update node --name nid00022 --drain
smw:# dwstat nodes pools
     node pool online drain  gran   capacity insts activs
nid00022 space   true true   8MiB    3.64TiB     0      0

 pool units quantity    free   gran
space bytes  7.12TiB  3.5TiB 128GiB
```

3. (Optional) If shutting down a node after draining it, wait for existing instances to be removed from the node. The `dwstat nodes` command displays the number of instances present in the `inst` column; `0` indicates no instances are present. In a healthy system, instances are removed over time as batch jobs complete. If it takes longer than expected, or to clean up the node more quickly, identify the fragments (pieces of instances) on the node by consulting the `node` column output of the `dwstat fragments` command and then finding the corresponding instance by looking at the `inst` column output:

```
smw:# dwstat fragments
frag state inst  capacity     node
 102  CA--   47 745.19GiB nid00022
```

4. (Optional) Remove that instance.

```
smw:# dwcli rm instance --id 47
```

   Persistent DataWarp instances, which have a lifetime that may span multiple batch jobs, must also be removed, either through a WLM-specific command or with `dwcli`.

5. When the node is fit for being used by the DWS again, unset the drain, thereby allowing the DWS to place new instances on the node.

```
smw:# dwcli update node --name nid00022 --no-drain
    node  pool online drain  gran   capacity insts activs
nid00022 space   true false  8MiB    3.64TiB     0      0

 pool units quantity    free   gran
space bytes  7.12TiB 7.12TiB 128GiB
```

5.10 Replace a Blown Fuse

After a workload manager sends DataWarp requests to the DWS, the DWS begins preparing the SSDs and compute nodes for the corresponding batch job. When things are working well, this process is quick and does not require admin intervention. The `dwstat` command reports `CA---` or `CA--` in the `state` column for all objects associated with the batch job. See *dwstat(1)* on page 23 for a description of the State column codes. If the DWS encounters difficulty creating or destroying an object, it retries a configurable number of times (defined in `dwsd.yaml`, see *Update DWS Configuration Files* on page 34) but eventually stops trying. To convey that the retry threshold has been exceeded, the DWS borrows terminology from another domain and reports that the object's *fuse* is blown. The `dwstat` command reports this as an `F` in the 3rd hyphen position of the `state` column. For example, `C-F--` as in the following `dwstat activations` output:

```
% dwstat activations
activ state sess conf nodes
    2 C-F--    5   11     1
```

When dwstat reports that an object's fuse is blown, it likely indicates a serious error that needs investigating by a system administrator. Clues as to what broke and why may be found in either the scheduler's log file (`dwsd`) or in the manager dameon's log files (the various `dwmd`, one per SSD-endowed node).

When the issue is understood and resolved, use the `dwcli` command to replace the blown fuse associated with the object, and thereby inform the DWS to retry the operations associated with the failed object. For example, continuing with the above failed activation:

```
% dwcli update activation --id 2 --replace-fuse
```

Use `dwstat` to find the status of the object again. Fuses are replaceable as many times as necessary.

5.11 Deconfigure DataWarp

Prerequisites

- Ability to log in as `root`
- The system is not running

About this task

Follow this procedure to remove the DataWarp configuration from a system.

Procedure

1. Log on to the SMW as `root`.
2. Disable DataWarp:
   a. Edit `CLEinstall.conf`.
   b. Set `datawarp=no`.
   c. Make a note of the SSD nodes listed in `datawarp_manager_nodes` as these are needed later in the procedure.
3. Execute `CLEinstall` to reconfigure.

   ```
   smw:# CLEinstall --reconfigure
   ```
4. Reboot the system.
5. Log on to an SSD-endowed node as `root`.

   This example uses `nid00349`.
6. Remove the data.
   a. Remove the LVM volume group.

      ```
      nid00349:# vgremove dwcache
      ```

A confirmation prompt may appear:

```
Do you really want to remove volume group "dwcache" containing 1 logical
volumes? [y/n]:
```

b. Answer `yes`.

c. Identify the SSD block devices.

```
nid00349:# pvs
  PV           VG      Fmt  Attr PSize PFree
  /dev/nvme0n1 dwcache lvm2 a--  1.46t 1.46t
  /dev/nvme1n1 dwcache lvm2 a--  1.46t 1.46t
  /dev/nvme2n1 dwcache lvm2 a--  1.46t 1.46t
  /dev/nvme3n1 dwcache lvm2 a--  1.46t 1.46t
```

d. Remove LVM ownership of devices. Specify all SSD block devices on the node.

```
nid00349:# pvremove /dev/nvme0n1,/dev/nvme1n1,/dev/nvme2n1,/dev/nvme3n1
Labels on physical volume "/dev/nvme0n1" successfully wiped
Labels on physical volume "/dev/nvme1n1" successfully wiped
Labels on physical volume "/dev/nvme2n1" successfully wiped
Labels on physical volume "/dev/nvme3n1" successfully wiped
```

7. Repeat steps *5* on page 52 through *6* on page 52 for all SSD nodes listed in `datawarp_manager_nodes`.

DataWarp is deconfigured.

6 Troubleshooting

6.1 Old Nodes in dwstat Output

The DataWarp Service (DWS) learns about node existence from two sources:

1. Heartbeat registrations between the `dwsd` process and the `dwmd` processes
2. Hostnames provided by workload managers as users are granted access to compute nodes as part of their batch jobs

The `dwsd` process on the sdb node stores the DWS state in its state file and controls the information displayed by `dwstat nodes`. On `dwsd` process restart, `dwsd` removes a node from its state file if the node meets the following criteria:

1. the node is not in a pool
2. there are no instances on the node
3. there are no activations on the node
4. the node does not belong to a session

If a node lingers in the `dwstat nodes` output longer than expected, verify the above criterion are met, and then restart the `dwsd` process on the sdb node: `service dwsd restart`.

6.2 Dispatch Requests

The `dwsd` is designed to dispatch requests to the `dwmd` processes as soon as there is work for the `dwmd` processes to perform. If the `dwsd` gets confused or has a bug, it may fail to dispatch a request at the appropriate time. If this is suspected, send `SIGUSR1` to the `dwsd` process on the sdb node, forcing it to look for tasks to perform.

```
sdb:# kill -USR1 $(</var/opt/cray/dws/dwsd.pid)
sdb:# tail -6 /var/opt/cray/dws/log/dwsd.log
2015-09-17 15:24:05 ========== Event on fd 4
2015-09-17 15:24:05 Caught signal User defined signal 1
2015-09-17 15:24:05 Alerting the task manager to wake up
2015-09-17 15:24:05 ========== Event on fd 7
2015-09-17 15:24:05 Finding tasks to spawn
2015-09-17 15:24:05 Nothing can be done right now
```

More likely than not, the `dwsd` cannot yet perform the action in question. Check if any nodes are not online (`dwstat nodes`) and if all prerequisites to the action are met. For example, the `dwsd` will not dispatch a request to create a configuration until after the corresponding instance has been created.

7 Diagnostics

7.1 SEC Notification when 90% of SSD Life Expectancy is Reached

When a DataWarp SSD reaches 90% of its life expectancy, a message is written to the console log file. If enabled, the Simple Event Correlator (SEC) monitors system log files for significant events such as this and sends a notification (either by email, IRC, writing to a file, or some user-configurable combination of all three) that this has happened. The notification for nearing the end of life of an SSD is as follows:

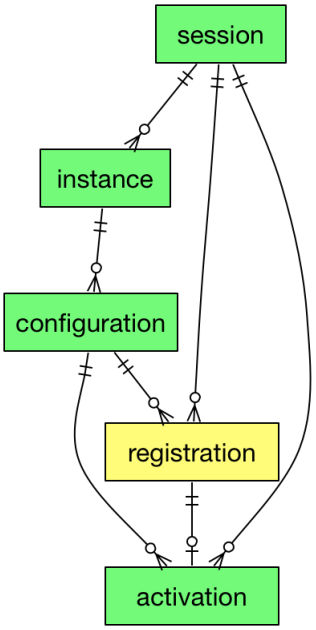
```
Mon 8/17/2015 3:17 PM
SEC: 15:17  sitename-systemname: Low SSD Life Remaining 8% c3-0c0s2n1 PCIe slot
-1
 Please contact your Cray support personnel or sales representative for SSD card
replacements.

 12 hours -- skip repeats period, applies on a per SSD basis.

 System:    sitename-systemname, sn9000
 Event:     Low ioMemory SSD Life Remaining (8%) c3-0c0s2n1 PCIe faceplate slot:
Unknown (only one slot is populated in this node)
 Time:      15:17:04 in logged string.
 Mon Aug 17 15:17:05 2015  --  Time when SEC observed the logged string.

 Entire line in log file:
 /var/opt/cray/log/p0-20150817t070336/console-20150817
 -----
 2015-08-17T15:17:04.871808-05:00 c3-0c0s2n1 PCIe slot#:-1,Name:ioMemory
SX300-3200,SN:1416G0636,Size:3200GB,Remaining life: 8%,Temperature:41(c)

 SEC rule file:
 --------------
 /opt/cray/sec/default/rules/aries/h_ssd_remaining_life.sr

 Note:
 -----
 The skip repeats period is a period during which any repeats of this event
type that occur will not be reported by SEC. It begins when the first  message
that triggered this email was observed by SEC.
```

For detailed information about configuring SEC, see *Configure Cray SEC Software (S-2542).*

8 Terminology

The following diagram shows the relationship between the majority of the DataWarp Service terminology using Crow's foot notation. A session can have 0 or more instances, and an instance must belong to only one session. An instance can have 0 or more configurations, but a configuration must belong to only one instance. A registration belongs to only one configuration and only one session. Sessions and configurations can have 0 or more registrations. An activation must belong to only one configuration, registration and session. A configuration can have 0 or more activations. A registration is used by 0 or no activations. A session can have 0 or more activations.

Figure 4. DataWarp Component Relationships



Activation An object that represents making a DataWarp configuration available to one or more client nodes, e.g., creating a mount point.

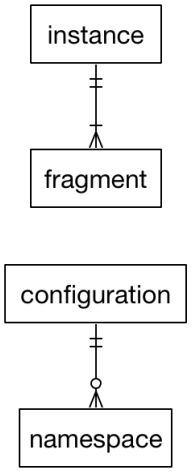
Client Node A compute node on which a configuration is activated; that is, where a DVS client mount point is created. Client nodes have direct network connectivity to all DataWarp server nodes. At least one parallel file system (PFS) is mounted on a client node.

Configuration A configuration represents a way to use the DataWarp space.

Fragment A piece of an instance as it exists on a DataWarp service node.

The following diagram uses Crow's foot notation to illustrate the relationship between an instance-fragment and a configuration-namespace. One instance has one or more fragments; a fragment can belong to only one instance. A configuration has 0 or more namespaces; a namespace can belong to only one configuration.

Figure 5. Instance/Fragment ↔ Configuration/Namespace Relationship



Instance A specific subset of the storage space comprised of DataWarp fragments, where no two fragments exist on the same node. An instance is essentially raw space until there exists at least one DataWarp instance configuration that specifies how the space is to be used and accessed.

DataWarp Service The DataWarp Service (DWS) manages access and configuration of DataWarp instances in response to requests from a workload manager (WLM) or a user.

Fragment A piece of an instance as it exists on a DataWarp service node

Job Instance A DataWarp instance whose lifetime matches that of a batch job and is only accessible to the batch job because the `public` attribute is not set.

Namespace A piece of a scratch configuration; think of it as a folder on a file system.

Node A DataWarp service node (with SSDs) or a compute node (without SSDs). Nodes with space are server nodes; nodes without space are client nodes.

Persistent Instance A DataWarp instance whose lifetime matches that of possibly multiple batch jobs and may be accessed by multiple user simultaneously because the `public` attribute is set.

Pool Groups server nodes together so that requests for capacity (instance requests) refer to a pool rather than a bunch of nodes. Each pool has an overall quantity (maximum configured space), a granularity of allocation, and a unit type. The units are either bytes or nodes (currently only bytes are supported). Nodes that host storage capacity belong to at most one pool.

Registration A known usage of a configuration by a session.

Server Node An IO service blade that contains two SSDs and has network connectivity to the PFS.

Session An intagible object (i.e., not visible to the application, job, or user) used to track interactions with the DWS; typically maps to a batch job.

9 Prefixes for Binary and Decimal Multiples

Multiples of bytes						
SI decimal prefixes				IEC binary prefixes		
Name	**Symbol**	**Standard SI**	**Binary Usage**	**Name**	**Symbol**	**Value**
kilobyte	kB	10^3	2^{10}	kibibyte	KiB	2^{10}
megabyte	MB	10^6	2^{20}	mebibyte	MiB	2^{20}
gigabyte	GB	10^9	2^{30}	gibibyte	GiB	2^{30}
terabyte	TB	10^{12}	2^{40}	tebibyte	TiB	2^{40}
petabyte	PB	10^{15}	2^{50}	pebibyte	PiB	2^{50}
exabyte	EB	10^{18}	2^{60}	exbibyte	EiB	2^{60}
zettabyte	ZB	10^{21}	2^{70}	zebibyte	ZiB	2^{70}
yottabyte	YB	10^{24}	2^{80}	yobibyte	YiB	2^{80}

For a detailed explanation, including a historical perspective, see *http://physics.nist.gov/cuu/Units/binary.html*.