DataWarp Installation and Configuration Guide

(CLE 5.2.UP04)

S-2547 Rev D

Contents

1 About the DataWarp Installation and Configuration Guide

This publication covers the installation procedure for DataWarp SSD cards as well as post-boot configuration; 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. See *Important Information about this DataWarp Release* on page 5.

Record of Revision

Revision D (1 May 2018): Removed incorrect content (commands and other content only available in CLE 6.0) that was caused by a bug in the portal.

Revision C (14 October 2016): Replace NVMe over-provision topic with Intel P3608 specific details, includes correct parameter value and improved examples.

Revision B (2 March 2016): Included source code for the `dwpoolhelp` command and modified the *DataWarp with DWS: Create a Storage Pool* on page 17 procedure to use `dwpoolhelp`.

Revision A (6 November 2015): Clean up errors and incorporate bugfixes.

Initial publication: September 2015

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 53.

2.1 Supported DataWarp SSD Cards

The following types of SSD cards are supported on Cray XC series systems:

- FusionIO ioScale2
- FusionIO SX300/PX600
- Intel NVMe

The installation procedures differ depending on which type of SSD card is installed. It is the administrator's responsibility to know which type is installed on the machine. Follow the appropriate procedures for the type of SSD cards installed on the system.

FusionIO SSD Cards

Installation of the appropriate FusionIO driver is included in these procedures based on the components defined in the following table.

Entity	ioScale2 SSD Cards	SX300/PX600 SSD Cards
Driver	VSL3	VSL4
RPM naming convention	fio-common-*`version_vsl`*.noarch.rpm fio-sysvinit-*`version_vsl`*.noarch.rpm fio-util-*`version_vsl`*.noarch.rpm iomemory-vsl-*`version_vsl`*.src.rpm	fio-common-*`version_vsl4`*.x86_64.rpm fio-preinstall-*`version_vsl4`*.x86_64.rpm fio-sysvinit-*`version_vsl4`*.x86_64.rpm fio-util-*`version_vsl4`*.x86_64.rpm iomemory-vsl4-*`version_vsl4`*.x86_64.rpm
Service name	`iomemory-vsl`	`iomemory-vsl4`

NVMe SSD Cards

The NVMe driver is already installed in the kernel.

3 CLEinstall Configuration Parameters

3.1 Set CLEinstall.conf Parameters for SSD Cards

Prerequisites

- A Cray XC series system with one or more SSD cards installed

About this task

The `CLEinstall` program ensures that the proper RPMs are installed during the installation or upgrade of CLE software based on SSD-specific parameters in the `CLEinstall.conf` file. Prior to running `CLEinstall` during an installation or upgrade of CLE system software, or when running `CLEinstall` as a separate event, follow this procedure to set the SSD-specific parameters.

Procedure

1. Edit `CLEinstall.conf`.

a. Set the value of `SSDtype` based on the type(s) of SSD card(s) present in the system. Use the table below to determine the value.

Table 1. SSDtype setting based on SSDs installed

Device Type(s)	`SSDtype`
NVMe only	`none`
IoScale2 only	`ioscale2`
SX300/PX600 only	`iomemory3`
NVMe + IoScale2	`ioscale2`
NVMe + SX300/PX600	`iomemory3`

b. (**Static DataWarp only**) Set `CNL_swap=yes` if using the SSD cards for swapping. Default is `no`.

2. Save the changes and proceed as directed based on the type of DataWarp being installed.

- **DataWarp with DWS**: Proceed to *DataWarp with DWS: Set CLEinstall.conf Parameters for the DataWarp Service* on page 8.
- **Static DataWarp**: Continue to the step *3* on page 8.

3. If changes were made to `CLEinstall.conf` in step *1* on page 7, follow the procedures in CLE Installation and Configuration Guide to run `CLEinstall`.

4. Proceed as directed based on the type of SSD card(s) installed:
   - **FusionIO**: Proceed to *Install the FusionIO Driver and Configure FusionIO Cards* on page 11.
   - **NVMe**: Proceed to *Static DataWarp: Configure Storage on SSD Cards* on page 31.

3.2 DataWarp with DWS: Set CLEinstall.conf Parameters for the DataWarp Service

Prerequisites

- A Cray XC series system running CLE5.2.UP04
- SSD cards exist on one or more service nodes
- Completion of *Set CLEinstall.conf Parameters for SSD Cards* on page 7
- A parallel file system (PFS) is mounted in the same location on all compute nodes as well as all service nodes identified by `datawarp_manager_nodes` (defined in `CLEinstall.conf`). In other words, the mount points must look the same on compute and SSD-endowed service nodes. More than one PFS is allowed.
  - This requirement supports the staging functionality of DataWarp and can be implemented after the configuration of DataWarp.

About this task

The CLE installation configuration file, `CLEinstall.conf`, includes the following section that defines the option and parameters to enable and initially configure the DataWarp Service (DWS).

```
#######################################################################
# DataWarp settings
#
# DataWarp is a combination of commercial SSD hardware and software,
# Linux community software, and Cray XC system with CLE software,
# which provides a high bandwidth file based store for application I/O.
#######################################################################
# Should DataWarp be enabled?
datawarp=no
#
# The comma-separated list of nodes (cnames)on which the DataWarp manager will be
# run.
# DataWarp managers manage the storage capacity of the SSDs
# These nodes must have SSDs on them.
#
#datawarp_manager_nodes=c0-0c1s2n2,c0-0c1s2n3
#
# The list of datawarp_api_gateways nodes--required to be either internal login or
# MOM node(s)--must contain one or more comma-separated triplets
# like <hostname>:<node>:<port>
# DataWarp API Gateway nodes act as an interface from users to the rest of the
# DataWarp service.
# <hostname> can be cname, nid#####, or hostname that other nodes can use to reach
```

```
# this node.
# <node> is the node number, an integer
# <port> the port that the DataWarp daemon listens on, 81 is standard
#
#datawarp_api_gateways=login:18:81,nid00123:123:81

#
# Comma-separated list of UIDs which are administrators for DataWarp.  Both root
# and crayadm are already on this list.
#
#
#datawarp_admin_uid_list=
```

To enable DWS, edit `CLEinstall.conf` as follows:

Procedure

1. Set `datawarp=yes`.

2. Determine which nodes have SSD cards.

   The `xtssdconfig` command displays nodes with NVMe cards, and the `xtcheckhss` command can display all PCIe cards. (Example output is truncated for readability.)

```
smw:# xtssdconfig
     node_id     ssd_id     sub_id  bus  device func    size ...
-----------------------------------------------------------------
   c0-0c0s5n2  0x8086953 0x80863709 0x05   0x00  0x0         ...
   c0-0c0s5n2  0x8086953 0x80863709 0x06   0x00  0x0         ...

smw:# xtcheckhss --nocolor --detail=f --pci
Node        Slot Name                                                ...
----------- ---- ------------------------------------------------- ----------
c0-0c0s0n1   2   QLogic_ISP2432_4Gb_Single/Dual_Fibre_Channel_HBA    ...
...
c0-0c0s5n1   3   Empty                                                ...
c0-0c0s5n2   0   Intel_P3600_Series_SSD                               ...
c0-0c0s5n2   0   Intel_P3600_Series_SSD                               ...
...
c0-0c1s0n1   2   Fusion-io_ioMemory3_Atomic_Series_SSD                ...
c0-0c1s0n1   3   Empty                                                ...
c0-0c1s0n2   0   Fusion-io_ioMemory3_Atomic_Series_SSD                ...
c0-0c1s0n2   1   Empty                                                ...
c0-0c1s1n1   2   Fusion-io_ioMemory3_Atomic_Series_SSD                ...
c0-0c1s1n1   3   Empty                                                ...
c0-0c1s1n2   0   Fusion-io_ioMemory3_Atomic_Series_SSD                ...
c0-0c1s1n2   1   Empty                                                ...
```

3. Uncomment the `datawarp_manager_nodes` line, and add a comma-separated list of one or more cnames of SSD-endowed nodes to be monitored by DWS (as shown in the configuration file).

4. Uncomment the `datawarp_api_gateways` line, and add a comma-separated list of one or more *`hostname:nid:port`* triads (as shown in the configuration file).

   - Specify login or MOM nodes only. Only one triad is required. Very large systems may need multiple triads, but Cray recommends starting with only one.

5. Optional: It is not necessary to modify `datawarp_admin_uid_list`, because the default values of `root` and `crayadm` are sufficient.
   - If the workload manager UID (numeric) is known, it can be added to the list. Remember to uncomment the line.
6. Follow the procedures in CLE Installation and Configuration Guide to run `CLEinstall`.
7. Update the boot automation file; follow the procedure in *Configure Boot Automation for DataWarp* on page 52.
8. Proceed as directed based on the type of SSD card(s) installed:
   - **FusionIO**: Proceed to *Install the FusionIO Driver and Configure FusionIO Cards* on page 11.
   - **NVMe**: Proceed to *DataWarp with DWS: Post-boot Configuration* on page 15.

4 Install the FusionIO Driver and Configure FusionIO Cards

Prerequisites

- **DataWarp with DWS**: Completion of *DataWarp with DWS: Set CLEinstall.conf Parameters for the DataWarp Service* on page 8
- **Static DataWarp**: Completion of *Set CLEinstall.conf Parameters for SSD Cards* on page 7 followed by successful execution of `CLEinstall`
- If adding a new blade to the system, see "Updating the System Configuration After a Blade Change" in S-2393 prior to proceeding.

Procedure

1. Log on to the boot node (use the currently booted system or boot the boot and SDB nodes).

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

2. Initiate `xtopview`.

```
boot:# xtopview
```

3. Turn off the `iomemory-vsl` service in the default view and verify the results.
   - For VSL3:

```
default/:/# chkconfig iomemory-vsl off
default/:/# chkconfig --list iomemory-vsl
iomemory-vsl      0:off   1:off   2:off   3:off   4:off   5:off   6:off
```

   - For VSL4:

```
default/:/# chkconfig iomemory-vsl4 off
default/:/# chkconfig --list iomemory-vsl4
iomemory-vsl4     0:off   1:off   2:off   3:off   4:off   5:off   6:off
```

4. Edit `/etc/sysconfig/iomemory-vsl` to modify the iomemory configuration for this node.
   - For VSL3:

```
default/:/# vi /etc/sysconfig/iomemory-vsl
```

   - For VSL4:

```
default/:/# vi /etc/sysconfig/iomemory-vsl4
```

   a. Ensure that `ENABLED=1` is set and not commented out.

b. Save the changes.

5. Exit `xtopview`

```
default/:/# exit
```

6. Initiate the node specialization view, where *N* is the integer node ID or physical ID of an SSD node.

```
boot:# xtopview -n N
```

7. Turn on the `iomemory-vsl` service and verify the results.

- For VSL3:

```
node/N:/# chkconfig iomemory-vsl on
node/N:/# chkconfig --list iomemory-vsl
iomemory-vsl      0:off   1:on   2:on   3:on   4:on   5:on   6:off
```

- For VSL4:

```
node/N:/# chkconfig iomemory-vsl4 on
node/N:/# chkconfig --list iomemory-vsl4
iomemory-vsl4     0:off   1:on   2:on   3:on   4:on   5:on   6:off
```

8. Exit xtopview and `ssh` to the specialized node.

```
node/N:/# exit
boot:# ssh cname
```

9. Use the `init` script to load the driver.

- For VSL3:

```
nid:# /etc/init.d/iomemory-vsl start
```

- For VSL4:

```
nid:# /etc/init.d/iomemory-vsl4 start
```

10. Look for the new devices.

```
nid:# fdisk -l /dev/fio*
Disk /dev/fioa: 2600.0 GB, 2600000000000 bytes
255 heads, 63 sectors/track, 39512 cylinders, total 634765625 sectors
Units = sectors of 1 * 4096 = 4096 bytes
Sector size (logical/physical): 4096 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 32768 bytes
Disk identifier: 0x6901a7ad

Disk /dev/fioa doesn't contain a valid partition table
Note: sector size is 4096 (not 512)

Disk /dev/fiob: 2600.0 GB, 2600000000000 bytes
255 heads, 63 sectors/track, 39512 cylinders, total 634765625 sectors
Units = sectors of 1 * 4096 = 4096 bytes
Sector size (logical/physical): 4096 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 32768 bytes
Disk identifier: 0x00000000
```

```
Disk /dev/fiob doesn't contain a valid partition table
```

11. Gather the serial numbers for each SSD card; these are used to change the power limit for these cards. This occurs in step *14* on page 13 after all serial numbers are gathered.

```
nid:# fio-status | grep SN
```

12. Exit to the boot node and repeat steps *6* on page 12 through *11* on page 13 for each SSD node.

13. Exit to the boot node and initiate `xtopview`.

```
nid:# exit
boot:# xtopview
```

14. Edit the configuration file and add an `options` entry to set the power limit for all PCI SSD cards to 40W in order to allow full performance, where *`SN-value`*`:40` represents an adapter serial number and the maximum amount of power that device should pull (in watts). Multiple pairs must be comma-separated.

> **IMPORTANT:** Currently there is a limitation of only one `options` entry within the configuration file. This means that all `serial number:40` pairs for all SSD cards must be entered in that one entry as a comma-separated list. If there are multiple lines labeled `options`, only the last one will be recognized.

- For VSL3:

  ```
  default/:/# vi /etc/modprobe.d/iomemory-vsl.conf
  ```

  ```
  options iomemory-vsl external_power_override=SN-value:40
  ```

  For example:

  ```
  options iomemory-vsl external_power_override=1234G5678:40,0987G6543:40
  ```

- For VSL4:

  ```
  default/:/# vi /etc/modprobe.d/iomemory-vsl4.conf
  ```

  ```
  options iomemory-vsl4 external_power_override=SN-value:40
  ```

  For example:

  ```
  options iomemory-vsl4 external_power_override=1234G5678:40,0987G6543:40
  ```

When this is properly configured, a message similar to the following appears in the console log after a reload/reboot.

```
[12:37:46][c0-0c0s7n1]fioinf ioDrive 0000:03:00.0.0: PCIe power monitor
enabled (master). Limit set to 39.750 watts.
```

15. Exit `xtopview`.

16. Reboot the node for the new settings to take effect. See *Proper Swap Server and SSD Shutdown for FusionIO Cards* on page 48.

17. Repeat step *6* on page 12 though step *16* on page 13 for all SSD nodes.

18. Proceed to post-boot configuration based on the type of DataWarp being installed:

- **DataWarp with DWS**: Proceed to *DataWarp with DWS: Post-boot Configuration* on page 15.
- **Static DataWarp**: Proceed to *Static DataWarp: Configure Storage on SSD Cards* on page 31.

5 DataWarp with DWS: Post-boot Configuration

About this task

After the system boots, and after the device-specific setup is completed, DWS requires further manual configuration. The steps required are:

Procedure

1. Ensure that all DVS and DataWarp patches are installed. For proper execution and outcome, these procedures require all patches.

2. Initialize SSDs for use with DWS. See *DataWarp with DWS: Initialize an SSD* on page 15.

3. Create storage pools. See *DataWarp with DWS: Create a Storage Pool* on page 17.

4. Assign nodes with space to a storage pool. See *DataWarp with DWS: Assign a Node to a Storage Pool* on page 27.

   IMPORTANT: Repeat step *2* on page 15 and step *4* on page 15 for each SSD-endowed node that DWS manages.

5. Configure the Node Health Checker (NHC) to run the DataWarp test. See *Enable the Node Health Checker DataWarp Test* on page 28.

6. Verify the configuration. See *DataWarp with DWS: Verify the DataWarp Configuration* on page 29.

   Completion of these steps concludes the installation and configuration procedure for DataWarp with DWS.

5.1 DataWarp with DWS: Initialize an SSD

Prerequisites

- **FusionIO cards**: Completion of *Install the FusionIO Driver and Configure FusionIO Cards* on page 11
- **NVMe cards**: Completion of *DataWarp with DWS: Set CLEinstall.conf Parameters for the DataWarp Service* on page 8
- **NVMe cards**: Completion of *Over-provision an Intel P3608 SSD* on page 40
- Ability to log in as `root`

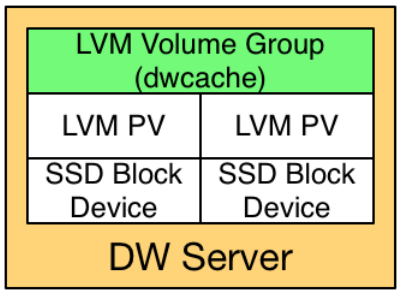
About this task

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 53.

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 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.



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 the logical volume manager (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.

   Update example output.

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

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 *DataWarp with DWS: Initialize an SSD* on page 15.

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 21.

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 20 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:

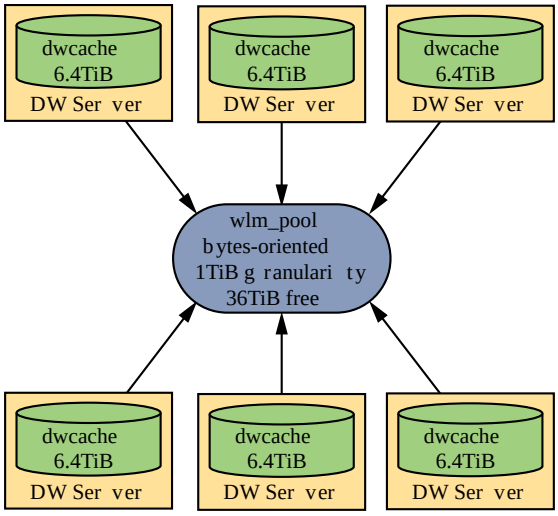
1. The byte-oriented allocation granularity for a pool must be at least 16MiB.
2. Each node's volume group (`dwcache`, configured in *DataWarp with DWS: Initialize an SSD* on page 15) has a Physical Extent size (*`PE_size`*) and Physical Volume count (*`PV_count`*). The default *`PE_size`* is 4MiB, and *`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 21 of the procedure.

Procedure

1. Build the `dwpoolhelp` command from the source in *The dwpoolhelp Command Source Code* on page 21.

```
$ 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 17).

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 DataWarp with DWS: Assign a Node to a Storage Pool

Prerequisites

- At least one storage pool exists, see *DataWarp with DWS: Create a Storage Pool* on page 17.
- At least one SSD is initialized for use with the DWS, see *DataWarp with DWS: Initialize an SSD* on page 15.
- 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
```

5.5 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.

5.6 DataWarp with DWS: Verify the DataWarp Configuration

Prerequisites

- At least one storage node is assigned to a storage pool, see *DataWarp with DWS: Assign a Node to a Storage Pool* on page 27.
- 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 53.

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 *DataWarp with DWS: Assign a Node to a Storage Pool* on page 27.
   - 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* on page 15. Refer to the site-specific Workload Manager documentation for further configuration steps to integrate the WLM with Cray DataWarp.

6 Static DataWarp: Post-boot Configuration

6.1 Static DataWarp: Configure Storage on SSD Cards

Prerequisites

- **FusionIO cards**: Completion of *Install the FusionIO Driver and Configure FusionIO Cards* on page 11.
- **NVMe cards**: Completion of *Set CLEinstall.conf Parameters for SSD Cards* on page 7.
- If adding a new blade to the system, see "Update the System Configuration After a Blade Change" in *CLE XC™ System Administration Guide (S-2393)* prior to proceeding.

About this task

The SSD cards can be configured as scratch/temporary storage or for swapping. These procedures cover both options.

Procedure

1. Log on to the boot node (use the currently booted system or boot the boot and SDB nodes).

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

2. Initiate `xtopview`.

```
boot:# xtopview
```

3. Edit the LVM configuration file in the default shared root view to ensure that the following lines are included in the `advanced settings` area within the `device` section.

```
default/:/# vi /etc/lvm/lvm.conf
```

```
types = [ "fio", 16
          "nvme", 64 ]
```

4. Edit /etc/sysconfig/iomemory-vsl to modify the iomemory configuration for this node.
   - For VSL3:

     ```
     default/:/# vi /etc/sysconfig/iomemory-vsl
     ```

   - For VSL4:

```
default/:/# vi /etc/sysconfig/iomemory-vsl4
```

a. Ensure that `ENABLED=1` is set and not commented out.

b. Define `LVM_VGS`, the LVM volume groups parameter.

```
LVM_VGS="/dev/vg_name"
```

For example:

```
LVM_VGS="/dev/ssd_vg"
```

5. Exit `xtopview`

```
default/:/# exit
```

6. `ssh` to the specialized node.

```
boot:# ssh cname
```

7. Look for the new devices (`/dev/fio*` or `/dev/nvm*`).

```
nid:# ls -l /dev/nvm*
crw-rw---- 1 root root  10, 57 Jun  3 12:13 /dev/nvme0
brw-rw---- 1 root disk 254,  0 Jun  3 12:28 /dev/nvme0n1
crw-rw---- 1 root root  10, 56 Jun  3 12:13 /dev/nvme1
brw-rw---- 1 root disk 254, 64 Jun  3 12:28 /dev/nvme1n1
```

8. Set up devices with LVM (create physical volume and logical volume group).

```
nid:# pvcreate /dev/nvme#n#p# /dev/nvme#n#p# [/dev/nvme#n#p# ...]
nid:# vgcreate ssd_vg /dev/nvme#n#p# /dev/nvme/nvm#n#p# [/dev/nvme#n#p# ...]
```

For example:

```
nid:# pvcreate /dev/nvme0n1 /dev/nvme1n1
  Physical volume "/dev/nvme0n1" successfully created
  Physical volume "/dev/nvme1n1" successfully created
nid:# vgcreate ssd_vg /dev/nvme0n1 /dev/nvme1n1
  Volume group "ssd_vg" successfully created
```

9. (**Scratch setup only**) Create scratch volume.

Where *`stripes`* is the number of physical volumes in the volume group, i.e., *`stripes`*=2 for 2 FusionIO SSDs or NVMe SSDs with one logical device per card, and *`stripes`*=4 for NVMe SSDs with two logical devices per card.

```
nid:# lvcreate -i stripes -I stripesize -l extents -n lv_name ssd_vg
nid:# mkfs.xfs /dev/ssd_vg/lv_name
```

This example creates one volume using 100% of the capacity of the volume group with data striped across the devices in 1MiB blocks.

```
nid:# lvcreate -i 2 -I 1M -l 100%VG -n scratch ssd_vg
nid:# mkfs.xfs /dev/ssd_vg/scratch
```

10. (**Swap setup only**) Create swap volume. Create one logical volume per compute node using swap.

Where *stripes* is the number of physical volumes in the volume group, i.e., *stripes*=2 for 2 FusionIO SSDs or NVMe SSDs with one logical device per card, and *stripes*=4 for NVMe SSDs with two logical devices per card.

```
nid:# lvcreate --size lvsize -n lv_name -i stripes -I stripesize ssd_vg
nid:# mkswap /dev/ssd_vg/lv_name
```

This example creates one volume using 32GiB of capacity of each device with data striped in 1MB blocks.

```
nid:# lvcreate --size 32G -n swap1 -i 2 -I 1M ssd_vg
nid:# mkswap /dev/ssd_vg/swap1
```

11. Exit to the boot node.

12. Proceed to post-boot configuration based on the storage purpose:

- **Scratch Storge**: proceed to *Static DataWarp: Configure Service Node for Scratch Storage* on page 33.
- **Swap Storage**: proceed to *Static DataWarp: Configure Compute Node Access to SSD Swap Storage* on page 36.

6.2 Static DataWarp: Configure Service Node for Scratch Storage

Prerequisites

- Completion of *Static DataWarp: Configure Storage on SSD Cards* on page 31

Procedure

1. Initiate the node specialization view, where *N* is an integer node ID or a physical ID.

```
boot:# xtopview -n N
```

2. Create a mount point for the scratch file system.

```
node/N:/# mkdir -p mount_point
```

For example:

```
node/N:/# mkdir -p /flash/scratch1
```

3. Specialize the `/etc/fstab` file for this node only.

```
node/N:/# xtspec /etc/fstab
```

4. Add an entry in the `/etc/fstab` file to mount the file system, where *ssd_vg* is the volume group and *lv_name* is the logical volume name, both of which will be created later.

IMPORTANT: The `/etc/fstab` entry must be tab-separated for LVM, and the entry must be entered on one line only. Note that the displays below will wrap based on page width.

```
/dev/vg_name/lv_name mount_point  xfs
defaults,noatime,nobarrier,discard,allocsize=1M  0  0
```

For example:

```
/dev/ssd_vg/scratch1 /flash/scratch1  xfs
defaults,noatime,nobarrier,discard,allocsize=1M  0   0
```

5. (Optional - NFS setup) Set up NFS export so that scratch is accessible from the login or other service node.

a. Specialize the `/etc/exports` file for this node only.

```
node/N:/# xtspec /etc/exports
```

b. Edit the `/etc/exports` file and add an entry for scratch.

```
node/N:/# vi /etc/exports
```

```
mount_point *(rw,no_root_squash,no_subtree_check)
```

For example:

```
/flash/scratch1 *(rw,no_root_squash,no_subtree_check)
```

c. Turn on the NFS server.

```
node/N:/# chkconfig nfsserver on
```

6. Exit xtopview and proceed to *Static DataWarp: Configure Compute Node Access to SSD Scratch Storage* on page 34.

6.3 Static DataWarp: Configure Compute Node Access to SSD Scratch Storage

Prerequisites

- Completion of *Static DataWarp: Configure Service Node for Scratch Storage* on page 33

Procedure

1. Log on to the SMW as `root`.

2. Change directory to the boot image template directory.

```
smw:#  cd /opt/xt-images/templates/cle_version-system_set
```

3. Create a mount point for the scratch file system.

```
smw:#  mkdir -p mount_point
```

For example:

```
smw:#  mkdir -p flash/scratch1
```

4. Edit the compute node `fstab` file and add a tab-separated entry (on one line) to mount the file system, where *`cname`* is the SSD node.

```
smw:# vi etc/fstab
```

```
 device   mount_point  dvs path=mount_point,nodename=cname,blksize=1048576,ro_cache
```

For example:

```
/flash/scratch1   /flash/scratch1    dvs path=/flash/
scratch1,nodename=c0-0c0s7n1,blksize=1048576,ro_cache
```

5. Optional: Exclude the new file system from node health checks by appending an entry to the Node Health Checker (NHC) configuration file.

   DataWarp is configured for maximum performance by using LVM to create a striped, RAID0-like file system. This significantly increases the risk of device failures; therefore, Cray recommends either excluding the scratch storage from NHC or directing NHC to issue warnings rather than take down the nodes when a file system fails.

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

Append with:

```
Excluding: mount_point
```

For example:

```
Excluding: /flash/scratch1
```

> **IMPORTANT:** If this exclusion is not made and the file system fails, compute nodes will be marked down.

6. Invoke `shell_bootimage.sh` and create a new boot image using the site's regular procedure.

   Now that the storage is configured, the SSD service node and the compute nodes must be rebooted to pick up the changes. There are two options for how to proceed:

   - The first option is to exit from this procedure and reboot the system following the site-specific procedures.
   - The second option is to complete the steps in this procedure to reboot the SSD service node, reboot the compute nodes, then mount the scratch filesystem on the login nodes.

7. (**FusionIO cards only**) Log on to the SSD service node.

```
smw:#  ssh cname
```

For example:

```
smw:#  ssh c0-0c0s7n1
```

8. (**FusionIO cards only**) Stop the file system safely, deactivate volume groups and unload the driver by invoking the `init` script.
   - For VSL3:

     ```
     nid:# /etc/init.d/iomemory-vsl stop
     ```
   - For VSL4:

     ```
     nid:# /etc/init.d/iomemory-vsl4 stop
     ```

9. (**FusionIO cards only**) Exit the node.

   ```
   nid:# exit
   ```

10. Reboot the SSD service node.

    ```
    smw:#  xtbootsys --reboot -L SNL0 cname
    ```

11. Log off as `root`.

    ```
    smw:#  exit
    ```

12. Reboot the compute nodes, where *`cnames`* is a comma-separated list of all compute nodes for this partition. Alternatively, shutdown and reboot the entire system if desired.

    ```
    crayadm@smw:#  xtbootsys --reboot -L CNL0 cnames
    ```

    This concludes the installation procedure of SSD cards configured as scratch storage for Static DataWarp.

6.4 Static DataWarp: Configure Compute Node Access to SSD Swap Storage

Prerequisites

- Completion of *Static DataWarp: Configure Storage on SSD Cards* on page 31

Procedure

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

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

2. Create a mapping file `/etc/opt/cray/swap/swap.map` in the shared root that maps each compute node to its target swap server node and the logical volume.

The file format is as follows, with one compute node per line and columns separated by white space.

```
Compute node(cname) swap server node(swap_cname) Logical Unit Number(LUN)
```

This file should be visible from the compute nodes, and the path to this file will be an argument to a script that configures the swap space per compute node.

For example:

```
c3-0c0s4n0 c3-0c0s2n2 0
c3-0c0s4n1 c3-0c0s2n2 1
```

3. Edit the `/etc/opt/cray/swap/swap.config` file to set the desired level of log messages.

   The valid logging levels are: `CRITICAL`, `ERROR`, `WARNING`, `INFO`, `DEBUG`, `NOTSET`. Default is `WARNING`. Log messages of the selected severity or greater will be outputted.

   For example:

```
LOG_LEVEL=INFO
```

4. Exit `xtopview`.

```
default/:/# exit
boot:#
```

5. `SSH` to an SSD node.

```
boot:# ssh cname
```

6. Determine the path to each logical volume. Output shown is for example purposes only, each site's configuration will vary.

```
nid:# lvdisplay | grep "LV Name"
LV Name /dev/swap_vg/swap-c0-0c0s4n0
LV Name /dev/swap_vg/swap-c0-0c0s4n1
LV Name /dev/swap_vg/swap-c0-0c0s4n2
```

7. Exit to the boot node and then initiate the node specialization view for the swap node.

```
nid:# exit
boot:# xtopview -n swap_nid
```

8. Create the configuration file `/etc/ietd.conf` if it doesn't exist. Use `xtspec` to specialize the file.

```
nid:# touch /etc/ietd.conf
nid:# xtspec /etc/ietd.conf
```

9. Edit `/etc/ietd.conf` and add the following, where *`cname`* is the node on which the the swap server is running, and *`swap_path`* and *`lv_path`* are obtained from step *6* on page 37.

```
Target iqn.2001-01.com.cray:cname.swap
      Lun 0 Path=/dev/swap_path/lv_path
      Lun 1 Path=/dev/swap_path/lv_path
      ...
```

For example:

```
Target iqn.2001-01.com.cray:c0-0c0s7n2.swap
      Lun 0 Path=/dev/swap_vg/swap-c0-0c0s4n0
      Lun 1 Path=/dev/swap_vg/swap-c0-0c0s4n1
      Lun 2 Path=/dev/swap_vg/swap-c0-0c0s4n2
```

10. Exit `xtopview` and `ssh` to the SSD node.

```
nid:# exit
boot:# ssh cname
```

11. Turn on the `iscsi-target` service so that it starts upon boot of the swap server node.

```
nid:# chkconfig iscsi-target on
```

12. Restart the iSCSI server by either executing `iscsi-target restart` or rebooting the swap server node.

```
nid:# /etc/init.d/iscsi-target restart
```

13. Verify that the target and LUNS are present.

```
nid:# cat /proc/net/iet/volume
tid:1 name:iqn.2001-01.com.cray:c0-0c0s7n2.swap
   lun:0 state:0 iotype:fileio iomode:wt blocks:10000000 blocksize:512 path:/dev/
swap_vg/swap-c0-0c0s4n0
   lun:0 state:0 iotype:fileio iomode:wt blocks:10000000 blocksize:512 path:/dev/
swap_vg/swap-c0-0c0s4n1
   lun:0 state:0 iotype:fileio iomode:wt blocks:10000000 blocksize:512 path:/dev/
swap_vg/swap-c0-0c0s4n2
```

14. Exit to the boot node and repeat steps *5* on page 37 through step *13* on page 38 for each swap server node.

```
nid:# exit
```

Client Configuration - Configure Multipath within the Shared Root.

15. Initiate `xtopview`.

```
boot:# xtopview
```

16. Create the `/etc/multipath` directory prior to configuring the compute nodes.

```
default/:/# mkdir /etc/multipath
```

17. Edit `/etc/multipath.conf`. Add the bolded items in the example shown if they are not already in the configuration file.

```
blacklist_exceptions {
       property (ID_WWN|ID_SCSI_VPD|UDEV_LOG)
       device {
       vendor "IET"
       product "VIRTUAL-DISK"
       }
}
defaults {
```

```
    polling_interval 10
    path_selector "round-robin 0"
    path_grouping_policy multibus
    prio const
    path_checker directio
    max_fds 8192
    rr_weight priorities
    failback immediate
    no_path_retry 30
    user_friendly_names yes
    getuid_callout "/lib/udev/scsi_id -g -u -d /dev/%n"
}
devices {
     device {
        vendor "IET"
        product "VIRTUAL-DISK"
        no_path_retry queue
     }
...
}
```

18. Exit `xtopview`.

```
default/:/# exit
```

This concludes the installation procedure of SSD cards configured as swap for Static DataWarp.

7 Administration Tasks for NVMe SSD Cards

7.1 Over-provision an Intel P3608 SSD

Prerequisites

- A Cray XC series system with one or more Intel P3608 SSD cards installed
- Ability to log in as `root`

About this task

This procedure is only valid for Intel P3608 SSDs.

WARNING: This procedure destroys any existing data on the SSDs.

Over-provisioning determines the size of the device available to the Logical Volume Manager (LVM) commands and needs to occur prior to executing any LVM commands. Typically, over-provisioning is done when the SSD cards are first installed.

> **TIP:** Throughout these procedures, units of bytes are described using the binary prefixes defined by the International Electrotechnical Commission, e.g., mebi- (MiB), gibi- (GiB), tebi- (TiB), pebi- (PiB).

Procedure

1. Log in to an Intel P3608 SSD-endowed node as `root`.

   This example uses `nid00350`.

2. Remove any existing configuration.

   > **TIP:** Numerous methods exist for creating configurations on an SSD; these instructions may not capture all possible cleanup techniques.

   a. Unmount file systems (if any).

   ```
   nid00350# df
   boot:/home                    20961280     11352064     9609216  55% /home
   tmp                        61504671488    624927640 57802802440   2% /scratch
   nid00350# umount -f /scratch
   ```

   b. Remove logical volumes (if any).

```
nid00350# lvdisplay
  --- Logical volume ---
  LV Path                /dev/dwcache/s98i94f104o0
  LV Name                s98i94f104o0
  VG Name                dwcache
  LV UUID                910tio-RJXq-puYV-s3UL-yDM1-RoQl-HugeTM
  LV Write Access        read/write
  LV Creation host, time nid00350, 2016-02-22 13:29:11 -0500
  LV Status              available
  # open                 0
  LV Size                3.64 TiB
  Current LE             953864
  Segments               2
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     1024
  Block device           253:0

nid00350# lvremove /dev/dwcache
```

c. Remove volume groups (if any).

```
nid00350# vgs
  VG       #PV #LV #SN Attr   VSize VFree
  dwcache    4   0   0 wz--n- 7.28t 7.28t
nid00350# vgremove dwcache
  Volume group "dwcache" successfully removed
```

d. Remove physical volumes (if any).

```
nid00350# pvs
PV           VG        Fmt Attr PSize PFree
/dev/nvme0n1          lvm2 a--  1.82t 1.82t
/dev/nvme1n1          lvm2 a--  1.82t 1.82t
/dev/nvme2n1          lvm2 a--  1.82t 1.82t
/dev/nvme3n1          lvm2 a--  1.82t 1.82t

nid00350# 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
```

e. Remove partitions for each device removed in the previous step (if any).

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

```
nid00350# dd if=/dev/zero of=phys_vol bs=512 count=1
```

```
nid00350# dd if=/dev/zero of=/dev/nvme0n1 bs=512 count=1
nid00350# dd if=/dev/zero of=/dev/nvme1n1 bs=512 count=1
nid00350# dd if=/dev/zero of=/dev/nvme2n1 bs=512 count=1
nid00350# dd if=/dev/zero of=/dev/nvme3n1 bs=512 count=1
```

3. Reconfigure the device.

```
nid00350# module load linux-nvme-ctl
nid00350# nvme set-feature device -n 1 -f 0XC1 -v 3125623327
set-feature:193(Unknown), value:00000000
```

4. Confirm the change. Note that 0xba4d3a1f = 3125623327.

```
nid00350# nvme get-feature device -n 1 -f 0XC1 --sel=0
get-feature:193(Unknown), value:0xba4d3a1f
```

5. Repeat for all devices; for example: `/dev/nvme0` and `/dev/nvme1` on the first card and `/dev/nvme2` and `/dev/nvme3` on the second card (if installed).

6. Return to the SMW, and warm boot the node.

```
crayadm@smw> xtnmi cname
crayadm@smw> sleep 60
crayadm@smw> xtbootsys --reboot -r "warmboot for Intel SSD node" cname
```

7. Confirm that `SIZE` = `1600319143936` for all volumes.

```
nid00350# lsblk -b
NAME    MAJ:MIN RM          SIZE RO TYPE MOUNTPOINT
loop0     7:0    0        196608 0  loop /var/opt/cray/imps-distribution/squash/
loop1     7:1    0         65536 0  loop /var/opt/cray/imps-distribution/squash/
nvme0n1 259:0    0 1600319143936 0  disk
nvme1n1 259:1    0 1600319143936 0  disk
nvme2n1 259:2    0 1600319143936 0  disk
nvme3n1 259:3    0 1600319143936 0  disk
```

Contact Cray service personnel if `SIZE` is incorrect.

7.2 Flash the Intel P3608 Firmware

Prerequisites

- A Cray XC series system with one or more Intel P3608 SSD cards installed
- Ability to log in as `root`
- Access to an Intel P3608 image flash file (location provided by Cray)

About this task

This procedure, typically only done at Cray's recommendation, ensures that the firmware of any Intel P3608 SSD cards is up-to-date with an image flash file. The `xtiossdflash` command compares the current flash version to the image flash file and flashes the device (up or down) only if the two are different.

For further information, see the `xtiossdflash(8)` man page.

```
boot# man -l /opt/cray/ssd-flash/man/man8/xtiossdflash.8
```

Procedure

1. Log on to the boot node as `root` and load the `pdsh` module.

```
smw:# ssh root@boot
boot:# module load pdsh
```

2. Copy the firmware image to the target nodes.

```
boot:# scp P3608_fw_image_file target:/location
```

For example:

```
boot:# scp /P3608_FW/FW1B0_BL133 c0-1c0s9n2:/tmp
```

3. Flash the firmware.

```
boot:# /opt/cray/ssd-flash/bin/xtiossdflash -f -i /location/P3608_fw_image_file target
```

Where:

/location/P3608_fw_image_file

Specifies the path to the Intel P3608 flash image

target

Specifies a single node with SSDs, a comma-separated list of nodes (with SSDs), or the keyword `all_service`

For example:

```
boot:# /opt/cray/ssd-flash/bin/xtiossdflash -f -i /tmp/FW1B0_BL133 c0-1c0s9n2
```

If the firmware updates successfully, one of the following messages is displayed.

- `Firmware application requires conventional reset.`
- `Firmware application requires NVM subsystem reset.`

If the firmware does not update successfully, one of the following messages is displayed.

- `No devices available` - No /dev/nvmeX devices were found.
- The file /root/8DV101B0_8B1B0133_signed.bin does not exist. Skipping. - The firmware image flash file could not be found.
- `Could not find image file compatible with /dev/nvmeX. Skipping` - The device exists, but no firmware image was found that matches the device.
- `/dev/nvmeX already flashed to firmware version <version>. Skipping` - The device is already flashed to the firmware selected.

If applicable, rectify the problem and try again.

4. Reboot the *`target`* node(s) to load the new firmware.

```
boot# xtbootsys --reboot c0-1c0s9n2
```

5. Verify the flash version.

The firmware version displayed below is for example purposes only and should not be expected.

```
boot# /opt/cray/ssd-flash/bin/xtiossdflash -v c0-1c0s9n2
c0-1c0s9n2: <nvme_flash>: /dev/nvme3: Model = INTEL SSDPECME040T4Y , FW Version = 8DV101B0
c0-1c0s9n2: <nvme_flash>: /dev/nvme2: Model = INTEL SSDPECME040T4Y , FW Version = 8DV101B0
c0-1c0s9n2: <nvme_flash>: /dev/nvme1: Model = INTEL SSDPECME040T4Y , FW Version = 8DV101B0
c0-1c0s9n2: <nvme_flash>: /dev/nvme0: Model = INTEL SSDPECME040T4Y , FW Version = 8DV101B0
```

7.2.1 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).
```

8 Deconfigure DataWarp with DWS

Prerequisites

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

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 DWS-managed data.
   a. Remove the LVM volume group used by the DWS.

```
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 46 through *6* on page 46 for all SSD nodes listed in `datawarp_manager_nodes`.

DataWarp with DWS is deconfigured.

9 Proper Swap Server and SSD Shutdown for FusionIO Cards

About this task

The swap servers must shut down prior to shutting down the FusionIO SSD devices, and the internal software running on the SSDs must properly shut itself down when the Cray system is shutting down. There are two methods, manual or automatic, of ensuring that this occurs.

Procedure

Manual method:

1. Shut down swap on the compute nodes.

   On any compute node set up to do swap, run:

   ```
   nid:# /etc/init.d/swap stop
   ```

   Or, from the boot node, run:

   ```
   boot:# pcmd –n nid[,nid,...] /etc/init.d/swap stop
   ```

   IMPORTANT: Watch xtconsole to verify that the swap server has shut down.

2. Watch xtconsole to verify that the SSD drivers unloaded. This must occur prior to answering y/n to the "clear alerts before halting" prompt.

Automatic method:

3. Modify the shutdown script `/opt/cray/hss/default/etc/auto.xtshutdown` and add the following two lines for each SSD node:

   ```
   lappend actions { crms_exec_on_bootnode "root" "ssh cname /bin/umount mount_point }
   lappend actions { crms_exec_on_bootnode "root" "ssh cname /etc/init.d/iscsi-target stop" }
   lappend actions { crms_exec_on_bootnode "root" "ssh cname /etc/init.d/iomemory-vsl stop" }
   ```

   For example:

   ```
   ...
   set actions {}
   lappend actions { crms_exec "xtcli shutdown $data(idlist)" }
   lappend actions { crms_exec "xtcli shutdown $data(idlist)" }
   lappend actions { crms_exec_on_bootnode "root" "ssh c0-0c1s7n1 /bin/umount /flash/scratch1 }
   lappend actions { crms_exec_on_bootnode "root" "ssh c0-0c1s7n2 /bin/umount /flash/scratch1 }
   lappend actions { crms_exec_on_bootnode "root" "ssh c0-0c1s7n1 /etc/init.d/iscsi-target stop" }
   lappend actions { crms_exec_on_bootnode "root" "ssh c0-0c1s7n2 /etc/init.d/iscsi-target stop" }
   lappend actions { crms_exec_on_bootnode "root" "ssh c0-0c1s7n1 /etc/init.d/iomemory-vsl stop" }
   lappend actions { crms_exec_on_bootnode "root" "ssh c0-0c1s7n2 /etc/init.d/iomemory-vsl stop" }
   lappend actions { crms_exec_on_bootnode "root" "xtshutdown -y" }
   lappend actions { crms_sleep 10 }
   ```

```
lappend actions { crms_exec_on_bootnode "root" "/sbin/shutdown -h now" }
lappend actions { crms_ssh_close }
lappend actions { crms_sleep 20 }
lappend actions { my_halt }
```

IMPORTANT: Save a copy of `/opt/cray/hss/default/etc/auto.xtshutdown` as it may get overwritten when the SMW software is updated.

10 Troubleshooting FusionIO Card Issues

In addition to the standard practices for troubleshooting a block device or file system, the following tools may provide additional information to help determine a cause of failure.

The `fio-status` Command

This command is a proprietary tool included with the SSD driver and is available on the DataWarp service node.

```
# fio-status -a
```

The `-a` option specifies that all available data from the SSDs connected to this node is outputted.

The FusionIO SSD Driver

When the FusionIO SSD driver initially loads, it outputs information to the console of the service node. The driver can be manually reloaded to obtain this information again by invoking the `init.d` script.

CAUTION: This shuts down and restarts all SSDs attached to the node; ensure that no other nodes are using or mounting the relevant file systems.

- For VSL3:

  ```
  nid:~ # /etc/init.d/iomemory-vsl restart
  ```

- For VSL4:

  ```
  nid:~ # /etc/init.d/iomemory-vsl4 restart
  ```

11 Static DataWarp: Repair a Volume Group when an SSD Fails

Prerequisites

`/etc/lvm` must be bind mounted to a writable location.

About this task

If a logical volume spans multiple SSDs and one of the SSD fails, the logical volume on the downed SSD is lost and must be removed along with the failed SSD. Note that this procedure uses the `/dev/fio`*X* style of device naming, e.g., `/dev/fioa` or `/dev/fiob`.

Alternatively, all LVM information can be removed from the remaining SSDs and the configuration procedure restarted.

Procedure

1. Remove the logical volume and failed physical volume from the volume group.

   ```
   nid:# vgreduce --removemissing --force vg_name
   ```

2. Add a new SSD.

3. Initialize the device.

   ```
   nid:# pvcreate device
   ```

4. Add the new physical volume to the volume group.

   ```
   nid:# vgextend vg_name device
   ```

5. Recreate the logical volume as was done during initial install.

12 Configure Boot Automation for DataWarp

About this task

When Cray DataWarp is enabled and configured on the system, edit the boot automation file to add a required kernel boot parameter for the DataWarp manager nodes for increased performance.

Procedure

1. Edit the boot automation file.

```
crayadm@smw:~> vi /opt/cray/hss/default/etc/auto.xthostname
```

2. Add the DataWarp manager nodes with SSDs to the boot automation file. Add these commands **after** the service database (SDB) node and any Lustre or DVS service nodes on which the DataWarp nodes are dependent.

   - If the workload manager is Slurm:

```
lappend actions { crms_sleep 5 }
lappend actions [list crms_boot_loadfile SNL0 service node_list linux \
"numa=fake=2 active_wlm=SLURM"]
```

   - For all other workload managers:

```
lappend actions { crms_sleep 5 }
lappend actions [list crms_boot_loadfile SNL0 service node_list linux \
numa=fake=2]
```

   Where *node_list* is either a space- or comma-separated list of DataWarp manager node cnames.

   Further guidance for booting these nodes

   After the system is booted using a boot automation file containing these types of DataWarp edits, the `xtbootsys --reboot` command reboots the DataWarp manager nodes and maintains the `numa=fake=2` or `"numa=fake=2 active_wlm=SLURM"` kernel parameters (as long as no other types of nodes are rebooted on the same command line).

   If the DataWarp manager nodes are rebooted manually using `xtcli boot` directly, these parameters are not preserved and must be respecified. When a manual boot is needed, be sure to pass the parameters on the `xtcli` command line. For example:

   - If the workload manager is Slurm:

```
crayadm@smw> xtcli boot SNL0 c0-0c0s3n2 -- numa=fake=2 active_wlm=SLURM
```

   - For all other workload managers:

```
crayadm@smw> xtcli boot SNL0 c0-0c0s3n2 -- numa=fake=2
```

13 Prefixes for Binary and Decimal Multiples

The International System of Units (SI) prefixes and symbols (e.g., **k**ilo-, **M**ega-, **G**iga-) are often used interchangeably (and incorrectly) for decimal and binary values. This misuse not only causes confusion and errors, but the errors compound as the numbers increase. In terms of storage, this can cause significant problems. For example, consider that a kilobyte (10^3) of data is only 24 bytes less than 2^{10} bytes of data. Although this difference may be of little consequence, the table below demonstrates how the differences increase and become significant.

To alleviate the confusion, the International Electrotechnical Commission (IEC) adopted a standard of prefixes for binary multiples for use in information technology. The table below compares the SI and IEC prefixes, symbols, and values.

SI decimal vs IEC binary prefixes for multiples					
SI decimal standard			IEC binary standard		
Prefix (Symbol)	**Power**	**Value**	**Value**	**Power**	**Prefix (Symbol)**
kilo- (kB)	10^3	1000	1024	2^{10}	kibi- (KiB)
mega- (MB)	10^6	1000000	1048576	2^{20}	mebi- (MiB)
giga- (GB)	10^9	1000000000	1073741824	2^{30}	gibi- (GiB)
tera- (TB)	10^{12}	1000000000000	1099511627776	2^{40}	tebi- (TiB)
peta- (PB)	10^{15}	1000000000000000	1125899906842624	2^{50}	pebi- (PiB)
exa- (EB)	10^{18}	1000000000000000000	1152921504606846976	2^{60}	exbi- (EiB)
zetta- (ZB)	10^{21}	1000000000000000000000	1180591620717411303424	2^{70}	zebi- (ZiB)
yotta- (YB)	10^{24}	1000000000000000000000000	1208925819614629174706176	2^{80}	yobi- (YiB)

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