XC™ Series DVS Administration Guide

(CLE 6.0.UP06)

S-0005

Contents

1 About the XC™ Series DVS Administration Guide

The *XC™ Series DVS Administration Guide* (S-0005) describes the Cray Data Virtualization Service (DVS) and provides guidance on how to configure it to project external file systems.

Release CLE 6.0.UP05

This publication supports Cray software release CLE 6.0.UP06, released March 2018.

Audience and Scope

This publication is intended for site personnel who administer and/or configure DVS on Cray XC™ Series systems.

Command Prompt Conventions

Host name and account in command prompts

The host name in a command prompt indicates where the command must be run. The account that must run the command is also indicated in the prompt.

- The `root` or super-user account always has the `#` character at the end of the prompt.
- Any non-`root` account is indicated with *`account@hostname>`*. A user account that is neither `root` nor `crayadm` is referred to as `user`.

`smw#`	Run the command on the SMW as `root`.
`cmc#`	Run the command on the CMC as `root`.
`sdb#`	Run the command on the SDB node as `root`.
`crayadm@boot>`	Run the command on the boot node as the `crayadm` user.
`user@login>`	Run the command on any login node as any non-`root` user.
`hostname``#`	Run the command on the specified system as `root`.
`user@`*`hostname`*`>`	Run the command on the specified system as any non-`root` user.
`smw1#` `smw2#`	For a system configured with the SMW failover feature there are two SMWs—one in an active role and the other in a passive role. The

	SMW that is active at the start of a procedure is smw1. The SMW that is passive is smw2.
`smwactive#` `smwpassive#`	In some scenarios, the active SMW is smw1 at the start of a procedure—then the procedure requires a failover to the other SMW. In this case, the documentation will continue to refer to the formerly active SMW as smw1, even though smw2 is now the active SMW. If further clarification is needed in a procedure, the active SMW will be called smwactive and the passive SMW will be called smwpassive.

Command prompt inside chroot

If the `chroot` command is used, the prompt changes to indicate that it is inside a chroot environment on the system.

```
smw# chroot /path/to/chroot
chroot-smw#
```

Directory path in command prompt

Example prompts do not include the directory path, because long paths can reduce the clarity of examples. Most of the time, the command can be executed from any directory. When it matters which directory the command is invoked within, the `cd` command is used to change into the directory, and the directory is referenced with a period (`.`) to indicate the current directory.

For example, here are actual prompts as they appear on the system:

```
smw:~ # cd /etc
smw:/etc# cd /var/tmp
smw:/var/tmp# ls ./file
smw:/var/tmp# su - crayadm
crayadm@smw:~> cd /usr/bin
crayadm@smw:/usr/bin> ./command
```

And here are the same prompts as they appear in this publication:

```
smw# cd /etc
smw# cd /var/tmp
smw# ls ./file
smw# su - crayadm
crayadm@smw> cd /usr/bin
crayadm@smw> ./command
```

Typographic Conventions

`Monospace`	Indicates program code, reserved words, library functions, command-line prompts, screen output, file/path names, 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 and key strokes (e.g., **Enter**, **Alt-Ctrl-F**).

\ (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.

Trademarks

2 Introduction to DVS

Cray Data Virtualization Service (DVS) is a distributed network service that projects local file systems resident on I/O nodes or remote file servers to compute and service nodes within the Cray system. Projecting is simply the process of making a file system available on nodes where it does not physically reside. DVS-specific configuration settings enable clients (compute nodes) to access a file system projected by DVS servers. Thus, Cray DVS, while not a file system, represents a software layer that provides scalable transport for file system services. See the `mount(8)` and `dvs(5)` man pages for more information.

DVS Use Cases

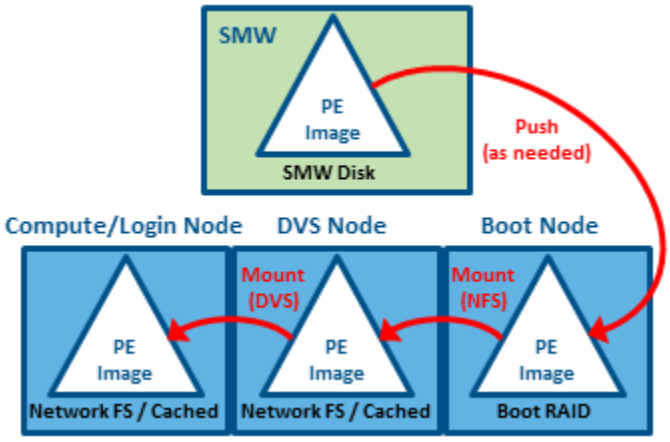
DVS plays an essential role in the new Cray management system (CMS) paradigm.

- DVS is tightly coupled with the Simple Shares service.
- During system boot, Netroot is mounted as a DVS mount.
- DVS is used by the `cray_image_binding` service to distribute the Cray programming environment (PE) to compute and login nodes.

Cray automatically enables DVS and configures it with the settings necessary to achieve this functionality. Sites are not required to configure DVS further, which is why this service is level `basic` rather than `required`.

This figure illustrates how the system uses DVS to distribute PE images to compute and login nodes. The PE image is stored on the SMW and a copy is pushed out to the boot node and stored on its local storage. The boot node shares the PE image with the DVS node using NFS, and then the DVS node projects it to compute and login nodes using DVS.

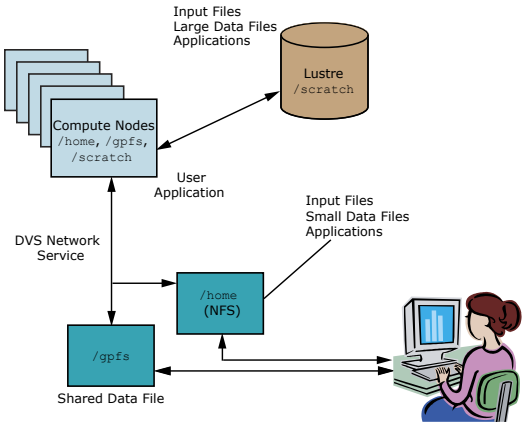
Figure 1. PE Distribution to Compute and Login Nodes



DVS needs further configuration only if a site plans to use it to project external file systems to nodes within the Cray system. The `cray_dvs` service configuration parameters enable system administrators to provide their

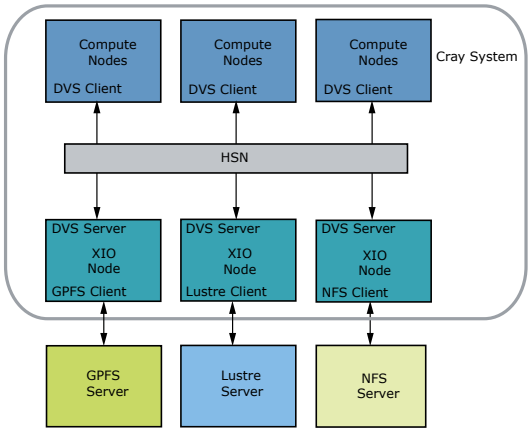
users with client mounts that can be tuned for high performance in a variety of use cases. When projecting external file systems, DVS provides I/O performance and scalability to a large number of nodes, far beyond the typical number of clients supported by a single NFS server. Operating system noise and impact on compute node memory resources are both minimized in the Cray DVS configuration. Cray DVS uses the Linux-supplied virtual file system (VFS) interface to process file system access operations. This allows DVS to project any POSIX-compliant file system. Cray has extensively tested DVS with NFS and General Parallel File System (now Spectrum Scale).

Figure 2. Cray DVS Projection of External File Systems



When DVS is used to project external file systems, an administrator's view of Cray DVS looks like this.

Figure 3. Cray DVS In a Cray System



3 DVS Modes

A DVS mode is simply the name given to a combination of mount options used to achieve a particular goal. DVS has two primary modes of use: serial and parallel. In serial mode, one DVS node projects a file system to multiple compute node clients. In parallel mode, multiple DVS nodes—in configurations that vary in purpose, layout, and performance—project a file system to multiple compute node clients. Those varying configurations give rise to several flavors of parallel mode.

The availability of these DVS modes is determined by the system administrator's choice of DVS mount options during system configuration. Users cannot choose among DVS modes unless the system administrator has configured the system to make more than one mode available. A system administrator can make several DVS modes available on the same compute node by mounting a file system with different mount options on different mount points on that compute node. Here is a summary of the rationale and example configuration settings for each DVS mode. Note that these modes represent only some of the possible ways to configure DVS. There are many other mount options available.

In the "Example Configuration Settings" column, the `server_groups` setting is a list of node groups. See the entry for `server_groups` in *DVS Configuration Settings and Mount Options* on page 55 for more information.

Mode	Rationale	Example Configuration Settings

DVS Serial Mode	Simplest implementation of DVS. Only option if no cluster/shared file system available.	`server_groups: [dvs_server_serial]`
		`options: maxnodes=1`
		(*`dvs_server_serial`* is a node group that has a single member, such as c0-0c1s1n1)

DVS Cluster Parallel Mode	Often used for a large file system, must be a shared file system such as GPFS (Spectrum Scale). Can distribute file I/O and metadata operations among several servers to avoid overloading any one server and to speed up operations. I/O for a single file goes only to the chosen server.	`server_groups: [dvs_servers_parallel]`
		`options: maxnodes=1`
		(*`dvs_servers_parallel`* is a node group that has several members, such as c0-0c1s1n1, c0-0c1s1n2, c0-0c0s2n1)

DVS Stripe Parallel Mode	Used to distribute file I/O load at the granularity of a block of data within a file. Adds another level of parallelism to better distribute the load. I/O for a single file may go to multiple servers.	`server_groups: [dvs_servers_parallel]` `options: maxnodes=3`
DVS Atomic Stripe Parallel Mode	Used when stripe parallel makes sense and POSIX read/write atomicity required.	`server_groups: [dvs_servers_parallel]` `options: maxnodes=3,atomic`

Mode	Rationale	Example Configuration Settings
DVS Loadbalance Mode	Used for near-optimal load distribution when a read-only file system is being used. By default, enables `readonly` and sets `cache=1`, `failover=1`, `maxnodes=1`, and `hash_on_nid=0`.	`server_groups: [dvs_servers_parallel]` `loadbalance: true`

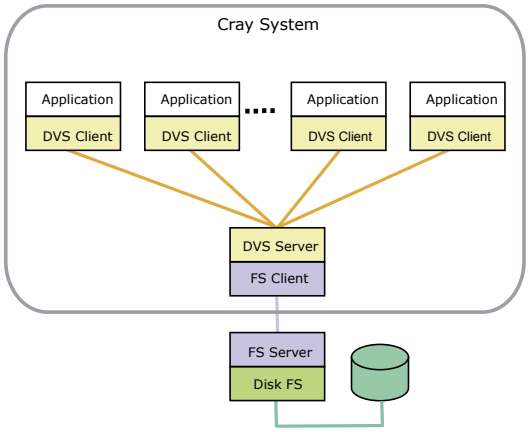
Serial, cluster parallel, and atomic stripe parallel modes all adhere to POSIX read/write atomicity rules, but stripe parallel mode does not. POSIX read/write atomicity guarantees that all bytes associated with a read or write are not interleaved with bytes from other read or write operations.

DVS Serial Mode

Serial mode is the simplest implementation of DVS, where each file system is projected from a single DVS server (node) to multiple clients (compute nodes). DVS can project multiple file systems in serial mode from the same or different DVS nodes by entering `maxnodes=1` in the `options` configuration setting for each client mount set up during configuration or reconfiguration.

DVS serial mode adheres to POSIX read/write atomicity rules.

Figure 4. Cray DVS Serial Access Mode

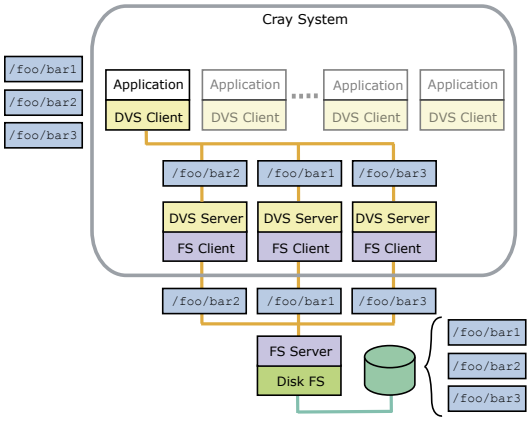


DVS Cluster Parallel Mode

In cluster parallel mode, each client interacts with multiple servers. For example, in the figure below, DVS is mounted to `/foo` on the DVS client, and three different files—`bar1`, `bar2`, and `bar3`—are handled by three different DVS servers (nodes), thus distributing the load. The server used to perform a file's I/O or metadata operations is selected using an internal hash involving the underlying file or directory inode number. Once a server has been selected for a file, cluster parallel mode looks like serial mode: all of that file's I/O and metadata operations from all clients route to the selected server to prevent file system coherency thrash.

DVS cluster parallel mode adheres to POSIX read/write atomicity rules.

Figure 5. Cray DVS Cluster Parallel Access Mode



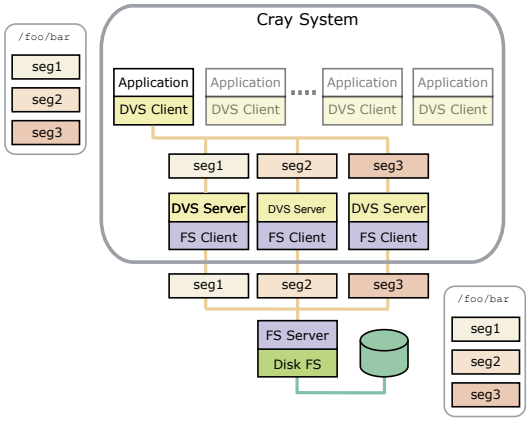
DVS Stripe Parallel Mode

Stripe parallel mode builds upon cluster parallel mode to provide an extra level of parallelized I/O forwarding for clustered file systems. Each DVS server (node) can serve all files, and DVS servers are automatically chosen based on the file inode and offsets of data within the file relative to the DVS block size value (`blksize`). For example, in the figure below, DVS is mounted to `/foo` on the DVS client, and the I/O for three different blocks (or segments) of data within file `bar`—`seg1`, `seg2`, and `seg3`—is handled by three different DVS servers, thus distributing the load at a more granular level than that achieved by cluster parallel mode. All I/O from all clients involving the same file routes each block of file data to the same server to prevent file system coherency thrash. Note that while file I/O is distributed at the block level, file metadata operations are distributed as in cluster parallel mode: the metadata operations of a given file are always handled by the same DVS server. Stripe parallel mode provides the opportunity for greater aggregate I/O bandwidth when forwarding I/O from a coherent cluster file system. GPFS (Spectrum Scale) has been tested extensively using this mode.

> **ATTENTION:** NFS cannot be used in stripe parallel mode because NFS implements close-to-open cache consistency; therefore striping data across the NFS clients could compromise data integrity.

DVS stripe parallel mode **does not** adhere to POSIX read/write atomicity rules.

Figure 6. Cray DVS Stripe Parallel Mode



DVS Atomic Stripe Parallel Mode

Stripe parallel mode provides parallelism within a file at the granularity of the DVS block size. However, when applications do not use their own file locking, stripe parallel mode cannot guarantee POSIX read/write atomicity. In contrast, atomic stripe parallel mode adheres to POSIX read/write atomicity rules while still allowing for possible parallelism within a file. It is similar to stripe parallel mode in that the server used to perform the I/O or metadata operation is selected using an internal hash involving the underlying file or directory inode number, and the offset of data into the file is relative to the DVS block size. However, once that server is selected, the entire read or write request is handled by that server only. This ensures that all I/O requests are atomic while allowing DVS clients to access different servers for subsequent I/O requests if they have different starting offsets within the file.

Users can request atomic stripe parallel mode by setting the `DVS_ATOMIC` user environment variable to `on`.

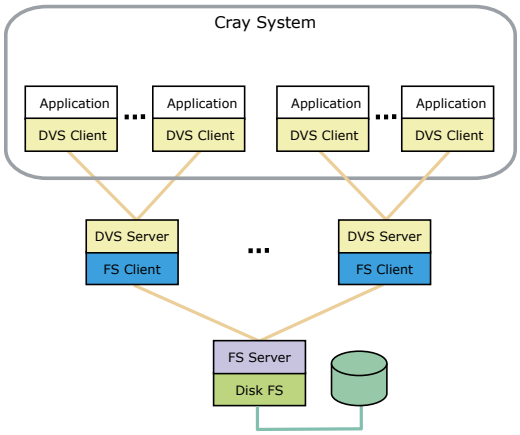
DVS Loadbalance Mode

Loadbalance mode is used to more evenly distribute loads across servers. The clients, Cray system compute nodes, automatically select the server based on a DVS-internal node ID (NID) from the list of available server nodes specified in the `servers` setting within the configurator or configuration worksheet. When `loadbalance` is enabled, the underlying DVS implementation automatically sets the `readonly` setting to `true` and sets these additional options: `cache=1`, `failover=1`, `maxnodes=1`, and `hash_on_nid=0`.

To enable attribute caching as well, set the `attrcache_timeout` setting for `loadbalance` client mounts (this is a separate configuration setting within the configurator or configuration worksheet). This allows attribute-only file system operations to use local attribute data instead of sending the request to the DVS server. This is useful in loadbalance mode because with a read-only file system, attributes are not likely to change.

DVS automatically enables the `cache` mount option in loadbalance mode because using cache on a read-only mount can improve performance. With cache enabled, a DVS client pulls data from the DVS server the first time it is referenced, but then the data is stored in the client's page cache. While the application is running, all future references to that data are local to the client's memory, and DVS will not be involved at all. However, if the node runs low on memory, the Linux kernel may remove these pages, and then the client must fetch the data from the DVS server on the next reference to repopulate the client's page cache.

Figure 7. Cray DVS Loadbalance Mode



4 DVS Configuration and Use

What Needs DVS?

DVS (`cray_dvs`) is required for Netroot[1] to function and for distributing the programming environment (PE) to compute and login nodes using the `cray_image_binding` service. However, sites are not required to make any changes to the `cray_dvs` service to achieve this functionality.

If DVS is used to provide access to DataWarp, some DVS configuration will be required.

What does DVS Need?

For DVS to function properly, the following services may need to be configured. The first two are required for a functional system regardless of DVS, but they are included because DVS-specific information or a consideration of how DVS will be used is necessary during their configuration.

`cray_scalable_services`	Defines which servers are used in the scaling of the system. When configuring the `cray_scalable_services` service, ensure that any DVS servers/nodes that will be used to project external file systems are NOT added to the list of tier2 servers.
`cray_net`	Configures key network attributes. When configuring the `cray_net` service, set network values to define a network interface for any DVS servers/nodes that will be used to project external file systems. Needed only for communications using anything other than the Cray Aries high-speed network (HSN).
`cray_lnet`	DVS uses Lustre™ Networking (LNet) to communicate on Aries networks, so even if Lustre is not used on this system, the LNet service must be configured. No DVS-specific settings are necessary; it is sufficient to ensure that the `cray_lnet` service is enabled.
`cray_multipath`	Detects and coalesces multiple paths to devices. If the node acting as the DVS server is natively mounting the file system, it has access to the storage directly, and if more than one path to the storage is desired, then the `cray_multipath` service needs to be enabled. This is not typical; DVS is usually a client of the file system.

Node Groups

To configure DVS, it is necessary to define at least one DVS node group that contains one or more DVS servers. Sites that are doing a fresh install will have an opportunity to define DVS node groups during the installation process. Sites that are updating CLE 6.0.UP01 software to the current release, and had already defined one or

[1] Netroot is a mechanism to enable nodes booted with a minimal, local in-memory file system to execute within the context of a larger, full-featured root file system. For more information, see *XC™ Series Software Installation and Configuration Guide* (S-2559).

more DVS node groups, will have an opportunity to migrate that node group data to the `cray_dvs_worksheet` for the new release. Information about defining and migrating node groups data is found in the *XC™ Series Software Installation and Configuration Guide* (S-2559).

See *About Node Groups* on page 105 for more information about node groups in general.

Procedures to Configure DVS using the Configurator

DVS is one of many services that store service configuration content in CLE configuration sets (config sets) on Cray systems. DVS can be configured when config sets are created during a fresh install or major upgrade, or it can be configured/reconfigured later by updating existing config sets during normal system operation (bearing in mind that some of the DVS module parameters are best set during initial system configuration). These procedures guide site administrators and staff in entering appropriate values for DVS configuration settings using the configurator. Whether sites enter values in an interactive configurator session or enter values in a configuration worksheet for bulk import, the configurator takes the supplied values and ensures that they become part of the config set being created or updated.

Use one of the following procedures to configure or reconfigure DVS using the configurator. In all procedures, steps correspond to order of settings as encountered in an interactive session or configuration worksheet.

Configure DVS using the Configurator	General-purpose procedure, emphasis on decision support. • For both initial configuration and reconfiguration • Detailed description of settings • No examples
Configure DVS using Worksheets	Procedure for editing the DVS worksheet, emphasis on examples. • For initial configuration or major update • Detailed steps and examples
Reconfigure DVS Interactively	Procedure for using the configurator interactively, emphasis on examples. • For minor updates • Detailed steps and examples

Caveat: The above procedures do not cover how to use `cfgset` and the configurator (which is invoked by `cfgset`). See the `cfgset` man page and the *XC™ Series Configurator User Guide* (S-2560).

Procedure to Configure DVS using Modprobe.d, Proc Files, and Simple Sync

At this time, there are some DVS configuration parameters that cannot be set using the configurator. For such cases, configuration files can be created within the config set and distributed throughout the system using the Simple Sync mechanism, part of the new Cray management system. It is important to make these changes within the config set directory structure, otherwise changes may not persist. For instructions on how to do that, use this procedure.

Configure DVS using Modprobe or Proc Files on page 30

For more information about Simple Sync, see *About Simple Sync* on page 100. For general information about the way Cray XC systems are configured, see *Cray XC System Configuration* on page 93.

Procedure to Apply Configuration Changes

After using one of the configuration procedures or making any configuration changes, validate the revised config set and run Ansible plays to propagate and apply the configuration changes.

Validate the Config Set and Run Ansible Plays on page 43

Procedures to Manage and Optimize DVS Use

Use the following procedures, as needed, during operation or for further configuration.

- *Quiesce a DVS-projected File System* on page 44
- *DVS Client-side Write-back Caching can Yield Performance Gains* on page 46
- *Force a Cache Revalidation on a DVS Mount Point* on page 49
- *Disable DVS Fairness of Service* on page 50
- *Reconfigure DVS for an External NFS Server* on page 52
- *Improve Performance and Scalability of GPFS (Spectrum Scale) Mounts* on page 54
- For procedures to configure a DVS mount of GPFS (Spectrum Scale), see *XC Series GPFS Software Installation Guide* (S-2569), which is available at *http://pubs.cray.com*

4.1 Configure DVS using the Configurator

Prerequisites

This procedure assumes that the user is either in a configurator interactive session with the DVS service (`cray_dvs`) selected or is editing the DVS configuration worksheet (`cray_dvs_worksheet.yaml`).

- For instructions on how to start a configurator session, see step 2 of *Reconfigure DVS Interactively* on page 25.
- For instructions on how to start editing the DVS configuration worksheet, see the prerequisites of *Configure DVS using Worksheets* on page 19

About this task

The following steps correspond to the DVS configuration settings available through the configurator, and step numbering reflects the order in which those settings are presented when using the configurator in auto mode. They can be accessed in any order if using the configurator in interactive mode or if entering data in the configuration worksheet.

CAUTION: Configure DVS and all services **only** through the configurator or by placing/editing configuration files in the Simple Sync directory structure within the config set. Do not configure DVS by manually adding lines to `/etc/fstab` or other `/etc` files. In general, changes to those files are not persistent, and rebooting could result in loss of data.

IMPORTANT: When configurator guidance indicates a relationship or interaction between one or more settings, it is advisory only; the configurator does not automatically check to ensure compatibility among settings. However, the underlying implementation of DVS is unchanged, and it **does** automatically set

related mount options when certain mount options are specified. To prevent mount failure, enter setting values that are compatible, in accordance with the instructions in this publication.

Procedure

1. Enable Cray DVS service.

   Ensure that `cray_dvs.enabled` is set to `true`.

2. Configure a DVS client mount. Repeat this step to configure multiple client mounts, as needed.

   A client mount defines the file system shares to be projected from DVS nodes to all (or selected) compute nodes. Each client mount is specified by a multival setting, which has a single key field followed by one or more other fields. The following substeps correspond to the fields (settings) of a client mount setting; substep numbering reflects the order in which these settings are presented when using the configurator in auto mode.

   a. Set the client mount reference.

   `reference` — A human-readable string—a name—that is used to uniquely identify a client mount. `reference` cannot be set by accepting the default: a non-empty string is required.

   Related settings/options: Because this is the key field of a client mount setting entry, each setting within the client mount setting includes this string in its full setting name.

   b. Set the pathname of the mount point on the client.

   `mount_point` — A string that specifies the full pathname on the client of the projected file system. `mount_point` cannot be set by accepting the default: a non-empty string is required.

   Related settings/options: none

   c. Set the pathname of the mount point on the DVS server node.

   `spath` — A string that specifies the full pathname on the DVS server of the file system that is to be projected for a client mount. It must be an absolute path and it must exist on the DVS server. `spath` cannot be set by accepting the default: a non-empty string is required.

   Related settings/options: none

   d. Set the list of DVS server groups.

   `server_groups` — A list of node groups that will function as DVS servers for a client mount. Enter one node group per line (see *About Node Groups* on page 105). `server_groups` cannot be set by accepting the default: a non-empty list is required.

   IMPORTANT: DVS servers should be dedicated because they use unlimited amounts of CPU and memory resources based directly on the I/O requests sent from DVS clients. Avoid using nodes that have other services (Lustre nodes, login nodes, etc.) or are tier2 nodes.

Related settings/options: Functionally equivalent to the `nodename` or `nodefile` "additional" option in the `options` setting of the client mount setting. The use of those two additional options is deprecated.

e. Set the list of DVS client groups.

`client_groups` A list of node groups that will function as DVS clients for a client mount. Enter node groups one per line. Unlike `server_groups`, `client_groups` can be set to an empty list. If no node groups are specified, the mount will be performed on all suitable compute nodes (a compute node functioning as a DVS server is an example of an unsuitable node). This is common.

Related settings/options: none

f. Set the loadbalance option.

`loadbalance` Used to specify loadbalance mode, which more evenly distributes loads across DVS servers. Loadbalance mode is valid only for read-only mounts. For more information, see *DVS Loadbalance Mode*.

Related settings/options: When `loadbalance` is enabled, the underlying DVS implementation automatically sets the `readonly` setting to `true` and sets these additional options: `cache=1`, `failover=1`, `maxnodes=1`, and `hash_on_nid=0`. Cray recommends setting the `attrcache_timeout` setting as well to take advantage of the mount being read-only. If `loadbalance` is enabled, leave the `readonly` setting unconfigured or set it to `true` to maintain consistency with the way DVS implements `loadbalance`.

g. Set the attribute cache timeout option.

`attrcache_timeout` Enables client-side attribute caching, which can significantly increase performance, most notably in pathname lookup situations. When attribute caching is disabled, DVS clients must send a lookup request to a DVS server for every level of a pathname, and repeat this for every pathname operation. When it is enabled, it sends a lookup request to a DVS server for every level of a pathname once per *n* seconds.

The configurator default is `14400` seconds. The underlying DVS implementation default is `3` seconds, which is safer for read-write mounts. This means that to enhance system performance for read-only mounts, configure this setting by accepting the configurator default (or entering some other value). Leaving this setting unconfigured will result in the underlying default being used.

Related settings/options: The Ansible play that consumes DVS configuration data prevents use of this mount option for read-write file systems due to the risk of file system corruption. Run-time mounts not accompanied by that Ansible play do not have that safeguard. In such cases, if a read-write mount is created, it is safe to leave `attrcache_timeout` unconfigured so that the underlying default is used.

h. Set the read-only option.

`readonly` Determines whether the client mount is read-only or read-write. If intending to enable client-side caching of read data on a non-writable file system, use this

`readonly` setting to force the DVS mount to be read-only. This will disable write caching.

The configurator default is `true` or `1`. The underlying DVS implementation default is `false`. Leaving this setting unconfigured will result in the underlying DVS default being used.

Related settings/options: When `loadbalance` is enabled, DVS automatically enables `readonly` but the configurator does not, so either leave this setting unconfigured or accept the configurator default. If the `attrcache_timeout` setting is set for this client mount, `readonly` should be enabled (set to true) in the configurator/worksheet. If the `cache` option is specified in the `options` setting for this client mount, enabling `readonly` is the only way to enable read caching without enabling write caching as well).

i. Set other client mount options.

`options`

Provides the only way to specify mount options in addition to the ones already specified in the other mount point settings. Enter a string with mount options separated by comma and no spaces. For information about available options and their implications, see *Additional Options for Use in the Options Setting of a Client Mount*. Note that it is necessary to specify `maxnodes=1` here for a read-write client mount of an NFS or other non-cluster, non-coherent file system.

Related settings/options: Options contained in this setting will be appended to the mount options specified in other settings. Any that are functionally redundant with settings already configured (such as *`nodename`*/*`nodefile`*, which are redundant with the `server_groups` setting) will override those settings.

This completes the entry of a client mount setting. Review the setting fields in the configurator summary or configuration worksheet. Revisit any of the settings to make changes, if needed. Repeat *Step 2: Configure a DVS client mount* to specify additional client mounts.

3. Set DVS kernel module parameters.

Two DVS kernel module parameters can be set using the configurator by following the substeps below. Note that changing these parameters after initial configuration may require reloading the module to enable the change to take effect. To change other kernel module parameters, see *Configure DVS using Modprobe or Proc Files* on page 30.

a. Set the DVS interprocess communication (IPC) heartbeat timeout.

`dvsipc_heartbeat_timeout`

DVS inter-process communication (IPC) heartbeat timeout, in seconds. This parameter is no longer used; it has been preserved only to maintain backwards compatibility with existing DVS config files. Leave this parameter unconfigured or accept the default value.

Related settings/options: none

b. Set the DVS debug mask.

`dvs_debug_mask`

Hex mask of the bits to set to enable debug output to be printed to the console. It can flood the console file and negatively affects performance, so it

is generally used only for development or troubleshooting. Different mask values enable the output of different sets of debug information. Leave this parameter unconfigured or accept the default value.

Related settings/options: none

4. When done configuring DVS, take one of the following actions:

If configuring DVS	Do this
interactively	Save the changes and exit the configurator session. `Cray DVS Service Menu [default: save & exit - Q] $ Q` `cfgset` will save all of the configuration settings in the config set (in configuration worksheets and templates), run post-configuration scripts, create a time-stamped clone of the config set, and exit automatically.

using the worksheet	Import the completed DVS worksheet by updating the config set and specifying the worksheet path.
	`smw# cfgset update --worksheet-path \`
	`'/some/edit/location/cray_dvs_worksheet.yaml' p0`

5. Proceed to *Validate the Config Set and Run Ansible Plays* on page 43.

4.2 Configure DVS using Worksheets

Prerequisites

This procedure assumes that the user has generated configuration worksheets and is editing the DVS configuration worksheet (`cray_dvs_worksheet.yaml`). If new worksheets need to be generated, use this procedure:

1. Generate up-to-date worksheets for config set *p0* (merges any new service packages installed on the system with data already in config set *p0*).

   ```
   smw# cfgset update --mode prepare --no-scripts p0
   ```

2. Make a copy of the CLE configuration worksheets directory outside the config set to be used as a work area for editing. The worksheets should not be edited in their original location for two reasons: (1) the configurator will not permit updating a config set from worksheets within that config set, and (2) edits would be overwritten when the config set is updated.

   ```
   smw# cp /var/opt/cray/imps/config/sets/p0/worksheets/* \
   /var/adm/cray/release/p0_worksheet_workarea
   ```

   (This is the same work area used in an initial installation of SMW/CLE software.

3. Edit the DVS worksheet:

   ```
   smw# vi /var/adm/cray/release/p0_worksheet_workarea/cray_dvs_worksheet.yaml
   ```

About this task

The following steps correspond to the configuration settings available in the DVS worksheet, and step numbering reflects the order in which those settings appear there.

CAUTION: Configure DVS and all services **only** through the configurator or by placing/editing configuration files in the Simple Sync directory structure within the config set. Do not configure DVS by manually adding lines to `/etc/fstab` or other `/etc` files. In general, changes to those files are not persistent, and rebooting could result in loss of data.

IMPORTANT: When configurator guidance indicates a relationship or interaction between one or more settings, it is advisory only; the configurator does not automatically check to ensure compatibility among settings. However, the underlying implementation of DVS is unchanged, and it **does** automatically set related mount options when certain mount options are specified. To prevent mount failure, enter setting values that are compatible, in accordance with the instructions in this publication.

Procedure

1. Enable Cray DVS service.

   Uncomment the line that sets the `cray_dvs.enabled` setting, and ensure that it is set to `true`.

   ```
   # Enable 'cray_dvs' Service? (boolean, level=basic)
   cray_dvs.enabled: true
   #
   #***************** END Service Enable/Disable ****************
   ```

2. Configure a client mount. Repeat this step to configure multiple mounts, as needed.

   This step defines the file system shares to be projected from DVS nodes to all or selected compute nodes. Each client mount entry is a multival setting, which means it has a single key field followed by one or more other fields.

   In the worksheet, copy the nine lines below `# ** EXAMPLE 'client_mount' VALUE (with current defaults) **` and paste them below the line `# NOTE: Place additional 'client_mount' setting entries here, if desired.` Repeat this for each client mount.

   ```
   # ** EXAMPLE 'client_mount' VALUE (with current defaults) **
   #   cray_dvs.settings.client_mount.data.reference.sample_key_a: null   <-- setting a multival key
   #   cray_dvs.settings.client_mount.data.sample_key_a.mount_point: ''
   #   cray_dvs.settings.client_mount.data.sample_key_a.spath: ''
   #   cray_dvs.settings.client_mount.data.sample_key_a.server_groups: []
   #   cray_dvs.settings.client_mount.data.sample_key_a.client_groups: []
   #   cray_dvs.settings.client_mount.data.sample_key_a.loadbalance: false
   #   cray_dvs.settings.client_mount.data.sample_key_a.attrcache_timeout: 14400
   #   cray_dvs.settings.client_mount.data.sample_key_a.readonly: true
   #   cray_dvs.settings.client_mount.data.sample_key_a.options: ''
   ```

   Uncomment the lines and remove the `<-- setting a multival key` text at the end of the first line (note that the null value is required; do not remove or change it). Finally, modify the values as needed for this site. The substeps that follow provide guidance for changing or keeping the default value for each field. To leave a setting unconfigured, keep/restore the comment symbol at the beginning of the line.

   a. Set the client mount reference.

reference

A human-readable string—a name—that is used to uniquely identify a client mount. `reference` cannot be set by accepting the default: a non-empty string is required. Replace `sample_key_a` with the chosen string (*`REF-NAME`* in example below) in all lines. Do not change the `null` value in the first line.

Related settings/options: Because this is the key field of a client mount setting entry, each setting within the client mount setting includes this string in its full setting name.

```
# NOTE: Place additional 'client_mount' setting entries here, if desired.
cray_dvs.settings.client_mount.data.reference.REF-NAME: null
cray_dvs.settings.client_mount.data.REF-NAME.mount_point: ''
cray_dvs.settings.client_mount.data.REF-NAME.spath: ''
cray_dvs.settings.client_mount.data.REF-NAME.server_groups: []
cray_dvs.settings.client_mount.data.REF-NAME.client_groups: []
cray_dvs.settings.client_mount.data.REF-NAME.loadbalance: false
cray_dvs.settings.client_mount.data.REF-NAME.attrcache_timeout: 14400
cray_dvs.settings.client_mount.data.REF-NAME.readonly: true
cray_dvs.settings.client_mount.data.REF-NAME.options: ''
#*************** END Service Setting: client_mount ***************
```

b. Set the pathname of the mount point on the client.

mount_point

A string that specifies the full pathname on the client of the projected file system. `mount_point` cannot be set by accepting the default: a non-empty string is required. Replace the default empty string with a full pathname.

Related settings/options: none

```
cray_dvs.settings.client_mount.data.REF-NAME.mount_point: /CLIENT-PATH
```

c. Set the pathname of the mount point on the DVS server node.

spath

A string that specifies the full pathname on the DVS server of the file system that is to be projected for a client mount. It must be an absolute path and it must exist on the DVS server. `spath` cannot be set by accepting the default: a non-empty string is required. Replace the default empty string with a pathname that is an absolute path that exists on the DVS server.

Related settings/options: none

```
cray_dvs.settings.client_mount.data.REF-NAME.spath: /DVS-PATHNAME
```

d. Set the list of DVS server nodes.

server_groups

A list of node groups that will function as DVS servers for a client mount. Enter one node group per line (see *About Node Groups* on page 105). `server_groups` cannot be set by accepting the default: a non-empty list is required.

IMPORTANT: DVS servers should be dedicated because they use unlimited amounts of CPU and memory resources based directly on the I/O requests sent from DVS clients. Avoid using nodes that have other services (Lustre nodes, login nodes, etc.) or are tier2 nodes.

Replace the default empty list with one or more node groups; if no node groups are entered, no mount is made.

Related settings/options: Functionally equivalent to the `nodename` or `nodefile` "additional" option in the `options` setting of the client mount setting. The use of those two additional options is deprecated.

```
cray_dvs.settings.client_mount.data.REF-NAME.server_groups:
node-group-1
node-group-2
```

e. Set the list of DVS client nodes.

client_groups A list of node groups that will function as DVS clients for a client mount. Enter node groups one per line. Unlike `server_groups`, `client_groups` can be set to an empty list. If no node groups are specified, the mount will be performed on all suitable compute nodes (a compute node functioning as a DVS server is an example of an unsuitable node). This is common.

Related settings/options: none

```
cray_dvs.settings.client_mount.data.REF-NAME.client_groups:
node-group-3
```

f. Set the loadbalance option.

loadbalance Used to specify loadbalance mode, which more evenly distributes loads across DVS servers. Loadbalance mode is valid only for read-only mounts. For more information, see *DVS Loadbalance Mode*.

Related settings/options: When `loadbalance` is enabled, the underlying DVS implementation automatically sets the `readonly` setting to `true` and sets these additional options: `cache=1`, `failover=1`, `maxnodes=1`, and `hash_on_nid=0`. Cray recommends setting the `attrcache_timeout` setting as well to take advantage of the mount being read-only. If `loadbalance` is enabled, leave the `readonly` setting unconfigured or set it to `true` to maintain consistency with the way DVS implements `loadbalance`.

```
cray_dvs.settings.client_mount.data.REF-NAME.loadbalance: true
```

g. Set the attribute cache timeout option.

attrcache_timeout Enables client-side attribute caching, which can significantly increase performance, most notably in pathname lookup situations. When attribute caching is disabled, DVS clients must send a lookup request to a DVS server for every level of a pathname, and repeat this for every pathname operation. When it is enabled, it sends a lookup request to a DVS server for every level of a pathname once per *n* seconds.

The configurator default is `14400` seconds. The underlying DVS implementation default is `3` seconds, which is safer for read-write mounts. This means that to enhance system performance for read-only mounts, configure this setting by accepting the configurator default (or entering some other

value). Leaving this setting unconfigured will result in the underlying default being used.

Related settings/options: The Ansible play that consumes DVS configuration data prevents use of this mount option for read-write file systems due to the risk of file system corruption. Run-time mounts not accompanied by that Ansible play do not have that safeguard. In such cases, if a read-write mount is created, it is safe to leave `attrcache_timeout` unconfigured so that the underlying default is used.

```
cray_dvs.settings.client_mount.data.REF-NAME.attrcache_timeout: 14400
```

h. Set the read-only option.

readonly

Determines whether the client mount is read-only or read-write. If intending to enable client-side caching of read data on a non-writable file system, use this `readonly` setting to force the DVS mount to be read-only. This will disable write caching.

The configurator default is `true` or `1`. The underlying DVS implementation default is `false`. Leaving this setting unconfigured will result in the underlying DVS default being used.

Related settings/options: When `loadbalance` is enabled, DVS automatically enables `readonly` but the configurator does not, so either leave this setting unconfigured or accept the configurator default. If the `attrcache_timeout` setting is set for this client mount, `readonly` should be enabled (set to true) in the configurator/worksheet. If the `cache` option is specified in the `options` setting for this client mount, enabling `readonly` is the only way to enable read caching without enabling write caching as well).

```
cray_dvs.settings.client_mount.data.REF-NAME.readonly: true
```

i. Set other client mount options.

options

Provides the only way to specify mount options in addition to the ones already specified in the other mount point settings. Enter a string with mount options separated by comma and no spaces. For information about available options and their implications, see *Additional Options for Use in the Options Setting of a Client Mount*. Note that it is necessary to specify `maxnodes=1` here for a read-write client mount of an NFS or other non-cluster, non-coherent file system.

Related settings/options: Options contained in this setting will be appended to the mount options specified in other settings. Any that are functionally redundant with settings already configured (such as *`nodename`*/*`nodefile`*, which are redundant with the `server_groups` setting) will override those settings.

In the example, this client mount point is being set to atomic stripe parallel mode.

```
cray_dvs.settings.client_mount.data.REF-NAME.options: maxnodes=3,atomic
```

This completes the entry of a client mount setting. Review the setting fields and make changes, if needed. Repeat step *2* on page 20 to configure additional client mount points.

3. Set DVS kernel module parameters.

Two DVS kernel module parameters can be set in this worksheet by following the substeps below. Note that changing these parameters after initial configuration may require reloading the module to enable the change to take effect. To change other kernel module parameters, see *Configure DVS using Modprobe or Proc Files* on page 30.

a. Set the DVS interprocess communication (IPC) heartbeat timeout.

dvsipc_heartbeat_timeout	DVS inter-process communication (IPC) heartbeat timeout, in seconds. This parameter is no longer used; it has been preserved only to maintain backwards compatibility with existing DVS config files. Leave this parameter unconfigured or accept the default value. Related settings/options: none

```
# ---------------- kernel_param : dvsipc_heartbeat_timeout ----------------
#
#cray_dvs.settings.kernel_param.data.dvsipc_heartbeat_timeout: 60
```

b. Set the DVS debug mask.

dvs_debug_mask	Hex mask of the bits to set to enable debug output to be printed to the console. It can flood the console file and negatively affects performance, so it is generally used only for development or troubleshooting. Different mask values enable the output of different sets of debug information. Leave this parameter unconfigured or accept the default value. Related settings/options: none

```
# ---------------- kernel_param : dvs_debug_mask ----------------
#
#cray_dvs.settings.kernel_param.data.dvs_debug_mask: 0
```

4. Import the completed DVS worksheet by updating the config set and specifying the worksheet path.

```
smw# cfgset update --worksheet-path \
'/some/edit/location/cray_dvs_worksheet.yaml' p0
```

When the config set is updated using the configurator, all of the pre-and post-configuration scripts are run.

REMEMBER: When importing worksheets using `cfgset` with the `--worksheet-path` option,

- Always add single quote marks around the worksheet path if a wildcard is used (e.g.,`*_worksheet.yaml`).
- Do not add mode, state, level, or service options; the configurator ignores them for worksheet import.
- The type of the config set must match the type of the worksheets being imported.

5. Proceed to *Validate the Config Set and Run Ansible Plays* on page 43.

4.3 Reconfigure DVS Interactively

Prerequisites

This procedure assumes that a config set has been created and DVS is already configured.

About this task

This procedure describes how to reconfigure DVS by updating the DVS service in config set *p0*. To use the commands in the examples, replace *p0* with the name of the config set being updated.

CAUTION: Configure DVS and all services **only** through the configurator or by placing/editing configuration files in the Simple Sync directory structure within the config set. Do not configure DVS by manually adding lines to `/etc/fstab` or other `/etc` files. In general, changes to those files are not persistent, and rebooting could result in loss of data.

IMPORTANT: When configurator guidance indicates a relationship or interaction between one or more settings, it is advisory only; the configurator does not automatically check to ensure compatibility among settings. However, the underlying implementation of DVS is unchanged, and it **does** automatically set related mount options when certain mount options are specified. To prevent mount failure, enter setting values that are compatible, in accordance with the instructions in this publication.

Procedure

1. (Optional) Use the `cfgset` command to view current DVS settings.

   ```
   smw# cfgset search --service cray_dvs --level advanced p0
   ```

2. Use `cfgset` to update DVS.

   To reconfigure DVS interactively:

   ```
   smw# cfgset update -s cray_dvs -l advanced --mode interactive p0
   ```

   The `cfgset` command invokes the configurator in interactive mode, and the configurator presents the Cray DVS Service Menu, which displays all DVS settings and their values.

   ```
   Service Configuration Menu (Config Set : p0, type: cle)

   cray_dvs          { status: enabled }  { validation: valid }
   ------------------------------------------------------------
    Selected   #    Settings           Value/Status (level=advanced)
   ------------------------------------------------------------
              1)   client_mount
                      reference: CSS            [OK]
                      reference: ComputeHome    [OK]

                   kernel_param
              2)      dvsipc_heartbeat_timeout  60
              3)      dvs_debug_mask            0
   ```

```
...
Cray DVS Service Menu [default: save & exit - Q] $
```

3. Make changes to an existing client mount (`client_mount`) or kernel module parameter (`kernel_param`), or add a new client mount.

   - To make changes to an existing client mount, enter **1** at the prompt and press `<cr>` (**Enter**), then enter **C** and press `<cr>` to display the configuration setting screen.

     ```
     Service Configuration Menu (Config Set : p0, type: cle)
     ...
     Cray DVS Service Menu [default: save & exit - Q] $ 1
     ...
     Cray DVS Service Menu [default: configure - C] $ C
     *********** cray_dvs.settings.client_mount **************

          client_mount
            <guidance>

          Configured Values:
              1) 'CSS'
                  a) mount_point: /cray
                  b) spath: /cray
                  c) server_groups:
                         dvs_nodes
                  d) client_groups: (none)
                  e) loadbalance: False
                  f) attrcache_timeout: 3
                  g) readonly: False
                  h) options: maxnodes=1

              2) 'ComputeHome'
                  a) mount_point: /home
                  b) spath: /home
                  c) server_groups:
                         dvs_nodes
                  d) client_groups: (none)
                  e) loadbalance: False
                  f) attrcache_timeout: 3
                  g) readonly: False
                  h) options: maxnodes=1

     cray_dvs.settings.client_mount
     [<cr>=set 2 entries, +=add an entry, ?=help, @=less] $
     ```

     In the configuration setting screen:

     - To modify the setting values of an existing client mount, enter the number of the client mount to be modified and the letter of the setting to be modified, followed by an asterisk. In this example, the second client mount is selected and the setting corresponding to list item 'b' is selected (the `spath` setting).

       ```
       cray_dvs.settings.client_mount
       [<cr>=set 2 entries, +=add an entry, ?=help, @=less] $ 2b*
       ```

     - To delete a client mount, enter the number of the client mount to be deleted, followed by a minus sign. In this example, the second client mount is deleted.

       ```
       cray_dvs.settings.client_mount
       [<cr>=set 2 entries, +=add an entry, ?=help, @=less] $ 2-
       ```

- To set or change one of the kernel module parameters listed in the Cray DVS Service Menu, enter the number of the parameter to be set/changed at the prompt and press <cr>, then enter **C** and press <cr> to display the configuration setting screen. In this example, setting 2 is selected, which is the first kernel parameter. To change kernel module parameters that do not appear in the Cray DVS Service Menu, see *Configure DVS using Modprobe or Proc Files* on page 30.

```
Cray DVS Service Menu [default: save & exit - Q] $ 2
...
Cray DVS Service Menu [default: configure - C] $ C
...
cray_dvs.settings.kernel_param.data.dvsipc_heartbeat_timeout
[<cr>=keep'60', <new value>, ?=help, @=less] $
```

  In the configuration setting screen, accept the current value (**<cr>**), revert to the default value (**#**), or enter a new value. Enter **?** to see a list of possible commands.

- To add a new client mount, follow these instructions:

a. Select the client mount setting.

```
Cray DVS Service Menu [default: save & exit - Q] $ 1
...
Cray DVS Service Menu [default: configure - C] $ C
```

b. Add a new client mount entry.

```
cray_dvs.settings.client_mount
[<cr>=set 2 entries, +=add an entry, ?=help, @=less] $ +
```

c. Set the client mount reference.

reference	A human-readable string—a name—that is used to uniquely identify a client mount. `reference` cannot be set by accepting the default: a non-empty string is required. Related settings/options: Because this is the key field of a client mount setting entry, each setting within the client mount setting includes this string in its full setting name.

  In this example, the string *tmp_failover* is used as the client mount reference. Whatever is chosen as the reference is used in the full setting names of all of the rest of that mount's settings.

```
cray_dvs.settings.client_mount.data.reference
[<cr>=set '', <new value>, ?=help, @=less] $ tmp_failover
```

d. Set the pathname of the mount point on the client.

mount_point	A string that specifies the full pathname on the client of the projected file system. `mount_point` cannot be set by accepting the default: a non-empty string is required. Related settings/options: none

```
cray_dvs.settings.client_mount.data.tmp_failover.mount_point
[<cr>=set '', <new value>, ?=help, @=less] $ /here
```

e. Set the pathname of the mount point on the DVS server node.

spath A string that specifies the full pathname on the DVS server of the file system that is to be projected for a client mount. It must be an absolute path and it must exist on the DVS server. `spath` cannot be set by accepting the default: a non-empty string is required.

Related settings/options: none

```
cray_dvs.settings.client_mount.data.tmp_failover.spath
[<cr>=set '', <new value>, ?=help, @=less] $ /tmp
```

f. Set the list of DVS server nodes.

server_groups A list of node groups that will function as DVS servers for a client mount. Enter one node group per line (see *About Node Groups* on page 105). `server_groups` cannot be set by accepting the default: a non-empty list is required.

> **IMPORTANT:** DVS servers should be dedicated because they use unlimited amounts of CPU and memory resources based directly on the I/O requests sent from DVS clients. Avoid using nodes that have other services (Lustre nodes, login nodes, etc.) or are tier2 nodes.

Related settings/options: Functionally equivalent to the `nodename` or `nodefile` "additional" option in the `options` setting of the client mount setting. The use of those two additional options is deprecated.

```
cray_dvs.settings.client_mount.data.tmp_failover.servers
[<cr>=set 0 entries, +=add an entry, ?=help, @=less] $ +
Add servers (Ctrl-d to exit) $ c0-0c2s1n1
Add servers (Ctrl-d to exit) $ c0-0c1s2n1
Add servers (Ctrl-d to exit) $ <Ctrl-d>
     2 entries added. Press <cr> to set.
cray_dvs.settings.client_mount.data.tmp_failover.servers
[<cr>=set 2 entries, +=add an entry, ?=help, @=less] $ <cr>
```

g. Set the list of DVS clients.

client_groups A list of node groups that will function as DVS clients for a client mount. Enter node groups one per line. Unlike `server_groups`, `client_groups` can be set to an empty list. If no node groups are specified, the mount will be performed on all suitable compute nodes (a compute node functioning as a DVS server is an example of an unsuitable node). This is common.

Related settings/options: none

```
cray_dvs.settings.client_mount.data.tmp_failover.clients
[<cr>=set 0 entries, +=add an entry, ?=help, @=less] $ <cr>
```

This example sets `clients` to an empty list.

h. Set the loadbalance option.

loadbalance Used to specify loadbalance mode, which more evenly distributes loads across DVS servers. Loadbalance mode is valid only for read-only mounts. For more information, see *DVS Loadbalance Mode*.

Related settings/options: When `loadbalance` is enabled, the underlying DVS implementation automatically sets the `readonly` setting to `true` and sets these additional options: `cache=1`, `failover=1`, `maxnodes=1`, and `hash_on_nid=0`. Cray recommends setting the `attrcache_timeout` setting as well to take advantage of the mount being read-only. If `loadbalance` is enabled, leave the `readonly` setting unconfigured or set it to `true` to maintain consistency with the way DVS implements `loadbalance`.

```
cray_dvs.settings.client_mount.data.tmp_failover.loadbalance
[<cr>=set 'false', <new value>, ?=help, @=less] $ <cr>
```

Pressing **<cr>** accepts the configurator default, which is to not enable loadbalance.

i. Set the attribute cache timeout option.

attrcache_timeout Enables client-side attribute caching, which can significantly increase performance, most notably in pathname lookup situations. When attribute caching is disabled, DVS clients must send a lookup request to a DVS server for every level of a pathname, and repeat this for every pathname operation. When it is enabled, it sends a lookup request to a DVS server for every level of a pathname once per *n* seconds.

The configurator default is `14400` seconds. The underlying DVS implementation default is `3` seconds, which is safer for read-write mounts. This means that to enhance system performance for read-only mounts, configure this setting by accepting the configurator default (or entering some other value). Leaving this setting unconfigured will result in the underlying default being used.

Related settings/options: The Ansible play that consumes DVS configuration data prevents use of this mount option for read-write file systems due to the risk of file system corruption. Run-time mounts not accompanied by that Ansible play do not have that safeguard. In such cases, if a read-write mount is created, it is safe to leave `attrcache_timeout` unconfigured so that the underlying default is used.

```
cray_dvs.settings.client_mount.data.tmp_failover.attrcache_timeout
[<cr>=set '14400', <new value>, ?=help, @=less] $ <cr>
```

Pressing **<cr>** accepts the configurator default of 14400, which will be fine because the next setting will make this a read-only mount.

j. Set the read-only option.

readonly Determines whether the client mount is read-only or read-write. If intending to enable client-side caching of read data on a non-writable file system, use this `readonly` setting to force the DVS mount to be read-only. This will disable write caching.

The configurator default is `true` or `1`. The underlying DVS implementation default is `false`. Leaving this setting unconfigured will result in the underlying DVS default being used.

Related settings/options: When `loadbalance` is enabled, DVS automatically enables `readonly` but the configurator does not, so either leave this setting

unconfigured or accept the configurator default. If the `attrcache_timeout` setting is set for this client mount, `readonly` should be enabled (set to true) in the configurator/worksheet. If the `cache` option is specified in the `options` setting for this client mount, enabling `readonly` is the only way to enable read caching without enabling write caching as well).

```
cray_dvs.settings.client_mount.data.tmp_failover.readonly
[<cr>=set 'true', <new value>, ?=help, @=less] $ <cr>
```

k. Set other client mount options.

`options`

Provides the only way to specify mount options in addition to the ones already specified in the other mount point settings. Enter a string with mount options separated by comma and no spaces. For information about available options and their implications, see *Additional Options for Use in the Options Setting of a Client Mount*. Note that it is necessary to specify `maxnodes=1` here for a read-write client mount of an NFS or other non-cluster, non-coherent file system.

Related settings/options: Options contained in this setting will be appended to the mount options specified in other settings. Any that are functionally redundant with settings already configured (such as *nodename*/*nodefile*, which are redundant with the `server_groups` setting) will override those settings.

```
cray_dvs.settings.client_mount.data.tmp_failover.options
[<cr>=set '', <new value>, ?=help, @=less] $ failover,cache
...
cray_dvs.settings.client_mount
[<cr>=set 3 entries, +=add an entry, ?=help, @=less] $ <cr>
```

Suppose a desired option was omitted. Because the `options` setting is a string rather than a list, another option can be added only by entering a new string with all desired options, which replaces the old string. Revisit that setting (enter **`1`** at the service menu prompt and press `<cr>` twice, then enter **`3h*`**, where 3 is the number of the mount entry, at the configuration setting screen prompt) and enter the new string of options.

```
cray_dvs.settings.client_mount.data.tmp_failover.options
[<cr>=keep 'failover,cache', <new value>, ?=help, @=less] $
failover,cache,maxnodes=3
```

4. Save the changes and exit the configurator session when done making changes.

```
Cray DVS Service Menu [default: save & exit - Q] $ Q
```

`cfgset` will save all of the configuration settings in the config set (in configuration worksheets and templates), run post-configuration scripts, create a time-stamped clone of the config set, and exit automatically.

5. Proceed to *Validate the Config Set and Run Ansible Plays* on page 43.

4.4 Configure DVS using Modprobe or Proc Files

Most DVS kernel module parameters cannot be changed using the configurator. These parameters are typically changed by adding lines to a `modprobe.d` configuration file or echoing values to a `/proc` file. Changes to a `modprobe.d` file are made prior to booting the affected nodes, and the changes take effect at boot. To ensure that these changes persist across boots, the `modprobe.d` files can be placed in the Simple Sync directory of the working CLE config set. Changes made to those files will then be propagated to all target nodes using the Simple Sync mechanism.

The `dvs.conf` file is one of many files that are generated automatically and controlled by the Cray Configuration Management Framework (CMF). Such files can be identified by the warning statement in the file header, which includes a statement that the file should not be modified directly or updated using Simple Sync. For this reason, Cray recommends that sites create a local DVS configuration file (*`dvs-local`*`.conf`) for configuring the kernel module parameters listed in this section. Then *`dvs-local`*`.conf` can be placed in the appropriate Simple Sync directory to augment the configuration specified in `dvs.conf`.

The Simple Sync Mechanism and File Structure

Cray Simple Sync provides a generic mechanism to automatically distribute files to targeted locations on a Cray XC system. This mechanism can be used to override or change default system behavior through the contents of the distributed files. When enabled, the Simple Sync service is executed on all internal CLE nodes and eLogin nodes at boot time and whenever the administrator executes `/etc/init.d/cray-ansible start` on a CLE node or eLogin node. When Simple Sync is executed, files placed in the following directory structure are copied to the root file system (`/`) on the target nodes.

The Simple Sync directory structure has this root:
`smw:/var/opt/cray/imps/config/sets/<config_set>/files/simple_sync/`

Below that root are the directories listed on the left. Files placed in those directories are copied to their associated target nodes.

Files placed here	are copied to
`./common/files/`	all internal and eLogin nodes
`./platform/[compute|service]/files/`	all CLE compute nodes or all service nodes (not applicable to eLogin nodes)
`./hardwareid/<hardwareid>/files/`	nodes with matching hardware ID, which is the cname of a CLE node or the output of the `hostid` command (e.g., 1eac0b0c) on other nodes (not applicable to eLogin nodes)
`./hostname/<hostname>/files/`	nodes with matching host name (use this for eLogin nodes ONLY)
`./nodegroups/<node_group_name>/files/`	nodes in the matching node group

> **NOTE:** The directory structure for a particular hardware ID or host name (everything below `./hardwareid/` and `./hostname/`) must be created manually as needed. This is unnecessary for node groups because their associated directories are created automatically by post-configuration callback scripts when the config set is created or updated using `cfgset`.

Anything (directory structure and files) placed below `./files/` in the Simple Sync directory structure on the SMW is replicated on the target node starting at root (`/`). For example, if the `myapplication.conf` file is placed in this path on the SMW

```
/var/opt/cray/imps/config/sets/p0/files/simple_sync/common/files/etc/myapplication.conf
```

then Simple Sync will place `myapplication.conf` here on all nodes:

```
/etc/myapplication.conf
```

Note that the ownership and permissions of files in the config set are preserved in the copies made to nodes.

For more information and use cases about Simple Sync, see *Supplemental Information* on page 93.

Change Kernel Module Parameters Prior to Boot using Modprobe.d Files and Simple Sync

To make a change to DVS kernel module parameters that will persist across boots, edit the local copy of the DVS configuration file (`modprobe.d/dvs-local.conf`), and then place it under the Simple Sync directory structure that targets the appropriate nodes in the system. The lines to add to `modprobe.d/dvs-local.conf` are provided for each kernel module parameter in the list that follows. Substitute the appropriate Simple Sync directory structure (that is, the one that targets the appropriate nodes) for *<simple sync path>* shown in the instructions for each parameter.

Procedure for Changing Kernel Module Parameters Prior to Boot using Modprobe.d Files and Simple Sync

In this example, DVS request logging is enabled on a node with hardware ID c0-0c0s3n2 in config set p0.

1. Change directory to the Simple Sync root directory.

   ```
   smw# cd /var/opt/cray/imps/config/sets/p0/files/simple_sync
   ```

2. Create and change to the directory under the Simple Sync root that targets the appropriate nodes. Use the above table to determine what directory that should be. This example targets a particular hardware ID, so the directory is `hardwareid/c0-0c0s3n2/files/`.

   ```
   smw# mkdir -p hardwareid/c0-0c0s3n2/files
   smw# cd hardwareid/c0-0c0s3n2/files
   ```

   The file path that results is referred to as *<simple sync path>* in the "To change prior to boot" instructions for each kernel module parameter listed below.

3. Create and change to the directory structure that is to be replicated on the target node. For a modprobe.d file, that is `/etc/modprobe.d`.

   ```
   smw# mkdir -p etc/modprobe.d
   smw# cd etc/modprobe.d
   ```

4. Create or edit the DVS configuration file and add the options line(s) for the parameter to be set/changed. For this example, the lines are from the `dvs_request_log_enabled` entry of the DVS kernel module parameters list below. Then comment/uncomment the appropriate lines, depending on which action is to be taken.

   ```
   # Disable DVS request log
   options dvsproc dvs_request_log_enabled=0
   ```

   ```
   # Enable DVS request log
   options dvsproc dvs_request_log_enabled=1
   ```

5. When done with all changes to kernel module parameters, proceed to *Validate the Config Set and Run Ansible Plays* on page 43.

Change Kernel Module Parameters Dynamically using Proc Files

Some of the kernel module parameters in the following list can be changed dynamically by echoing values to `/proc` files on the appropriate nodes. Those that can be changed using that method are indicated in the list, including the name of the `/proc` file and the values to use. Note that such changes do not persist.

List of DVS Kernel Module Parameters not Accessible through the Configurator

dvs_request_log_enabled

Logs each DVS request sent to servers.

- Default value: 1 (enabled)
- To view read-only:
  cat `/sys/module/dvsproc/parameters/dvs_request_log_enabled`
- To change prior to boot, add these lines to
  <simple sync path>`/etc/modprobe.d/dvs-local.conf`:

```
# Disable DVS request log
options dvsproc dvs_request_log_enabled=0

# Enable DVS request log
options dvsproc dvs_request_log_enabled=1
```

- To change dynamically:

```
hostname# echo 0 > /proc/fs/dvs/request_log
hostname# echo 1 > /proc/fs/dvs/request_log
hostname# echo 2 > /proc/fs/dvs/request_log
```

  The value `2` resets the log.

dvs_request_log_size_kb

Size (KB) of the request log buffer.

- Default value: 16384 KB (16384 * 1024 bytes)
- To view read-only:
  cat `/sys/module/dvsproc/parameters/dvs_request_log_size_kb`
- To change prior to boot, add these lines to
  <simple sync path>`/etc/modprobe.d/dvs-local.conf`:

```
# Set size (in kb) of the request log buffer
options dvsproc dvs_request_log_size_kb=17000
```

- To change dynamically:

```
hostname# echo 17000 > /proc/fs/dvs/request_log_size_kb
```

  To determine the current buffer size, cat the file. For example:

```
hostname# cat /proc/fs/dvs/request_log_size_kb
16384
```

dvs_request_log_min_time_secs

Defines a threshold of time for data to be logged to the `request_log`.

- Default value: 15 seconds
- To view read-only:
  cat `/sys/module/dvsproc/parameters/dvs_request_log_time_min_secs`
- To change prior to boot, add these lines to
  `<simple sync path>/etc/modprobe.d/dvs-local.conf`:

```
# Set threshold (in seconds) for time a DVS request
# takes before logged. Requests taking fewer seconds
# will not be logged.
options dvsproc dvs_request_log_time_min_secs
```

- To change dynamically:

```
hostname# echo 15 > /proc/fs/dvs/request_log_time_min_secs
```

dvs_fs_log_enabled

Logs information on I/O operations made from DVS to the underlying file system on DVS server nodes.

- Default value: 1 (enabled)
- To view read-only:
  cat `/sys/module/dvsproc/parameters/dvs_fs_log_enabled`
- To change prior to boot, add these lines to
  `<simple sync path>/etc/modprobe.d/dvs-local.conf`:

```
# Disable DVS fs log
options dvsproc dvs_fs_log_enabled=0

# Enable DVS fs log
options dvsproc dvs_fs_log_enabled=1
```

- To change dynamically:

```
hostname# echo 0 > /proc/fs/dvs/fs_log
hostname# echo 1 > /proc/fs/dvs/fs_log
hostname# echo 2 > /proc/fs/dvs/fs_log
```

dvs_fs_log_size_kb

Size (KB) of the log buffer.

- Default value: 32768 KB (32768 * 1024 bytes)
- To view read-only:
  cat `/sys/module/dvsproc/parameters/dvs_fs_log_size_kb`
- To change prior to boot, add these lines to
  `<simple sync path>/etc/modprobe.d/dvs-local.conf`:

```
# Set size (in kb) of the fs log buffer
options dvsproc dvs_fs_log_size_kb=17000
```

- To change dynamically:

```
hostname# echo 17000 > /proc/fs/dvs/fs_log_size_kb
```

To determine the current buffer size, cat the file. For example:

```
hostname# cat /proc/fs/dvs/fs_log_size_kb
32768
```

dvs_fs_log_min_time_secs

Defines a threshold of time for data to be logged to the `fs_log`.

- Default value: 15 seconds
- To view read-only:
  cat `/sys/module/dvsproc/parameters/dvs_fs_log_time_min_secs`
- To change prior to boot, add these lines to
  <simple sync path>`/etc/modprobe.d/dvs-local.conf`:

```
# Set threshold (in seconds) for time a DVS requests
# takes before logged. Requests taking fewer seconds
# will not be logged.
options dvsproc dvs_fs_log_time_min_secs
```

- To change dynamically:

```
hostname# echo 15 > /proc/fs/dvs/fs_log_time_min_secs
```

dvs_instance_info

Contains the following fields, which are parameters for the DVS thread pool in the DVS IPC layer. Most of these fields can be changed through other module parameters (e.g., `dvsipc_msg_thread_limit` and `dvsipc_single_msg_queue`); however, this module parameter has priority over the individual ones and if set, will override them.

Field	Definition
thread_min	Number of threads created at startup.
thread_max	Maximum number of persistent threads.
thread_limit	Maximum number of valid threads that DVS IPC allows to exist at one time. DVS IPC will dynamically scale up the number of threads to this number as the load increases.
thread_concurrent_creates	Maximum number of IPC threads that can be in the process of forking a new thread.
thread_nice	Nice value for the threads.
single_msg_queue	Disables/enables fairness of service. Setting to 1 disables fairness of service by processing incoming requests with a FIFO method. Setting to 0 (default) groups requests into different queues (qhdrs) based on apid, and round-robbins the queues to maintain quality of service (QOS) among jobs.

Field	Definition
init_free_qhdrs	Low water mark for the pool of unused qhdrs. When the pool falls below this number, more are allocated. Used only if single_msg_queue = 0.
max_free_qhdrs	Maximum number of unused qhdrs that can exist before the system starts freeing them. Used only if single_msg_queue = 0.
Interactions among fields: thread_min <= thread_max <= thread_limit thread_concurrent_creates <= thread_limit init_free_qhdrs and max_free_qhdrs used only when single_msg_queue == 0	

IMPORTANT: For this parameter to be valid, values must be specified for all of the fields. To avoid unintentional changes, be careful when changing any field values.

- Default value: see modprobe.d examples
- To view read-only: cat `/sys/module/dvsipc/parameters/dvs_instance_info`
- To change prior to boot, add these lines to `<simple sync path>/etc/modprobe.d/dvs-local.conf` (comment out either the compute or service line, depending on the type of node(s) being configured):

```
# Set parameters for DVS thread pool in DVS IPC layer
# Defaults for compute nodes
options dvsipc dvs_instance_info=4,16,1000,0,0,0,1,1

# Defaults for service nodes
options dvsipc dvs_instance_info=16,64,1000,0,0,0,64,2048
```

  For compute nodes, this translates to dvs_instance_info = thread_min = 4, thread_max = 16, thread_limit = 1000, thread_concurrent_creates = 0, thread_nice = 0, single_msg_queue = 0, init_free_qhdrs = 1, max_free_qhdrs = 1.

  For service nodes, this translates to dvs_instance_info=thread_min = 16, thread_max = 64, thread_limit = 1000, thread_concurrent_creates = 0, nice = 0, single_msg_queue = 0, init_free_qhdrs = 64, max_free_qhdrs = 2048.

  Note that if using the defaults for a compute node, ensure that dvsipc_config_type=0 is also set, and likewise, for a service node, ensure that dvsipc_config_type=1 for consistency.

- To change dynamically: N/A

dvsipc_config_type

Forces DVS to load in a mode optimized for DVS clients (0) or servers (1). This parameter can be used to make DVS load in a non-default manner. Frequently used for repurposed compute nodes.

- Default value: 0 for compute nodes, 1 for service nodes

- To view read-only: cat `/sys/module/dvsipc/parameters/dvsipc_config_type`
- To change prior to boot, add these lines to `<simple sync path>/etc/modprobe.d/dvs-local.conf` (comment out either the client or server line, depending on the type of node(s) being configured):

```
# Load DVS as a client
options dvsipc dvsipc_config_type=0

# Load DVS as a server
options dvsipc dvsipc_config_type=1
```

- To change dynamically:

```
hostname# echo 0 > /proc/fs/dvs/ipc/config-type
hostname# echo 1 > /proc/fs/dvs/ipc/config-type
```

dvsipc_single_msg_queue

Disables fairness of service, which is enabled by default. Setting this parameter to 1 disables fairness of service by forcing DVS to use a single message queue instead of a list of queues.

- Default value: 0 (fairness of service enabled)
- To view read-only:
  cat `/sys/module/dvsipc/parameters/dvsipc_single_msg_queue`
- To change prior to boot, use the `single_msg_queue` field of the `dvs_instance_info` parameter. If no other fields in `dvs_instance_info` need to be changed, it may be easier to change the `dvsipc_single_msg_queue` parameter directly by adding these lines to `<simple sync path>/etc/modprobe.d/dvs-local.conf`:

```
# Disable fairness of service
options dvsipc dvsipc_single_msg_queue=1

# Enable fairness of service
options dvsipc dvsipc_single_msg_queue=0
```

- To change dynamically: N/A

dvsof_concurrent_reads

Controls how many threads are allowed into the read path on the server. A value of -1 disables, 0 uses the number of cores on the CPU, and any other positive number sets the number of threads. Set to 0 for best DataWarp performance.

- Default value: -1
- To view read-only:
  cat `/sys/module/dvs/parameters/dvsof_concurrent_reads`
- To change prior to boot, add these lines to `<simple sync path>/etc/modprobe.d/dvs-local.conf`:

```
# Disable concurrent reads
options dvs dvsof_concurrent_reads=-1
```

```
# Set number of threads able to do concurrent
# reads = number of cores on CPU
options dvs dvsof_concurrent_reads=0

# Set number of threads able to do concurrent
# reads = a positive number (e.g., 3)
options dvs dvsof_concurrent_reads=3
```

- To change dynamically: N/A

dvsof_concurrent_writes

Controls how many threads are allowed into the write path on the server. A value of -1 disables, 0 uses the number of cores on the CPU, and any other positive number sets the number of threads. Set to 0 for best DataWarp performance.

- Default value: -1
- To view read-only:
  cat `/sys/module/dvs/parameters/dvsof_concurrent_writes`
- To change prior to boot, add these lines to
  <simple sync path>`/etc/modprobe.d/dvs-local.conf`:

```
# Disable concurrent writes
options dvs dvsof_concurrent_writes=-1

# Set number of threads able to do concurrent
# writes = number of cores on CPU
options dvs dvsof_concurrent_writes=0

# Set number of threads able to do concurrent
# writes = a positive number (e.g., 3)
options dvs dvsof_concurrent_writes=3
```

- To change dynamically: N/A

dvsproc_stat_control

(deprecated) Controls DVS statistics. This legacy parameter has been maintained for backward compatibility, but values are overridden by `dvsproc_stat_defaults`, if specified.

- Default value: 1 (enabled)
- To view: cat `/sys/module/dvsproc/parameters/dvsproc_stat_control`
- To change prior to boot, add these lines to
  <simple sync path>`/etc/modprobe.d/dvs-local.conf`:

```
# Disable DVS statistics
options dvsproc dvsproc_stat_control=0

# Enable DVS statistics
options dvsproc dvsproc_stat_control=1
```

- To change dynamically:

  This is root writable
  at `/sys/module/dvsproc/parameters/dvsproc_stat_control`, but changes

should be made only through the `/proc/fs/dvs/stats` interface, as shown in this example.

```
hostname# echo 0 > /proc/fs/dvs/stats
hostname# echo 1 > /proc/fs/dvs/stats
hostname# echo 2 > /proc/fs/dvs/stats
```

dvsproc_stat_defaults

Controls DVS statistics. Use this parameter to disable/enable and format DVS statistics. The options that can be specified are listed in *Option Values for Controlling DVS Statistics* on page 74.

- Default values: enable, legacy, brief, plain, notest
- To view: cat `/sys/module/dvsproc/parameters/dvsproc_stat_defaults`
- To change prior to boot, add these lines to *<simple sync path>*`/etc/modprobe.d/dvs-local.conf`:

```
# Disable/enable and format DVS statistics
options dvsproc
dvsproc_stat_defaults="enable,legacy,brief,plain,notest"
```

- To change dynamically:

  This is root writable at `/sys/module/dvsproc/parameters/dvsproc_stat_defaults`, but changes should be made only through the `/proc/fs/dvs/stats` interface, as shown in this example.

```
hostname# echo disable > /proc/fs/dvs/stats
hostname# echo enable > /proc/fs/dvs/stats
hostname# echo reset > /proc/fs/dvs/stats
hostname# echo json,pretty > /proc/fs/dvs/stats
```

estale_max_retry

Controls the number of times to retry an operation on the original server after it returns ESTALE.

- Default value: 36 iterations at a fixed 5 seconds per iteration (3 minutes)
- To view read-only: cat `/sys/module/dvsproc/parameters/estale_max_retry`
- To change prior to boot, add these lines to *<simple sync path>*`/etc/modprobe.d/dvs-local.conf`:

```
# Set the number of times to retry an operation
# on the original server after it returns ESTALE
options dvsproc estale_max_retry=
```

- To change dynamically (example changes `estale_max_retry` to `40` for illustration only):

```
hostname# echo 40 > /proc/fs/dvs/estale_timeout_secs
```

estale_timeout_secs

Controls the time to wait between retries of an operation after it returns ESTALE.

- Default value: 300 seconds
- To view read-only:
  cat `/sys/module/dvsproc/parameters/estale_timeout_secs`
- To change prior to boot, add these lines to
  `<simple sync path>/etc/modprobe.d/dvs-local.conf`:

```
# Set the time to wait between retries of an
# operation that returns ESTALE
options dvsproc estale_timeout_secs=
```

- To change dynamically (example changes `estale_timeout_secs` to `400` for illustration only):

```
hostname# echo 400 > /proc/fs/dvs/estale_timeout_secs
```

kdwfs_instance_info

Contains the following fields, which are parameters for the DataWarp thread pool in the DVS IPC layer.

Field	Definition
thread_min	Number of threads created at startup.
thread_max	Maximum number of persistent threads.
thread_limit	Maximum number of valid threads that DVS IPC allows to exist at one time. DVS IPC will dynamically scale up the number of threads to this number as the load increases.
thread_concurrent_creates	Maximum number of IPC threads that can be in the process of forking a new thread.
thread_nice	Nice value for the threads.
single_msg_queue	Disables/enables fairness of service. Setting to 1 disables fairness of service by processing incoming requests with a FIFO method. Setting to 0 (default) groups requests into different queues (qhdrs) based on apid, and round-robbins the queues to maintain quality of service (QOS) among jobs.
init_free_qhdrs	Low water mark for the pool of unused qhdrs. When the pool falls below this number, more are allocated. Used only if single_msg_queue = 0 .
max_free_qhdrs	Maximum number of unused qhdrs that can exist before the system starts freeing them. Used only if single_msg_queue = 0.
Interactions among fields: thread_min <= thread_max <= thread_limit (set all three equal for best DataWarp performance) thread_concurrent_creates <= thread_limit	

Field	Definition
init_free_qhdrs and max_free_qhdrs used only when single_msg_queue == 0	

IMPORTANT: For this parameter to be valid, values must be specified for all of the fields. To avoid unintentional changes, be careful when changing any field values.

- Default value: 1,1,1024,4,-10,1,1,1
- To view read-only:
  cat `/sys/module/dvsipc/parameters/kdwfs_instance_info`
- To change prior to boot, add these lines to
  `<simple sync path>/etc/modprobe.d/dvs-dws.conf` (values shown are defaults):

```
# Set parameters for DataWarp thread pool in DVS IPC layer
options dvsipc kdwfs_instance_info=256,256,1024,4,-10,1,1,1
```

  This translates to kdwfs_instance_info thread_min = 256, thread_max = 256, thread_limit = 1024, thread_concurrent_creates = 4, nice = -10, single_msg_queue = 1, init_free_qhdrs = 1, max_free_qhdrs = 1.
- To change dynamically: N/A

lnd_name

`lnd_name` uniquely identifies the LNet network that DVS will use. DVS communicates it to the LNet service when DVS is being initialized. It must match the `cray_lnet.settings.local_lnet.data.lnet_name` value set in the `cray_lnet` service for DVS to boot properly. To find that value, search the CLE config set (this example searches in config set `p0` and finds lnet_name = gni4):

```
smw# cfgset search --term lnet_name \
--state all --service cray_lnet p0
# 1 match for 'lnet_name' from cray_lnet_config.yaml
#-------------------------------------------------
cray_lnet.settings.local_lnet.data.lnet_name: gni4
```

- Default value: gni99
- To view read-only: not visible in `/sys/module`
- To change prior to boot, add these lines to
  `<simple sync path>/etc/modprobe.d/dvs-local.conf`, substituting for *gnix* the value found from the config set search:

```
# Set identifier of LNet network DVS will use
options dvsipc_lnet lnd_name=gnix
```

- To change dynamically: N/A

sync_dirty_timeout_secs

On DVS *servers*, specifies the number of seconds that must have passed since the file was written before DVS syncs it. The objective is to reduce unnecessary sync operations for files actively being updated. Decreasing this number increases the likelihood that the file is in use when it is synced. Increasing this number increases the likelihood that processes are killed during a server failure.

On DVS *clients*, specifies the number of seconds that must have passed since the file was written before DVS asks the server for an updated sync time. Decreasing this number increases the number of DVS requests being sent. Increasing this number increases the likelihood that processes are killed during a server failure.

- Default value: 300 (servers and clients)
- To view read-only:
  cat `/sys/module/dvs/parameters/sync_dirty_timeout_secs`
- To change prior to boot, add these lines to
  `<simple sync path>`/etc/modprobe.d/dvs-local.conf:

```
# Set the timeout (seconds) for syncing
# dirty files on a DVS server or client
options dvs sync_dirty_timeout_secs=300
```

- To change dynamically:

```
hostname# echo 300 > /proc/fs/dvs/sync_dirty_timeout_secs
```

sync_num_threads

Specifies the number of threads on the DVS server that perform sync operations. The number of threads must be a minimum of 1 thread and a maximum of 32. Note that it can take up to `sync_period_secs` for changes to this value to take effect.

- Default value: 8
- To view read-only: cat `/sys/module/dvs/parameters/sync_num_threads`
- To change prior to boot, add these lines to
  `<simple sync path>`/etc/modprobe.d/dvs-local.conf:

```
# Set the number of threads that perform
# sync operations on a DVS server
options dvs sync_num_threads=8
```

- To change dynamically:

```
hostname# echo 8 > /proc/fs/dvs/sync_num_threads
```

sync_period_secs

On DVS *servers*, specifies the number of seconds before the `sync_num_threads` syncs files on the DVS server (if necessary).

On DVS *clients*, specifies the number of seconds between checks for dirty files that need to request the last sync time from the server.

- Default value: 300 (server), 600 (client)
- To view read-only: cat `/sys/module/dvs/parameters/sync_period_secs`

- To change prior to boot, add these lines to *<simple sync path>*/etc/modprobe.d/dvs-local.conf:

```
# Set the sync period (seconds) on DVS server/client
options dvs sync_period_secs=300
```

- To change dynamically:

```
hostname# echo 300 > /proc/fs/dvs/sync_period_secs
```

4.5 Validate the Config Set and Run Ansible Plays

Prerequisites

This procedure assumes that configuration data has been changed.

About this task

Whenever data in a global or CLE config set has been changed, it is necessary to update and validate that config set and run `cray-ansible` on the SMW (for global config set changes) and any affected CLE nodes (for CLE config set changes) in order to apply the configuration changes. If the system will be rebooted, the steps to run `cray-ansible` are not needed because `cray-ansible` is run automatically when the system boots.

Using `cfgset update` ensures that all pre- and post-configuration scripts get run. Running `cray-ansible` on the SMW applies global configuration changes there. Running `cray-ansible` on a CLE node triggers a refresh of the CLE config set cache on that node and applies configuration changes on the node.

The example commands use a CLE config set named *p0*; substitute the correct CLE config set name for this system.

Procedure

1. (If `cfgset update` not already run) Update the config set.

   When the config set is updated using the configurator, all of the pre-and post-configuration scripts are run.

   Use one or both of these commands, depending on which config sets have been changed.

   ```
   smw# cfgset update p0
   ```

   ```
   smw# cfgset update global
   ```

2. Validate the config set.

   Use one or both of these commands, depending on which config sets have been updated.

   ```
   smw# cfgset validate p0
   ```

   ```
   smw# cfgset validate global
   ```

3. (If global config set updated and system will not be rebooted) Run Ansible plays on the SMW.

After the global config set has been updated, run any Ansible plays that consume global config set data to apply that data on the SMW (`cray-ansible` runs all of them).

```
smw# /etc/init.d/cray-ansible start
```

4. (If CLE config set updated and node will not be rebooted) Run Ansible plays on the CLE nodes.

   After the CLE config set has been updated, refresh the local config set cache to pull any config set changes to the node and run `cray-ansible` to apply them on the node.

   ```
   hostname# /opt/cray/imps-distribution/default/bin/refresh.py
   hostname# /etc/init.d/cray-ansible start
   ```

4.6 Quiesce a DVS-projected File System

Sites can use the DVS quiesce capability to temporarily suspend traffic to a DVS-projected file system on any number of DVS servers. When a directory is quiesced, the DVS server completes any outstanding requests but does not honor any new requests for that directory. Any outstanding requests for that directory are displayed in `/proc/fs/dvs/quiesce` file system interface. Administrators can read that proc file to know when it is safe to perform operations on the quiesced directory without any possibility of interference by a DVS client. DVS quiesce can be used when a file system needs to be repaired or to safely take a DVS server node out of service.

CAUTION: Because it may cause data corruption, do not use DVS quiesce to:

- Quiesce a directory that is being used by DataWarp
- Quiesce a directory on every DVS server

To use DVS quiesce, an administrator writes into and reads from `/proc/fs/dvs/quiesce`, as shown in the following use cases.

Use Case 1: Quiesce a single directory on a single DVS server

An admin wants to quiesce a directory `/gpfs/test/foo` on a DVS server. This is an unlikely use case but an illustrative example.

1. On the DVS server, quiesce the directory.

   ```
   dvs1# echo quiesce /gpfs/test/foo > /proc/fs/dvs/quiesce
   ```

2. Ensure that the directory was properly quiesced and see if there are any outstanding requests. Repeat this occasionally to know when all outstanding requests have been cleared.

   ```
   dvs1# cat /proc/fs/dvs/quiesce
   /gpfs/test/foo/:   Outstanding_Requests 3
   ```

3. When finished with the directory, unquiesce it.

   ```
   dvs1# echo unquiesce /gpfs/test/foo > /proc/fs/dvs/quiesce
   ```

4. Ensure that the directory is no longer on the quiesced list.

   ```
   dvs1# cat /proc/fs/dvs/quiesce
   ```

Use Case 2: Quiesce all directories on a DVS server

An admin wants to remove a DVS server from service but wants to let any outstanding request complete first.

1. On the DVS server, quiesce all directories on that server.

```
dvs1# echo quiesce / > /proc/fs/dvs/quiesce
```

2. Look for any outstanding requests. Repeat this occasionally to know when all outstanding requests have been cleared.

```
dvs1# cat /proc/fs/dvs/quiesce
/:           Outstanding_Requests 3
```

   When no outstanding requests remain, the server can be removed from service.

3. To allow traffic to this server to resume, unquiesce all of its projected directories.

```
dvs1# echo unquiesce / > /proc/fs/dvs/quiesce
```

How Quiesce Works: the Userspace Application View

Userspace applications have no visibility into any specific quiesce information. A quiesced directory will present in one of two ways:

- Entirely normally, if the directory is quiesced only on servers that the application is not using.
- Useable but with degraded performance, if the application finds that its main server is quiesced and must query other servers.

How Quiesce Works: the Server View

To provide the quiesce capability, DVS servers keep a list of quiesced directories and the current outstanding requests on each quiesced directory. When an admin requests that DVS quiesce a directory on a server, DVS does the following:

- Adds that directory to the server's list of quiesced directories
- Iterates over all open files, closing any file that resides in the quiesced directory and setting a flag to indicate that the file was closed due to the directory being quiesced

When a DVS server receives a request from a client, DVS checks the request path against the list of quiesced directories. The comparison between the path name in the request and the quiesced directory is a simple string compare to avoid any access of the underlying file system that has been quiesced. If DVS finds that the request is for a quiesced file system, it sends a reply indicating that the request could not be completed due to a quiesce and noting which directory is quiesced. If the client request is for a file that has been closed due to quiesce, the server returns a reply to the client indicating that the request could not be completed due to a quiesce.

When an admin unquiesces a directory on a DVS server, DVS simply removes that directory from the server's list of quiesced directories and clears all quiesce-related flags for that directory.

How Quiesce Works: the Client View

When making a request of a server, a client may get back reply indicating that the request was for a file in a quiesced directory. The client then retries the operation on the next server in its server list. If it makes the request of every server in its server list and gets the same reply from each of them, then one of two things happens, depending on the type of request:

path name request	If the request is a path name request (lookup, stat, file open, etc.), then DVS reattempts the operation on a different server in a round-robin fashion until it finds a server that allows the operation to complete successfully.
open file	If the request is for an open file (read, write, lseek, etc.), then DVS attempts the operation on a different server. If the file is not open on any other servers, DVS attempts to open on the file on a server in a round robin fashion until it gets a successful open. DVS will then attempt to perform the operation.

If a client receives a reply indicating a quiesced directory, the client adds that directory to a list of quiesced directories held on the DVS superblock. This is intended to reduce network traffic by avoiding requests that target quiesced directories. The client's list of quiesced directories expires about every 60 seconds, thereby allowing clients to try those directories again in case one or more have been unquiesced during that time. This mechanism enables DVS to strike a balance between the timely unquiescing of a file system and a large reduction in network traffic and requests coming into the server. It also has the effect of naturally staggering clients when they start to use a server.

4.7 DVS Client-side Write-back Caching can Yield Performance Gains

With the advent of DataWarp and faster backing storage, the overhead of network operations has become an increasingly large portion of overall file system operation latency. DVS provides the ability to cache both read and write data on a client node while preserving close-to-open coherency and without contributing to out-of-memory issues on compute nodes. Instead of using network communication for all read/write operations, DVS can aggregate those operations and reuse data already read by or written from a client. This can provide a substantial performance benefit for these I/O patterns, which typically bear the additional cost of network latency:

- Small reads and writes
- Reads following writes
- Multiple reads of the same data

Client-side Write-back Caching may not be Suitable for all Applications

⚠ **CAUTION:** Possible data corruption or performance penalty!

Using the page cache may not provide a benefit for all applications. Applications that require very large reads or writes may find that introducing the overhead of managing the page cache slows down I/O handling. Benefit can also depend on write access patterns: small, random writes may not perform as well as sequential writes. This is due to pages being aggregated for write-back. If random writes do not access sequential pages, then less-than-optimal-sized write-backs may have to be issued when a break in contiguous cache pages is encountered.

More important, successful use of write-back caching on client nodes requires a clear understanding and acceptance of the limitations of close-to-open coherency. It is important for site system administrators to ensure that users at their site understand how client-side write-back caching works before enabling it. Without that understanding, users could experience data corruption issues.

How to Use Client-side Write-back Caching

Client-side write-back caching is disabled by default. Both write caching and read-only caching must be explicitly enabled.

To enable both read and write file caching on a client mount, follow these steps. This example assumes that two client mounts have been previously configured in CLE config set p0. See one of the DVS configuration topics for information about how to configure a new client mount.

1. Invoke the configurator for only the `cray_dvs` service (example uses config set p0).

```
smw# cfgset update --service cray_dvs --level advanced \
--mode interactive p0
```

2. At the first configurator prompt, enter **1** to select the `client_mount` setting, and then enter **C** to configure it.

```
Cray DVS Service Menu [default: save & exit - Q] $ 1
...
Cray DVS Service Menu [default: configure - C] $ C
```

3. Ensure that `readonly` is set to false. Entering **2** selects the second client mount, **g** selects the `readonly` setting of that mount, and the asterisk (*****) indicates that setting is to be edited. For a more complete view of the configuration setting screen, see *Reconfigure DVS Interactively* on page 25.

```
cray_dvs.settings.client_mount
[<cr>=set 2 entries, +=add an entry, ?=help, @=less] $ 2g*
...
cray_dvs.settings.client_mount.data.my_mount.readonly
[<cr>=set 'true', <new value>, ?=help, @=less] $ false
```

4. Use the `options` setting to specify the DVS `cache` mount option, and specify `maxnodes=1` (entering **2h*** selects the `options` setting of the second client mount for editing).

```
cray_dvs.settings.client_mount
[<cr>=set 2 entries, +=add an entry, ?=help, @=less] $ 2h*
...
cray_dvs.settings.client_mount.data.my_mount.options
[<cr>=keep 'maxnodes=1', <new value>, ?=help, @=less] $ maxnodes=1,cache
```

5. Set the client mount entries, then save changes and exit the configurator.

```
cray_dvs.settings.client_mount
[<cr>=set 2 entries, +=add an entry, ?=help, @=less] $ <cr>
...
Cray DVS Service Menu [default: save & exit - Q] $ <cr>
```

To enable only the caching of read data on a non-writable file system, enable caching as shown above, but ensure that the `readonly` setting is set to true to force the DVS mount to be read-only. In this example, it is already set to true, so only a **<cr>** is necessary to set it.

```
cray_dvs.settings.client_mount
[<cr>=set 2 entries, +=add an entry, ?=help, @=less] $ 2g*
...
cray_dvs.settings.client_mount.data.my_mount.readonly
[<cr>=set 'true', <new value>, ?=help, @=less] $ <cr>
```

When done, remember to set the client mount entries, then save changes and exit the configurator, as in step 5 above.

Use of Associated Variables and Functions

Exercise caution if using the `DVS_CACHE` environment variable or its ioctl counterpart, `DVS_SET_FILE_CACHE`. Allowing an application to bypass cached data on the file system could lead to coherency issues between the local cached file data and what is read from the backing file system. Allowing a file to read data from the backing store could also cause issues with an inode shared between cached and uncached open file handles by updating inode attributes and putting the inode out of sync with the current state of cached data.

The `ro_cache` mount option is no longer needed, except in this use case: to allow read data to be cached as possible while allowing a file to be written and still maintain proper coherency across different DVS clients.

Because DVS now supports client-side caching of write data, the `MAP_SHARED` flag can now be specified to the memory map function `mmap()` on writable cached mounts. DVS continues to support the `MAP_PRIVATE` flag as well.

How Client-side Caching Works

Client-side caching of both read and write data is implemented as a write-back type of cache. This type of cache allows write data to be targeted to local cache on the client, aggregating the data before it is later 'written back' to the backing file system storage on a DVS server, thus providing low latency and high throughput for write operations. Aggregation of write data allows the DVS client to wait until a larger amount of data needs to be written, and to then write all the data with a single network operation rather than a network operation for each write performed by an application. This enables an application to complete writes more quickly while reducing overall network traffic and the total utilization load on DVS servers. A write-back cache also enables caching of read data and reuse of previously read or written data in the cache.

Note that DVS does not provide perfect cache coherency across disparate client nodes. Instead, it provides close-to-open coherency, which is described in *About the Close-to-Open Coherency Model* on page 48.

DVS write-back cache uses the standard Linux VFS page-cache mechanism and address-space operations interface, to which it adds its own write-back model and heuristics. The Linux kernel-provided functionality writes back cached data and reclaims cache pages as memory pressure increases or when a specified time limit is reached. To this, DVS adds the tracking of data being cached on the client, and when an optimal amount of contiguous data has been written, it is immediately written back to the DVS server with a single DVS remote memory access (RMA) operation. This provides a more steady and optimal flow of data to DVS servers, and it helps to avoid large amounts of cached data waiting to be written at file close time or when the cache timeout is reached. This DVS-provided functionality also helps to minimize the risk of large amounts of cached data contributing to out-of-memory issues on client nodes, because kernel memory management is able to reclaim the 'clean' (i.e., already written back) pages immediately without having to attempt to first initiate a write-back of the pages to the server.

4.7.1 About the Close-to-Open Coherency Model

The DVS read/write cache capability provides a close-to-open coherency model, which means:

- File read operations are only guaranteed to see file data that was available on the server at the time the file was opened.
- Write data cached on the client is not guaranteed to be written back to, and thus visible on, the DVS server and backing file system storage until file close time.

This does not imply that server data newer than file open time cannot be read by the client or that some amount of client write data will not be written to the server prior to file close. Rather, that file open and close are the only times this can be **guaranteed**. To preserve the close-to-open coherency, any remaining 'dirty' (i.e., not yet written

back) cached client data for a file is written back to the DVS server at file close time, and any cached client data is invalidated at file open if the file on the server is newer than the cached data. This coherency model is similar to that provided by NFS.

How Granularity Affects Coherency

The kernel VFS interface provides caching with the granularity of a kernel page, which is 4 Kb for the SLES12 kernel. This implies that 4Kb is the smallest amount of data that is stored in cache and that data is read from and written to the server in a minimum of 4Kb blocks. This page granularity affects coherency. Shared file access from different DVS clients are coherent only at page cache size boundaries. If two DVS clients attempt to write within the boundary of the same kernel page, coherency cannot be preserved, even if the two clients are writing non-overlapping byte ranges within that page. This is because both clients would attempt to cache and write-back the entire page, unable to see the data from the other client and ultimately conflicting with each other when writing back their cache pages. Cache access from separate clients can preserve coherency if they maintain the cache page size boundaries. If the separate clients each only write and cache distinct pages, then there will be no conflicts on the server when the pages are written back.

If More Fine-grained Control is Needed

Beyond using file opens and closes to control coherency, user applications can achieve more fine-grained control, if required. Here are two possible options.

Linux file operations with file locking	Applications can use the generic Linux file operations sync, flush, and invalidate (for example, `madvise` or `fadvise`) to force write-back of data or to clear existing data from the local cache. Using those operations in conjunction with file locking can enable applications on different client nodes to preserve coherency. Either approach by itself may not be sufficient to preserve coherency.
`O_DIRECT` flag	Applications can open a file with the `O_DIRECT` flag to trigger any operations for that file to bypass the local cache entirely and read and write directly to the backing file system on the server, just as a non-cached DVS file system would do normally. However, this poses the risk of a cached file and a direct IO file on a client not being consistent with each other, because the direct IO file sees only the server data and not any local cached data, and file operations using the local cache may not see the direct written data from the server.

4.8 Force a Cache Revalidation on a DVS Mount Point

About this task

Enabling the `attrcache_timeout` option for a DVS client mount can improve performance because cached file attributes on a DVS client reduce the need to make requests of the DVS server. However, this used to create a problem if the file attributes of the DVS mounted file system changed, because the only way to revalidate the cache was to wait the entire timeout, often as long as four hours (14400 seconds). This procedure enables a system administrator with root privilege to force a cache revalidation at any time without having to wait for the timeout to expire.

Procedure

1. Force a cache revalidation on a DVS client using one of the following methods:
   - To revalidate all cached attributes on a single DVS mount point, echo 1 into that mount point's `drop_caches proc` file. The following example uses the second mount point on the client and uses the `cat` command first to confirm that is the desired mount point. To specify a different mount point, replace the `2` with some other integer (`0-n`).

     ```
     hostname# cat /proc/fs/dvs/mounts/2/mount
     hostname# echo 1 > /proc/fs/dvs/mounts/2/drop_caches
     ```

   - To revalidate all attributes across all DVS mounts, echo 1 into the universal DVS `drop_caches /proc` file. For example:

     ```
     hostname# echo 1 > /proc/fs/dvs/drop_caches
     ```

2. (Optional) Clear out other caches on the DVS client to ensure that all data is revalidated.

   ```
   hostname# echo 3 > /proc/sys/vm/drop_caches
   ```

3. (Optional) Clear out other caches on the DVS *server* to ensure that all data is revalidated.

   If an NFS file system is the underlying file system, it is also likely that the same procedure will be required on the DVS servers to allow all of the changes on the NFS file system to properly propagate out to the DVS clients. This has to do with NFS caching behavior.

   ```
   hostname# echo 3 > /proc/sys/vm/drop_caches
   ```

4.9 Disable DVS Fairness of Service

About this task

Fairness of Service, described below, is enabled by default. This procedure describes how to disable it.

DVS creates user- or job-specific request queues for clients. Originally, DVS used one queue to handle requests in a FIFO (first-in, first out) fashion. This meant that since all clients shared one queue, a demanding job could tax the server disproportionately and the other clients would have to wait until the demanding client's request(s) completed. Fairness of Service was implemented to address that issue. With it enabled, DVS creates a *list* of queues—one queue for each client and/or job. The list of queues is processed in a circular fashion. When a message thread is available, it fetches the first queue on the list, moves that queue to the end of the list, and processes the first message in that queue. This helps to distribute the workload and potentially helps contending applications perform better.

Procedure

1. Stop DVS on all servers.

   Note that stopping and restarting DVS on all server nodes would typically be done only during a maintenance interval. See *Quiesce a DVS-projected File System* on page 44 for information about how to do this.

2. Disable Fairness of Service by setting the single message queue parameter to 1.

Use one of the following kernel module parameters to set the single message queue parameter. Because these parameters cannot be set using the configurator, use the procedure *Change Kernel Module Parameters Prior to Boot using Modprobe.d Files and Simple Sync* on page 32 and the following information, both from the topic *Configure DVS using Modprobe or Proc Files* on page 30.

dvsipc_single_msg_queue

Disables fairness of service, which is enabled by default. Setting this parameter to 1 disables fairness of service by forcing DVS to use a single message queue instead of a list of queues.

- Default value: 0 (fairness of service enabled)
- To view read-only:
  cat `/sys/module/dvsipc/parameters/dvsipc_single_msg_queue`
- To change prior to boot, use the `single_msg_queue` field of the `dvs_instance_info` parameter. If no other fields in `dvs_instance_info` need to be changed, it may be easier to change the `dvsipc_single_msg_queue` parameter directly by adding these lines to
  <simple sync path>`/etc/modprobe.d/dvs-local.conf`:

```
# Disable fairness of service
options dvsipc dvsipc_single_msg_queue=1

# Enable fairness of service
options dvsipc dvsipc_single_msg_queue=0
```

- To change dynamically: N/A

dvs_instance_info

Contains the following fields, which are parameters for the DVS thread pool in the DVS IPC layer. Most of these fields can be changed through other module parameters (e.g., `dvsipc_msg_thread_limit` and `dvsipc_single_msg_queue`); however, this module parameter has priority over the individual ones and if set, will override them.

Field	Definition
thread_min	Number of threads created at startup.
thread_max	Maximum number of persistent threads.
thread_limit	Maximum number of valid threads that DVS IPC allows to exist at one time. DVS IPC will dynamically scale up the number of threads to this number as the load increases.
thread_concurrent_creates	Maximum number of IPC threads that can be in the process of forking a new thread.
thread_nice	Nice value for the threads.
single_msg_queue	Disables/enables fairness of service. Setting to 1 disables fairness of service by processing incoming requests with a FIFO method. Setting to 0 (default) groups requests into different queues (qhdrs) based on apid, and round-robbins the queues to maintain quality of service (QOS) among jobs.

Field	Definition
init_free_qhdrs	Low water mark for the pool of unused qhdrs. When the pool falls below this number, more are allocated. Used only if single_msg_queue = 0.
max_free_qhdrs	Maximum number of unused qhdrs that can exist before the system starts freeing them. Used only if single_msg_queue = 0.
Interactions among fields: thread_min <= thread_max <= thread_limit thread_concurrent_creates <= thread_limit init_free_qhdrs and max_free_qhdrs used only when single_msg_queue == 0	

IMPORTANT: For this parameter to be valid, values must be specified for all of the fields. To avoid unintentional changes, be careful when changing any field values.

- Default value: see modprobe.d examples
- To view read-only: cat `/sys/module/dvsipc/parameters/dvs_instance_info`
- To change prior to boot, add these lines to `<simple sync path>/etc/modprobe.d/dvs-local.conf` (comment out either the compute or service line, depending on the type of node(s) being configured):

```
# Set parameters for DVS thread pool in DVS IPC layer
# Defaults for compute nodes
options dvsipc dvs_instance_info=4,16,1000,0,0,0,1,1

# Defaults for service nodes
options dvsipc dvs_instance_info=16,64,1000,0,0,0,64,2048
```

  For compute nodes, this translates to dvs_instance_info = thread_min = 4, thread_max = 16, thread_limit = 1000, thread_concurrent_creates = 0, thread_nice = 0, single_msg_queue = 0, init_free_qhdrs = 1, max_free_qhdrs = 1.

  For service nodes, this translates to dvs_instance_info=thread_min = 16, thread_max = 64, thread_limit = 1000, thread_concurrent_creates = 0, nice = 0, single_msg_queue = 0, init_free_qhdrs = 64, max_free_qhdrs = 2048.

  Note that if using the defaults for a compute node, ensure that dvsipc_config_type=0 is also set, and likewise, for a service node, ensure that dvsipc_config_type=1 for consistency.

- To change dynamically: N/A

3. Restart DVS on all servers.

4.10 Reconfigure DVS for an External NFS Server

Prerequisites

This procedure assumes that initial software installation and system configuration have occurred, though configuration may be incomplete.

About this task

This procedure reconfigures the system to add an external NFS server. Note that because DVS is a somewhat complex service, this procedure involves touching multiple service configuration templates.

Procedure

1. Update the network configuration on the DVS node to add a host, its network interfaces, and possibly a network if those interfaces are connected to a network that is not the same as the login node network.

   The following command line invokes `cfgset` to update service (`-s`) `cray_net` in config set `p0`. By specifying level (`-l`) `advanced`, all levels of configuration setting can be reviewed and changed. By specifying `interactive` mode (`-m`), any of those settings can be visited in any order. Settings of all states are visible by default in interactive mode. See the `cfgset` man page for a complete list of subcommands and options.

   NOTE: Reconfiguration tasks usually use the configurator in interactive mode to update a config set.

   ```
   smw# cfgset update --service cray_net --level advanced --mode interactive p0
   ```

2. Update the DVS configuration on the DVS node to specify how to access an external NFS server.

   ```
   smw# cfgset update -s cray_dvs -l advanced --state all p0
   ```

   In this example, the configurator is invoked in auto mode (default) to update `cray_dvs` service settings of all states (`--state all`) and levels. Auto mode ensures that all specified states and levels of configuration setting are presented in a predefined, logical order. For more information about updating the DVS configuration, see *Reconfigure DVS Interactively* on page 25

3. Configure LDAP.

   Configure this service to have the same accounts on the CLE system as on the file server. This service can be configured either for Microsoft Active Directory style of schema for LDAP or the OpenLDAP style of schema for LDAP. Alternatively, the service can be configured for NIS, if the site uses that directory service.

   ```
   smw# cfgset update -S all -s cray_auth --level advanced p0
   ```

4. Configure automount files on the DVS node using Simple Sync.

   This step uses Simple Sync to push some automount files to a specific DVS node. In this example, the specific DVS node is `c0-0c0s0n2`, the automount files are found on the SMW in `/home/crayadm/etc`, the NFS server name is CSS, and the CLE config set is p0. Substitute the correct node, filepath, NFS server name, and config set name for this site.

   ```
   smw# cd /var/opt/cray/imps/config/sets/p0/files
   smw# mkdir -p simple_sync/hardwareid/c0-0c0s0n2/files/etc/auto.master.d
   smw# cd simple_sync/hardwareid/c0-0c0s0n2/files/etc
   smw# cp -p /home/crayadm/etc/auto.css .
   smw# cp -p /home/crayadm/etc/auto.master.d/css.autofs auto.master.d
   ```

5. Run `cray-ansible` on the DVS node to pull that content to the DVS node.

```
dvs# /etc/init.d/cray-ansible start
```

4.11 Improve Performance and Scalability of GPFS (Spectrum Scale) Mounts

For procedures to configure a DVS mount of GPFS (Spectrum Scale), see *XC Series GPFS Software Installation Guide* (S-2569), which is available at *http://pubs.cray.com*.

The performance of a typical GPFS (Spectrum Scale) read-write mount can be improved by setting `attrcache_timeout` to enable client-side attribute caching. The more frequently content changes, the smaller the timeout value should be (e.g., 3 to 10 seconds).

When highly scalable access to read-only (or read-mostly) data, such as shared libraries and input files, is needed, Cray recommends configuring a separate DVS compute node mount of Spectrum Scale using the following options:

`loadbalance`

The `loadbalance` option allows all servers to be used to access the same file. The server is chosen based on the compute node doing the access. This maximizes the scalability of both reads and meta operations like open, stat, and close because all servers can be used for every file.

Enable this option using the `loadbalance` setting of a `client_mount` setting in the configurator or the DVS configuration worksheet.

`cache`

The `cache` option allows the compute node to cache file data in memory for the duration of an application execution. At the end of every application, the compute node page cache is cleared to maximize the availability of huge pages for the next application (from the same job or a different job).

This option does not have a corresponding setting; enable it using the `options` setting of a `client_mount` setting in the configurator or the DVS configuration worksheet.

`attrcache_timeout=5`

The `attrcache_timeout` option allows the results of file lookups (both successful and unsuccessful) to be cached on the compute nodes. Setting it to 5 (or more than 5 depending on how often the underlying content changes) allows the caching to be effective during the start-up of an application. DVS will detect a stale or deleted existing file on every open, so `attrcache_timeout` is needed primarily to prevent a previous failed access (due to ENOENT) from masking the subsequent creation of that file for too long.

Set this option using the `attrcache_timeout` setting of a `client_mount` setting in the configurator or the DVS configuration worksheet.

Having a dedicated mount configured with these options has the advantage of providing maximum scalability of access. However, having more than one mount has the disadvantage of requiring users, jobs, and applications to know which mount (path) to use when. This could be mitigated by setting environment variables (e.g., `LD_LIBRARY_PATH` to use the `loadbalance` path) for each path.

5 DVS Configuration Settings, Mount Options, Environment Variables, and ioctl Interfaces

The following reference topics provide information about DVS configuration settings, additional mount options, user environment variables, and `ioctl` interfaces.

5.1 DVS Configuration Settings and Mount Options

Administrators configure Cray DVS (Data Virtualization Service) using the configurator and `modprobe.d` files in the Simple Sync directory structure, not on the command line or by adding lines to `/etc` files (e.g., `/etc/fstab`). The following sections describe the settings that are available within the configurator for that purpose. The first two sections cover settings that are part of the client mount setting. The last section covers the remaining DVS settings, which are kernel module parameters.

> **IMPORTANT:** When configurator guidance indicates a relationship or interaction between one or more settings, it is advisory only; the configurator does not automatically check to ensure compatibility among settings. However, the underlying implementation of DVS is unchanged, and it **does** automatically set related mount options when certain mount options are specified. To prevent mount failure, enter setting values that are compatible, in accordance with the instructions in this publication.

Client Mount Settings

`reference`

A human-readable string—a name—that is used to uniquely identify a client mount. `reference` cannot be set by accepting the default: a non-empty string is required.

- Full setting name: `cray_dvs.settings.client_mount.data.reference.`*`REF-NAME`* (where *`REF-NAME`* is the user-provided client mount reference name)
- Level: `basic`
- Default value: `''` (empty string)

- Associated environment variable: none
- Related settings/options: Because this is the key field of a client mount setting entry, each setting within the client mount setting includes this string in its full setting name.

mount_point

A string that specifies the full pathname on the client of the projected file system. `mount_point` cannot be set by accepting the default: a non-empty string is required.

- Full setting name: `cray_dvs.settings.client_mount.data.`*`REF-NAME`*`.mount_point`
- Level: `basic`
- Default value: `''` (empty string)
- Associated environment variable: none
- Related settings/options: none

spath

A string that specifies the full pathname on the DVS server of the file system that is to be projected for a client mount. It must be an absolute path and it must exist on the DVS server. `spath` cannot be set by accepting the default: a non-empty string is required.

- Full setting name: `cray_dvs.settings.client_mount.data.`*`REF-NAME`*`.spath`
- Level: `basic`
- Default value: `''` (empty string)
- Associated environment variable: none
- Related settings/options: none

server_groups

A list of node groups that will function as DVS servers for a client mount. Enter one node group per line (see *About Node Groups* on page 105). `server_groups` cannot be set by accepting the default: a non-empty list is required.

> **IMPORTANT:** DVS servers should be dedicated because they use unlimited amounts of CPU and memory resources based directly on the I/O requests sent from DVS clients. Avoid using nodes that have other services (Lustre nodes, login nodes, etc.) or are tier2 nodes.

- Full setting name: `cray_dvs.settings.client_mount.data.`*`REF-NAME`*`.servers`
- Level: `basic`
- Default value: `[]` (empty list)
- Associated environment variable: none
- Related settings/options: Functionally equivalent to the `nodename` or `nodefile` "additional" option in the `options` setting of the client mount setting. The use of those two additional options is deprecated.

client_groups

A list of node groups that will function as DVS clients for a client mount. Enter node groups one per line. Unlike `server_groups`, `client_groups` can be set to an empty

list. If no node groups are specified, the mount will be performed on all suitable compute nodes (a compute node functioning as a DVS server is an example of an unsuitable node). This is common.

- Full setting name: `cray_dvs.settings.client_mount.data.`*`REF-NAME`*`.clients`
- Level: `basic`
- Default value: `[]` (empty list)
- Associated environment variable: none
- Related settings/options: none

`loadbalance`

Used to specify loadbalance mode, which more evenly distributes loads across DVS servers. Loadbalance mode is valid only for read-only mounts. For more information, see *DVS Loadbalance Mode*.

- Full setting name: `cray_dvs.settings.client_mount.data.`*`REF-NAME`*`.loadbalance`
- Level: `advanced`
- Default value: `false` or `0`
- Associated environment variable: none
- Related settings/options: When `loadbalance` is enabled, the underlying DVS implementation automatically sets the `readonly` setting to `true` and sets these additional options: `cache=1`, `failover=1`, `maxnodes=1`, and `hash_on_nid=0`. Cray recommends setting the `attrcache_timeout` setting as well to take advantage of the mount being read-only. If `loadbalance` is enabled, leave the `readonly` setting unconfigured or set it to `true` to maintain consistency with the way DVS implements `loadbalance`.

`attrcache_timeout`

Enables client-side attribute caching, which can significantly increase performance, most notably in pathname lookup situations. File attributes and `dentries` for `getattr` requests, pathname lookups, etc. are read from DVS servers and cached on the DVS client for *`n`* seconds. Subsequent lookups or `getattr` requests use the cached attributes until the timeout expires, at which point they are read and cached again on first reference. When attribute caching is disabled, DVS clients must send a lookup request to a DVS server for every level of a pathname, and repeat this for every pathname operation. When it is enabled, it sends a lookup request to a DVS server for every level of a pathname once per *`n`* seconds.

> **NOTICE:** An administrator with root privilege can force a cache revalidation at any time, not just when the timeout has expired. See *Force a Cache Revalidation on a DVS Mount Point* on page 49.

- Full setting name: `cray_dvs.settings.client_mount.data.`*`REF-NAME`*`.attrcache_timeout`
- Level: `advanced`

- Default value: `14400` seconds for read-only mounts.

  IMPORTANT: This is the configurator default. The underlying DVS implementation default is `3` seconds, which is safer for read-write mounts. This means that to enhance system performance for read-only mounts, configure this setting by accepting the configurator default (or entering some other value). Leaving this setting unconfigured will result in the underlying default being used.

- Associated environment variable: none
- Related settings/options: The Ansible play that consumes DVS configuration data prevents use of this mount option for read-write file systems due to the risk of file system corruption. Run-time mounts not accompanied by that Ansible play do not have that safeguard. In such cases, if a read-write mount is created, it is safe to leave `attrcache_timeout` unconfigured so that the underlying default is used.

`readonly`

Determines whether the client mount is read-only or read-write. If intending to enable client-side caching of read data on a non-writable file system, use this `readonly` setting to force the DVS mount to be read-only. This will disable write caching.

- Full setting name: `cray_dvs.settings.client_mount.data.`*`REF-NAME`*`.readonly`
- Level: `basic`
- Default value: `true` or `1`

  IMPORTANT: This is the configurator default. The underlying DVS implementation default is `false`. Leaving this setting unconfigured will result in the underlying DVS default being used.

- Associated environment variable: none
- Related settings/options: When `loadbalance` is enabled, DVS automatically enables `readonly` but the configurator does not, so either leave this setting unconfigured or accept the configurator default. If the `attrcache_timeout` setting is set for this client mount, `readonly` should be enabled (set to true) in the configurator/worksheet. If the `cache` option is specified in the `options` setting for this client mount, enabling `readonly` is the only way to enable read caching without enabling write caching as well).

`options`

Provides the only way to specify mount options in addition to the ones already specified in the other mount point settings. Enter a string with mount options separated by comma and no spaces. For information about available options and their implications, see *Additional Options for Use in the Options Setting of a Client Mount*. Note that it is necessary to specify `maxnodes=1` here for a read-write client mount of an NFS or other non-cluster, non-coherent file system.

- Full setting name: `cray_dvs.settings.client_mount.data.`*`REF-NAME`*`.options`
- Level: `advanced`
- Default value: "" (empty string)

- Associated environment variable: none
- Related settings/options: Options contained in this setting will be appended to the mount options specified in other settings. Any that are functionally redundant with settings already configured (such as *nodename*/*nodefile*, which are redundant with the `server_groups` setting) will override those settings.

Additional Options for Use in the Options Setting of a Client Mount

All of the mount options listed in this section can be used only in the options setting of a client mount setting in the configurator or DVS configuration worksheet. The options setting is level advanced, so specify **`-l advanced`** when invoking the configurator to be able to use these mount options.

`atomic` / `noatomic`

`atomic` enables atomic stripe parallel mode. This ensures that stripe parallel requests adhere to POSIX read/write atomicity rules. DVS clients send each I/O request to a single DVS server to ensure that the bytes are not interleaved with other requests from DVS clients. The DVS server used to perform the read, write, or metadata operation is selected using an internal hash involving the underlying file or directory inode number and the offset of data into the file relative to the DVS block size.

`noatomic` disables atomic stripe parallel mode. If there are multiple DVS servers and neither loadbalance nor cluster parallel mode is specified, DVS stripes I/O requests across multiple servers and does not necessarily adhere to POSIX read/write atomicity rules if file locking is not used.

- Default value: `noatomic` or `0`
- Associated environment variable: `DVS_ATOMIC`
- Related settings/options: none

`attrcache_timeout`

Do not use this option in the `options` setting. Use the **configurator setting** (`attrcache_timeout`) instead.

`blksize=`*`n`*

`blksize=`*`n`* sets the DVS block size to *`n`* bytes. Used in striping.

- Default value: `524288` (512 KB)
- Associated environment variable: `DVS_BLOCKSIZE`
- Related settings/options: none

`cache` / `nocache`

`cache` enables client-side caching of both read and write data. The client node caches reads from the DVS server node, caches writes from user applications that are aggregated and later 'written back' to the backing file system storage on the DVS server node, and provides data to user applications from the page cache if possible, instead of performing a data transfer from the DVS server node. For more information, see *DVS Client-side Write-back Caching can Yield Performance Gains* on page 46. Cray DVS is not a clustered file system; no coherency is maintained among multiple DVS client nodes reading and writing to the same file. If cache is enabled and data consistency is required, applications must take care to synchronize their accesses to the shared file.

`nocache` disables client-side read/write caching.

- Default value: `nocache` or `0`
- Associated environment variable: `DVS_CACHE` (use with caution)
- Related settings/options: When `loadbalance` is enabled, DVS automatically enables `cache`. If `readonly` enabled and the `cache` option is used, the client node will cache only read data (this is equivalent to disabling write caching).

  IMPORTANT: If enabling read/write caching, read *DVS Client-side Write-back Caching can Yield Performance Gains* on page 46 to understand the implications and prevent data corruption.

cache_read_sz

`cache_read_sz` is a limit that can be specified to prevent reads or writes over this size from being cached in the Linux page cache.

- Default value: `0`
- Associated environment variable: `DVS_CACHE_READ_SZ`
- Related settings/options: If `cache` is not enabled, DVS ignores `cache_read_sz`.

closesync / noclosesync

`closesync` enables data synchronization on last close of a file. When a process performs the final close of a file descriptor, in addition to forwarding the close to the DVS server, the DVS server node waits until data has been written to the underlying media before indicating that the close has completed. Because DVS does not cache data on client nodes (unless the `cache` option is used) and has no replay capabilities, this ensures that data is not lost if a server node crashes after an application has exited.

`noclosesync` causes DVS to return a `close()` request immediately.

- Default value: `noclosesync` or `0`
- Associated environment variable: `DVS_CLOSESYNC`
- Related settings/options: The `closesync` option is redundant with periodic sync, which is enabled by default. Because periodic sync is more efficient than `closesync`, Cray recommends letting periodic sync take care of data synchronization instead of using this mount option. (See *Periodic Sync Promotes Data and Application Resiliency* on page 70.)

datasync / nodatasync

`datasync` enables data synchronization. The DVS server node waits until data has been written to the underlying media before indicating that the write has completed. Can significantly impact performance.

`nodatasync` causes a DVS server node to return from a write request as soon as the user's data has been written into the page cache on the server node.

- Default value: `nodatasync` or `0`
- Associated environment variable: `DVS_DATASYNC`
- Related settings/options: none

deferopens / nodeferopens

`deferopens` defers DVS client open requests to DVS servers for a given set of conditions. When a file is open in stripe parallel mode or atomic stripe parallel mode, DVS clients send the open request to a single DVS server only. Additional open requests are sent as necessary when the DVS client performs a read or write to a different server for the first time. The `deferopens` option deviates from POSIX specifications. For example, if a file was removed after the initial open succeeded but before deferred opens were initiated by a read or write operation to a new server, the read or write operation would fail with `errno` set to `ENOENT` because the open was unable to open the file.

`nodeferopens` disables the deferral of DVS client open requests to DVS servers. When a file is open in stripe parallel mode or atomic stripe parallel mode, DVS clients send open requests to all DVS servers denoted by nodename or nodefile.

- Default value: `nodeferopens` or `0`
- Associated environment variable: `DVS_DEFEROPENS`
- Related settings/options: The `deferopens` option must be used if the `dwfs` option is used.

distribute_create_ops / nodistribute_create_ops

`distribute_create_ops` causes DVS to change its hashing algorithm so that create and lookup requests are distributed across all of the servers, as opposed to being distributed to a single server. This applies to creates, mkdirs, lookups, mknods, links, and symlinks.

`nodistribute_create_ops` causes DVS to use its normal algorithm of using just one target server.

- Default value: `nodistribute_create_ops` or `0`
- Associated environment variable: none
- Related settings/options: none

dwfs / nodwfs

`dwfs` specifies that the remote file system mounted under DVS is dwfs (DataWarp file system). This should be used even if there are layers between DVS and dwfs (e.g., DVS -> accountfs -> dwfs).

`nodwfs` is the default, where DVS does not support a DataWarp file system.

- Default value: `nodwfs` or `off`.
- Associated environment variable: none
- Related settings/options: The `dwfs` option can be used only if the `deferopens` option is used.

failover / nofailover

`failover` enables failover and failback of DVS servers. If all servers fail, operations for the mount point behave as described by the `retry` option until at least one server is rebooted and has loaded DVS. If multiple DVS servers are listed for a client mount and one or more of the servers fails, operations for that mount continue by using the subset of servers still available. When the downed servers are rebooted and start DVS, any client mounts that had

performed failover operations failback to once again include the servers as valid nodes for I/O forwarding operations.

`nofailover` disables failover and failback of DVS servers. If one or more servers for a given client mount fail, operations for that mount behave as described by the `retry` or `noretry` option specified for the client mount.

- Default value: `failover` or `1`
- Associated environment variable: none
- Related settings/options: When the `failover` option is enabled (occurs automatically when `loadbalance` is enabled), the `noretry` option cannot be enabled.

hash

Except in cases of extremely advanced administrators or specific advice from DVS developers, do not use the `hash` mount option. The best course of action is to let DVS use its default value. The `hash` option has three possible values:

fnv-1a `hash=fnv-1a` offers the best overall performance with very little variation due to differing numbers of servers.

jenkins `hash=jenkins` is the hash that DVS previously used. It is included in the unlikely case of end-case pathological issues with the `fnv-1a` hash, but it has worse overall performance.

modulo `hash=modulo` does not do any hash at all, but rather takes the `modulo` of the seed that it is given. This option can potentially have high load balancing characteristics, but is extremely vulnerable to pathological cases such as file systems that only allocate even numbered inodes or a prime number of servers.

- Default value: `fnv-1a`
- Associated environment variable: none
- Related settings/options: none

hash_on_nid

With `hash_on_nid` set to `on`, DVS uses the nid of the client as the hash seed instead of using the file inode number. This effectively causes all request traffic for the compute node to go to a single server. This can help metadata operation performance by avoiding lock thrashing in the underlying file system when each process on a set of DVS clients is using a separate file.

- Default value: `off` or `0`
- Associated environment variable: none
- Related settings/options: When `hash_on_nid` is enabled (set to 1), DVS automatically sets the `hash` option to `modulo`. When `loadbalance` is enabled, DVS automatically sets `hash_on_nid=0`.

killprocess / nokillprocess

`killprocess` enables killing processes that have one or more file descriptors with data that has not yet been written to the backing store. DVS provides this option to minimize the risk of silent data loss, such as when data still resides in

the kernel or file system page cache on the DVS server after a write has completed.

`nokillprocess` disables the killing of processes that have written data to a DVS server when a server fails. When a server fails, processes that have written data to the server are not killed. If a process continues to perform operations with an open file descriptor that had been used to write data to the server, the operations fail (with `errno` set to `EHOSTDOWN`). A new open of the file is allowed, and subsequent operations with the corresponding file descriptor function normally.

- Default value: `killprocess` or `1`
- Associated environment variable: `DVS_KILLPROCESS`
- Related settings/options: With the periodic sync feature (enabled by default), DVS servers attempt to `fsync` dirty files to minimize the number of processes that are killed and will also `fsync` a dirty file's data when the file is closed. If periodic sync is disabled (not recommended), the `killprocess` option alone cannot fully guarantee prevention of silent data loss (though it is highly unlikely) because a `close()` does not guarantee that data has been transferred to the underlying media (see the `closesync` option).

`loadbalance`/ `noloadbalance`

Do not use this option in the `options` setting. Use the **configurator setting** (`loadbalance`) instead.

`magic`

`magic` defines what the expected file system magic value for the projected file system on the DVS servers should be. When a DVS client attempts to mount the file system from a server, it verifies that the underlying file system has a magic value that matches the specified value. If not, the DVS client excludes that DVS server from the list of servers it uses for the mount point and prints a message to the system console. Once the configuration issue on the DVS server has been addressed and the client mounts the correct file system, DVS can be restarted on the server. All clients subsequently verify that the server is configured correctly and include the server for that mount point. Many file system magic values are defined in the `/usr/include/linux/magic.h` file. Commonly used magic values on Cray systems are:

NFS	0x6969
GPFS	0x47504653
BTRFS	0x9123683E
TMPFS	0x01021994

- Default value: the underlying file system's magic value
- Associated environment variable: none
- Related settings/options: none

`maxnodes`

`maxnodes` is used in configuring DVS modes. See *DVS Modes* on page 9.

- Default value: number of nodes available (`nnodes`)
- Associated environment variable: `DVS_MAXNODES`
- Related settings/options: When `loadbalance` is enabled, DVS automatically sets `maxnodes=1`.

`mds`

`mds=server`, where `server` is the hostname for a DVS server, specifies which DVS server to use for metadata operations. Metadata will be sent only to the server specified. Used only for DataWarp file systems.

- Default value: none
- Associated environment variable: none
- Related settings/options: When the `dwfs` option is used, `mds` must be used. Cray recommends not using `mds` if the `dwfs` option is not used.

`nodefile`

`nodefile` is the file name of a file with a list of server nodes specified as cnames separated by a colon (:) and no spaces. Do not use this option in the `options` setting. Use the **configurator setting** (`server_groups`) instead.

`nodename`

`nodename` is a list of server nodes specified as cnames separated by a colon (:) and no spaces. Do not use this option in the `options` setting. Use the **configurator setting** (`server_groups`) instead.

`path`

Do not use this option in the `options` setting. Use the **configurator setting** (`spath`) instead.

`retry` / `noretry`

`retry` enables the retry option, which affects how a DVS client node behaves in the event of a DVS server node going down. If `retry` is specified, any user I/O request is retried until it succeeds, receives an error other than a "node down" indication, or receives a signal to interrupt the I/O operation.

`noretry` disables retries of user I/O requests when the DVS server receiving the request is down.

- Default value: `retry` or `1`
- Associated environment variable: none
- Related settings/options: When the `failover` option is enabled, the `noretry` option cannot be enabled.

`ro_cache` / `no_ro_cache`

`ro_cache` enables read-only caching for files on writable client mounts. Files opened with read-only permissions in `ro_cache` mode are treated as if they were on a DVS read-only cached client mount. If the file has any concurrent open that has write permissions, all instances of that file revert to the default `no_ro_cache` mode for the current and subsequent reads.

`no_ro_cache` disables read-only caching for files on writable client mounts.

- Default value: `no_ro_cache` or `0`
- Associated environment variable: none
- Related settings/options: none

userenv / nouserenv

`userenv` specifies that DVS must honor end user environment variable overrides for DVS mount options.

`nouserenv` allows the administrator to block end user environment variable overrides for DVS mount options.

- Default value: `userenv` or `1`
- Associated environment variable: none
- Related settings/options: none

Kernel Module Parameter Settings

Setting kernel module parameters during initial system configuration is just like setting any other configuration data values. However, changing them later to reconfigure a service may require reloading the module to enable the change to take effect. That is why Cray recommends viewing all module parameter settings as permanent once they are set during initial configuration, before modules are loaded.

dvsipc_heartbeat_timeout

DVS inter-process communication (IPC) heartbeat timeout, in seconds. This parameter is no longer used; it has been preserved only to maintain backwards compatibility with existing DVS config files. Leave this parameter unconfigured or accept the default value.

- Full setting name: `cray_dvs.settings.dvsipc_heartbeat_timeout`
- Level: `advanced`
- Default value: `60`
- Related settings/options: none

dvs_debug_mask

Hex mask of the bits to set to enable debug output to be printed to the console. It can flood the console file and negatively affects performance, so it is generally used only for development or troubleshooting. Different mask values enable the output of different sets of debug information. Leave this parameter unconfigured or accept the default value.

- Full setting name: `cray_dvs.settings.dvs_debug_mask`
- Level: `advanced`
- Default value: `0` (disabled)
- Related settings/options: none

Additional Kernel Module Parameters

There are many DVS kernel module parameters that cannot be set within the configurator. For a list of them and instructions on how to set them, see *Configure DVS using Modprobe or Proc Files* on page 30.

5.2 DVS Environment Variables

By default, user environment variables allow client override of options specified during configuration and are evaluated whenever a file is opened by DVS. However, if the `nouserenv` option is included in the `options` setting of the `client_mount` configuration setting, then user environment variables are disabled for that client mount.

The following environment variables are for use in the default case:

Variable Name	Options	Purpose
`DVS_ATOMIC`	`on|off`	Overrides the `atomic` or `noatomic` mount options.
`DVS_BLOCKSIZE`	*n*	A nonzero number, *n* overrides the `blksize` mount option.
`DVS_CACHE`	`on|off`	Overrides the `cache` or `nocache` mount options. Exercise caution if using this variable. Allowing an application to bypass cached data on the file system could lead to coherency issues between the local cached file data and what is read from the backing file system. Allowing a file to read data from the backing store could also cause issues with an inode shared between cached and uncached open file handles by updating inode attributes and putting the inode out of sync with the current state of cached data. For additional cautions about write caching, see "Client-side Write-back Caching may not be Suitable for all Applications" in *DVS Client-side Write-back Caching can Yield Performance Gains* on page 46.
`DVS_CACHE_READ_SZ`	*n*	A positive integer, *n* overrides the `cache_read_sz` mount option.
`DVS_CLOSESYNC`	`on|off`	Overrides the `closesync` or `noclosesync` mount options. **NOTE:** Periodic sync functions similarly to the DVS `closesync` mount option, but it is more efficient and is enabled by default. Cray recommends not using `closesync` or this associated environment variable.
`DVS_DATASYNC`	`on|off`	Overrides the `datasync` or `nodatasync` mount options. **NOTE:** Setting `DVS_DATASYNC` to `on` can slow down an application considerably. The periodic sync feature, enabled by default, is a better way to synchronize data. See *Periodic Sync Promotes Data and Application Resiliency* on page 70.
`DVS_DEFEROPENS`	`on|off`	Overrides the `deferopens` or `nodeferopens` mount options.
`DVS_KILLPROCESS`	`on|off`	Overrides the `killprocess` or `nokillprocess` mount options.
`DVS_MAXNODES`	*n*	A nonzero number, *n* overrides the `maxnodes` mount option. The specified value of `maxnodes` must be greater than zero and less than or equal to the number of server nodes specified on the mount, otherwise the variable has no effect.

5.3 DVS ioctl Interfaces

The following are provided for advanced users who require DVS `ioctl` interfaces. Most are correlates of environment variable and mount options with the same name.

Variable Name	Argument Type/Size	Purpose
`DVS_GET_FILE_ATOMIC` / `DVS_SET_FILE_ATOMIC`	signed 16-bit (must be 0 or 1 for `SET`)	Retrieves/sets the `atomic` option value for a file on a DVS mount.
`DVS_GET_FILE_BLK_SIZE` / `DVS_SET_FILE_BLK_SIZE`	signed 32-bit (must be > 0 for `SET`)	Retrieves/sets the DVS block size for a file on a DVS mount.
`DVS_GET_FILE_CACHE` / `DVS_SET_FILE_CACHE`	signed 16-bit (must be 0 or 1 for `SET`)	Retrieves/sets the cache option for a file on a DVS mount. Exercise caution if using `DVS_SET_FILE_CACHE`. Allowing an application to bypass cached data on the file system could lead to coherency issues between the local cached file data and what is read from the backing file system. Allowing a file to read data from the backing store could also cause issues with an inode shared between cached and uncached open file handles by updating inode attributes and putting the inode out of sync with the current state of cached data. For additional cautions about write caching, see "Client-side Write-back Caching may not be Suitable for all Applications" in *DVS Client-side Write-back Caching can Yield Performance Gains* on page 46.
`DVS_GET_FILE_CACHE_READ_SZ` / `DVS_SET_FILE_CACHE_READ_SZ`	signed 32-bit (must be > 0 for `SET`)	Retrieves/sets the `cache_read_sz` value for a file on a DVS mount.
`DVS_GET_FILE_CLOSESYNC` / `DVS_SET_FILE_CLOSESYNC`	signed 16-bit (must be 0 or 1 for `SET`)	Retrieves/sets the `closesync` option for a file on a DVS mount.
`DVS_GET_FILE_DATASYNC` / `DVS_SET_FILE_DATASYNC`	signed 16-bit (must be 0 or 1 for `SET`)	Retrieves/sets the current `datasync` value for a file on a DVS mount.
`DVS_GET_FILE_DEFEROPENS` / `DVS_SET_FILE_DEFEROPENS`	signed 16-bit (must be 0 or 1 for `SET`)	Retrieves/sets the `deferopens` value for a file on a DVS mount.
`DVS_GET_FILE_KILLPROCESS` / `DVS_SET_FILE_KILLPROCESS`	signed 16-bit (must be 0 or 1 for `SET`)	Retrieves/sets the `killprocess` option for a file on a DVS mount.
`DVS_GET_FILE_STRIPE_WIDTH` / `DVS_SET_FILE_STRIPE_WIDTH`	signed 32-bit (must be > 0 for `SET`)	Retrieves/sets the stripe width size for a file on a DVS mount. To set the stripe width, loadbalance option must be set.
`DVS_GET_NNODES`	signed 32-bit	Retrieves the number of nodes currently available for a mount point.

Variable Name	Argument Type/Size	Purpose
DVS_GET_REMOTE_FS_MAGIC	unsigned 64-bit	Gets the remote file system type for a file on a DVS mount.
DVS_BCAST_IOCTL	struct dvs_ioctl_tunnel	Used for DataWarp to allow specialized ioctl calls to be passed through DVS to a remote server.
DVS_AUGMENTED_BCAST_IOCTL	struct dvs_augmented_ioctl_tunnel	Used for DataWarp to allow specialized ioctl calls to be passed through DVS to a remote server.
DVS_TUNNEL_IOCTL	struct dvs_ioctl_tunnel	Used for DataWarp to allow specialized ioctl calls to be passed through DVS to a remote server.
DVS_AUGMENTED_TUNNEL_IOCTL	struct dvs_augmented_ioctl_tunnel	Used for DataWarp to allow specialized ioctl calls to be passed through DVS to a remote server.

6 DVS Resiliency and Diagnostics

Cray has developed the following DVS features to promote resiliency and enable diagnosis of DVS and the Cray system:

- *DVS Supports Failover and Failback* for parallel modes. The topic describes how it works and includes example console messages.
- DVS *Periodic Sync Promotes Data and Application Resiliency* and is more efficient than the DVS mount option `closesync`. The topic describes how it works and how it can be tuned.
- *DVS Statistics Enable Analysis* of DVS performance on client and server nodes in CLE. The topic shows where the statistics are stored and describes how to enable/disable and format statistics. There are many per-mount statistics available. All currently collected statistics are listed and described in *DVS Statistics Collected* on page 76.
- *DVS Can Log Requests Sent to Servers* to aid in debugging. The topic shows an example log file and describes how to enable, disable, and reset request logging.
- *DVS Can Log Details About File System Calls* that the DVS server node makes. The topic describes what type of infromation is collected along with how to adjust buffer content.
- *DVS Lists Outstanding Client Requests*, including the DVS server node and the amount of time the request has been waiting for a response. In addition, *DVS Provides a Plugin for Node Health Checker* that outputs information on the oldest outstanding client request.

6.1 DVS Supports Failover and Failback

DVS clients use resiliency communication agent (RCA) events to determine when server nodes have failed, when DVS has been unloaded from a server node, and when server nodes have been booted and DVS is reloaded. This ensures that all clients are informed of server failures and reboots in the same manner at the same time, which reduces the underlying file system coherency traffic associated with rerouting I/O operations away from downed servers and back to rebooted servers.

Cray DVS supports failover and failback for parallel modes:

- For cluster, stripe, and atomic stripe parallel modes, add the `failover` option to the `options` setting of a `client_mount` setting in the `cray_dvs` service to specify failover and failback.
- For loadbalance mode, failover and failback are specified by default.

DVS failover and failback are done in an active-active manner. Multiple servers must be specified in the `servers` setting of a `client_mount` setting in the `cray_dvs` service for failover and failback to function. When a server fails, it is taken out of the list of servers to use for the client mount until it is rebooted. All open and new files use the remaining servers as described below. Files not using the failed server are not affected.

When failover occurs:

- If all servers fail, I/O is retried as described by the `retry` option (see *Additional Options for Use in the Options Setting of a Client Mount* on page 59).
- Any mount point using loadbalance mode automatically recalibrates the existing client-to-server routes to ensure that the clients are evenly distributed across the remaining servers. When failback occurs, this process is repeated.
- Any mount point using cluster parallel mode automatically redirects I/O to one of the remaining DVS servers for any file that previously routed to the now-down server. When failback occurs, files are rerouted to their original server.
- Any mount point using stripe parallel mode or atomic stripe parallel mode automatically restripes I/O across the remaining DVS servers in an even manner. When failback occurs, files are restriped to their original pattern.

Client System Console Message: "DVS: file_node_down: removing c0-0c2s1n3 from list of available servers for 2 mount points"

The following message indicates that a DVS server has failed.

```
DVS: file_node_down: removing c0-0c2s1n3 from list of available
servers for 2 mount points
```

In this example, `c0-0c2s1n3` is the DVS server that has failed and has been removed from the list of available servers provided in the `servers` setting of the DVS client mount.

After the issue is resolved, the following message is printed to the console log of each client of the projection:

```
DVS: file_node_up: adding c0-0c2s1n3 back to list of available servers
for 2 mount points
```

6.2 Periodic Sync Promotes Data and Application Resiliency

DVS periodic sync improves data resiliency and facilitates a degree of application resiliency so that applications may continue executing in the event of a stalled file system or DVS server failure. Without periodic sync, such an event would result in DVS clients killing any processes with open files that were written through the failed server. Any data written through that server that was only in the server's page cache and not written to disk would be lost, and processes using the file would see data corruption.

Periodic sync works by periodically performing an `fsync` on individual files with written data on the DVS servers, to ensure that those files are written to disk. For each file, the DVS client tracks when a DVS server performs a file sync and when processes on DVS clients write to it, and then notifies the DVS server when `fsync` is needed. Periodic sync functions similarly to the DVS `closesync` mount option, but it is more efficient because it is aware of which files may have written data. Unlike `closesync` and `datasync`, periodic sync can sync data over time asyncronously so that the client does not need to wait for the sync operation to complete.

DVS periodic sync is effectively enabled by default because the `sync_period_secs` parameter, which affects the amount of time between syncs on the server, is non-zero by default. The only way to effectively disable periodic sync is to set that parameter to 0. Cray recommends keeping periodic sync enabled instead of using the `closesync` mount option.

Use the following three parameters to tune DVS periodic sync behavior. Because these parameters cannot be set using the configurator, use the procedure *Change Kernel Module Parameters Prior to Boot using Modprobe.d Files and Simple Sync* on page 32 and the following information, both from the topic *Configure DVS using Modprobe or Proc Files* on page 30.

sync_num_threads

Specifies the number of threads on the DVS server that perform sync operations. The number of threads must be a minimum of 1 thread and a maximum of 32. Note that it can take up to `sync_period_secs` for changes to this value to take effect.

- Default value: 8
- To view read-only: cat `/sys/module/dvs/parameters/sync_num_threads`
- To change prior to boot, add these lines to `<simple sync path>/etc/modprobe.d/dvs-local.conf`:

```
# Set the number of threads that perform
# sync operations on a DVS server
options dvs sync_num_threads=8
```

- To change dynamically:

```
hostname# echo 8 > /proc/fs/dvs/sync_num_threads
```

sync_dirty_timeout_secs

On DVS *servers*, specifies the number of seconds that must have passed since the file was written before DVS syncs it. The objective is to reduce unnecessary sync operations for files actively being updated. Decreasing this number increases the likelihood that the file is in use when it is synced. Increasing this number increases the likelihood that processes are killed during a server failure.

On DVS *clients*, specifies the number of seconds that must have passed since the file was written before DVS asks the server for an updated sync time. Decreasing this number increases the number of DVS requests being sent. Increasing this number increases the likelihood that processes are killed during a server failure.

- Default value: 300 (servers and clients)
- To view read-only:
  cat `/sys/module/dvs/parameters/sync_dirty_timeout_secs`
- To change prior to boot, add these lines to `<simple sync path>/etc/modprobe.d/dvs-local.conf`:

```
# Set the timeout (seconds) for syncing
# dirty files on a DVS server or client
options dvs sync_dirty_timeout_secs=300
```

- To change dynamically:

```
hostname# echo 300 > /proc/fs/dvs/sync_dirty_timeout_secs
```

sync_period_secs

On DVS *servers*, specifies the number of seconds before the `sync_num_threads` syncs files on the DVS server (if necessary).

On DVS *clients*, specifies the number of seconds between checks for dirty files that need to request the last sync time from the server.

- Default value: 300 (server), 600 (client)
- To view read-only: cat `/sys/module/dvs/parameters/sync_period_secs`
- To change prior to boot, add these lines to `<simple sync path>/etc/modprobe.d/dvs-local.conf`:

```
# Set the sync period (seconds) on DVS server/client
options dvs sync_period_secs=300
```

- To change dynamically:

```
hostname# echo 300 > /proc/fs/dvs/sync_period_secs
```

A fourth `/proc` file, `/procsfs/dvs/sync_stats`, collects statistics of the syncing behavior. This setting is not tunable (read only).

6.3 DVS Statistics Enable Analysis

DVS provides both aggregate and per-mount statistics to enable performance and root-cause analysis. It reports statistics for CLE client and server nodes in the following `stats` files. Each stats file is readable and writable.

/proc/fs/dvs/stats	Contains aggregate statistics for the node. These reflect system operations that cannot be correlated to a specific DVS mount point and are therefore most interesting on DVS servers.
/proc/fs/dvs/mounts/*nnn*/stats	Contains per-mount statistics for the node, where *nnn* is a decimal integer value global to DVS on that node. Every invocation of the `mount` command creates the numbered mount directory and increments *nnn*. Every invocation of the `umount` command deletes the numbered mount directory but does not decrement *nnn*. The value of *nnn* appears in the `statsfile=/proc/fs/dvs/mounts/nnn/stats` parameter in the mount options for the mounted DVS file system. This can be obtained using the `mount -t dvs` command. More information about each of these mount points can be obtained by viewing the mount file that resides in the same directory (`/proc/fs/dvs/mounts/nnn/mount`).
/proc/fs/dvs/ipc/stats	Contains DVS interprocess communication (IPC) statistics, such as bytes transferred and received, NAK counts, and so forth. It also contains message counts by type and size.

NOTE: The `mount -t dvs` list produced by Linux will no longer show the `mntid=nnn` flag, but will instead show `statsfile=/proc/fs/dvs/mounts/nnn/stats`. This enables easier scripting: simply parse the output for comma-separated options, pick out the `statsfile` option, and use whatever value it provides as the stats file path.

How to Collect and Report DVS Statistics

DVS statistics are enabled and collected by default. Statistics can be controlled by

- specifying a module parameter at module load time
- writing text values into the corresponding stats file

Use one of the following parameters to enable and control DVS statistics. Because these parameters cannot be set using the configurator, use the procedure *Change Kernel Module Parameters Prior to Boot using Modprobe.d Files and Simple Sync* on page 32 and the following information, both from the topic *Configure DVS using Modprobe or Proc Files* on page 30.

dvsproc_stat_control

(deprecated) Controls DVS statistics. This legacy parameter has been maintained for backward compatibility, but values are overridden by `dvsproc_stat_defaults`, if specified.

- Default value: 1 (enabled)
- To view: cat `/sys/module/dvsproc/parameters/dvsproc_stat_control`
- To change prior to boot, add these lines to *<simple sync path>*`/etc/modprobe.d/dvs-local.conf`:

```
# Disable DVS statistics
options dvsproc dvsproc_stat_control=0

# Enable DVS statistics
options dvsproc dvsproc_stat_control=1
```

- To change dynamically:

  This is root writable at `/sys/module/dvsproc/parameters/dvsproc_stat_control`, but changes should be made only through the `/proc/fs/dvs/stats` interface, as shown in this example.

```
hostname# echo 0 > /proc/fs/dvs/stats
hostname# echo 1 > /proc/fs/dvs/stats
hostname# echo 2 > /proc/fs/dvs/stats
```

dvsproc_stat_defaults

Controls DVS statistics. Use this parameter to disable/enable and format DVS statistics. The options that can be specified are listed in *Option Values for Controlling DVS Statistics* on page 74.

- Default values: enable, legacy, brief, plain, notest
- To view: cat `/sys/module/dvsproc/parameters/dvsproc_stat_defaults`
- To change prior to boot, add these lines to *<simple sync path>*`/etc/modprobe.d/dvs-local.conf`:

```
# Disable/enable and format DVS statistics
options dvsproc
dvsproc_stat_defaults="enable,legacy,brief,plain,notest"
```

- To change dynamically:

  This is root writable at `/sys/module/dvsproc/parameters/dvsproc_stat_defaults`, but changes should be made only through the `/proc/fs/dvs/stats` interface, as shown in this example.

```
hostname# echo disable > /proc/fs/dvs/stats
hostname# echo enable > /proc/fs/dvs/stats
hostname# echo reset > /proc/fs/dvs/stats
hostname# echo json,pretty > /proc/fs/dvs/stats
```

To sample DVS statistics, read the contents of the corresponding stats file. All output is readable (ASCII) and in a self-describing format. Note that in future releases, Cray may change the order of lines and may deprecate (or remove as obsolete) individual statistics.

Control and Format Options

Use the options and values described in this table to control the collection and reporting of statistics.

- Multiple options can be specified, separated by commas, semicolons, spaces, or line-feeds.
- If incompatible options within the same category are specified (e.g., "verbose,brief"), the last option specified will govern.
- If incompatible options from different categories are specified (e.g., "disable,json"), the option in a higher level category will take precedence. The order of precedence (decreasing) for option categories is: control, format, flags. For example, control options such as "disable" take precedence over format options such as "json."
- Options not specified will continue to use their existing values.

Table 1. Option Values for Controlling DVS Statistics

Option Category	Option Value	Result
legacy	0	Disables statistics collection and reporting (deprecated).
	1	Enables statistics collection and reporting (deprecated).
	2	Resets statistics (deprecated). It is not used with module parameters.
control	disable	Disables statistics collection and reporting.
	enable	Enables statistics collection and reporting.
	reset	Resets statistics.
format	help	Displays current information about the statistics values collected and reported.
	legacy	Presents statistics in text format (legacy format).
	flat	Displays statistics in flattened text format.
	json	Displays statistics in JSON structured format.
flags	brief | verbose	• Brief: omits statistics that have not been collected. • Verbose: displays all statistics, even those that have not been collected. Applies to flat and json formats only.
	plain | pretty	• Plain: uses a more machine-readable presentation. • Pretty: uses a more human-readable presentation. Applies to flat and json formats only.

Option Category	Option Value	Result
	test | notest	• Test: overrides data with a test pattern, which means the statistics data is replaced with fixed test data designed to be used for regression testing of the formatting alone. • Notest: removes the test pattern override and restores normal operation.

These combinations of format and flag options are common:

flat, plain Structured data is displayed as key/value pairs, one pair per line, key and value separated by a single space. The format is: [container.[container.]...]key value[,value[,value...]]. Multiple comma-separated values represent an array of values.

flat, pretty Same as "flat, plain" except that the single space is expanded to produce columns to enhance readability.

json, plain Structured data is displayed in JSON format. It consists of a single, unnamed JSON object that contains key/value pairs. There are no spaces or line-feed characters to enhance readability.

json, pretty Same as "json, plain" except that spaces and line-feeds are added to enhance readability.

These examples illustrate the order of precedence when the options specified are incompatible:

- "legacy,json,pretty" resolves to "json pretty" for the printing format.
- "legacy,pretty" resolves to "legacy" because pretty and plain apply to the json and flat formats only.
- "flat,pretty,disable" resolves to "disable" and turns off stats.

Legacy Statistics

Cray left legacy statistics unchanged to ensure that existing scripts do not break, but does not provide formal documentation of them. To provide statistics in a legacy-compatible format, specify options in the legacy category. The legacy options are deprecated with this release and will be removed altogether in a future release.

Statistics and Caching

DVS statistics measure I/O that passes through DVS. In general, all read, aio_read, write, and aio_write operations specified by the user application result in a DVS read, aio_read, write, or aio_write operation. DVS records the total bytes reported to the application for each read or write, and the maximum file offset accessed. These are recorded regardless of whether the file system is cache-enabled.

If caching is disabled, each read or write operation is forwarded to the server(s). This results in a file system read or write on the server and a transfer of data between client and server. The accumulated totals in the statistics represent actual load on the DVS transport.

If (client) caching is enabled, the read or write operation is not forwarded to the server(s). Instead, it is diverted to a Linux routine that attempts to satisfy the read or write using the Linux file system cache.

- If the data is already in the cache, Linux completes this operation entirely in memory. The accumulated totals in the statistics represent no load on the DVS transport.
- If the data is not already in the cache, Linux issues a page read/write operation to DVS, which results in a read/write sent to the server(s). DVS statistics record the total bytes reported for this cache read/write.

The effectiveness of the cache is represented by the ratio of user read/write bytes to cache read/write bytes.

ratio > 1.0 Indicates that the cache is being used effectively, with more cache hits than misses.

ratio = 1.0 Represents something like sequential reads of a very large file: reads force continual cache fills, but the reads then consume all of the bytes in the cache before forcing a new cache fill.

ratio < 1.0 Indicates that cache is thrashing, such as would be caused by making small reads from a large number of open files, such that only a tiny fraction of each cache page is ever consumed.

It is not feasible to determine whether any specific user application read/write "hit" or "missed" the cache, without modifying the Linux kernel.

Statistics and the `mmap()` Functions

The `mmap()` functions use the same mechanisms as the Linux cache: mmap can be thought of as a named, "private" Linux cache that the application sets up. The `mmap()` functions cannot be used unless the DVS file system is cache-enabled. As a result, any attempt to read or write the file is diverted to the same Linux routine as would be used for any caching. The Linux routine determines that this is a `mmap()` file and sends a page request to DVS, then satisfies the read or write from memory. As in the case of caching, DVS statistics track the user application's reads/writes and the Linux page requests separately. As with caching, it cannot be determined whether a specific read/write "hit" or "missed" the mmap.

Because it is difficult to link a user application read/write with a corresponding page read/write, which is usually a many-to-one linkage, it is difficult to distinguish between a Linux file system cache operation and an `mmap()` cache operation.

Rates and Averaging

Performance rate measurements are simply counts that are automatically zeroed when sampled by reading the stats file. The resulting total is displayed, divided by the time since the last stats file read. Therefore it is possible to create real-time performance data with one-second resolution by reading the stats file every second. Because the DVS code normalizes the value by dividing by the time elapsed since the last sample, variations in the sampling rate should not be reflected in the averages. For example, if the actual data transfer rate is a steady 1GB/sec, a rate of 1GB/sec will be displayed every time the stats file is sampled, even if it is sampled manually at random intervals. If the actual data transfer rate is fluctuating, all of the fluctuating rates will be averaged together (unweighted) over the entire sampling window, however long that may be.

Races

To keep DVS performance high, statistics reporting is not atomic: atomic values are sampled as each line of output is rendered, while DVS is running. As a result, counts that might be expected to have an exact mathematical relationship may not. For example, if all applications are reading in fixed-size blocks, one might expect an exact relationship between iops and bytes read. However, that relationship will not generally be exact, because values are actually sampled as the statistics output is rendered, and values rendered later in time will contain new counts not found in values rendered earlier.

6.3.1 DVS Statistics Collected

The following table shows the full keys in flat format and describes whether the statistic is aggregate across all mounts (`/proc/fs/dvs/stats`), or per-mount (`/proc/fs/dvs/mounts/`*`nnn`*`/stats`), or both.

Table 2. Aggregate (AGG) and Per-mount (MNT) Statistics Collected by DVS

Key	AGG	MNT	Meaning
STATS.version	◆	◆	Current version of statistics. Any time this number changes, the format or content of the statistics have changed. Because DVS uses a self-describing format (key/value or JSON), analysis code may be able to run with new versions without code changes; however, there is no guarantee.
STATS.flags	◆	◆	Current option flags.
RQ.*req*.req.ok	◆	◆	Count of requests sent by this host that resulted in successful response. See the table *Valid req Values* for valid values for the RQ keys.
RQ.*req*.req.err	◆	◆	Count of requests sent by this host that resulted in a failed send or a failure response.
RQ.*req*.reqp.ok	◆	◆	Count of requests received by this host that resulted in successful action on this host.
RQ.*req*.reqp.err	◆	◆	Count of requests received by this host that resulted in failed action on this host.
RQ.*req*.reqp.dur.prv	◆	◆	Duration (seconds) of the most recent request received and processed by this host.
RQ.*req*.reqp.dur.max	◆	◆	Maximum duration (seconds) of any requests received and processed by this host.
OP.*fileop*.ok	◆	◆	Count of successful file operations called by Linux on this host. See the table *Valid fileop Values* for valid values for the OP keys.
OP.*fileop*.err	◆	◆	Count of failed file operations called by Linux on this host.
OP.*fileop*.dur.prv	◆	◆	Duration (seconds) of the most recent file operation on this host.
OP.*fileop*.dur.max	◆	◆	Maximum duration (seconds) of any file operations on this host.
IPC.requests.ok	◆		These have the same meanings as the RQ.*req*.req.ok and RQ.*req*.req.err keys above, except for these differences: • IPC keys show aggregated messages across all mounts (RQ keys show individual messages) • IPC keys grouped by whether messages sent synchronously, asynchronously, or as a reply message
IPC.requests.err	◆		
IPC.async_requests.ok	◆		
IPC.async_requests.err	◆		
IPC.replies.ok	◆		
IPC.replies.err	◆		
PERF.user.read.min_len		◆	Low-water mark of all nonzero-length user read operations. Includes both read() and aio_read() calls.
PERF.user.read.max_len		◆	High-water mark of all nonzero-length user read operations.
PERF.user.read.max_off		◆	High-water mark of all file byte offsets read by user calls.

Key	AGG	MNT	Meaning
PERF.user.read.total.iops		◆	Accumulated I/O operation count of all user read operations since module load.
PERF.user.read.total.bytes		◆	Accumulated byte transfer count of all user read operations since module load.
PERF.user.read.rate.iops		◆	Accumulated I/O operation count of all user read operations since the last read of stats, divided by the number of seconds since the last read of stats.
PERF.user.read.rate.bytes		◆	Accumulated byte transfer count of all user read operations since the last read of stats, divided by the number of seconds since the last read of stats.
PERF.user.write.min_len		◆	Low-water mark of all nonzero-length user write operations. Includes both write() and aio_write() calls.
PERF.user.write.max_len		◆	High-water mark of all nonzero-length user write operations.
PERF.user.write.max_off		◆	High-water mark of all file byte offsets written by user calls.
PERF.user.write.total.iops		◆	Accumulated I/O operation count of all user write operations since module load.
PERF.user.write.total.bytes		◆	Accumulated byte transfer count of all user write operations since module load.
PERF.user.write.rate.iops		◆	Accumulated I/O operation count of all user write operations since the last read of stats, divided by the number of seconds since the last read of stats.
PERF.user.write.rate.bytes		◆	Accumulated byte transfer count of all user write operations since the last read of stats, divided by the number of seconds since the last read of stats.
PERF.cache.read.min_len		◆	Low-water mark of all nonzero-length Linux cache read operations.
PERF.cache.read.max_len		◆	High-water mark of all nonzero-length Linux cache read operations.
PERF.cache.read.max_off		◆	High-water mark of all file byte offsets read by Linux cache calls.
PERF.cache.read.total.iops		◆	Accumulated I/O operation count of all Linux cache read operations since module load.
PERF.cache.read.total.bytes		◆	Accumulated byte transfer count of all Linux cache read operations since module load.
PERF.cache.read.rate.iops		◆	Accumulated I/O operation count of all Linux cache read operations since last read of stats, divided by the number of seconds since the last read of stats.
PERF.cache.read.rate.bytes		◆	Accumulated byte transfer count of all Linux cache read operations since last read of stats, divided by the number of seconds since the last read of stats.

Key	AGG	MNT	Meaning
PERF.cache.write.min_len		◆	Low-water mark of all nonzero-length Linux cache write operations.
PERF.cache.write.max_len		◆	High-water mark of all nonzero-length Linux cache write operations.
PERF.cache.write.max_off		◆	High-water mark of all file byte offsets written by Linux cache calls.
PERF.cache.write.total.iops		◆	Accumulated I/O operation count of all Linux cache write operations since module load.
PERF.cache.write.total.bytes		◆	Accumulated byte transfer count of all Linux cache write operations since module load.
PERF.cache.write.rate.iops		◆	Accumulated I/O operation count of all Linux cache write operations since the last read of stats, divided by the number of seconds since the last read of stats.
PERF.cache.write.rate.bytes		◆	Accumulated byte transfer count of all Linux cache write operations since the last read of stats, divided by the number of seconds since the last read of stats.
PERF.legacy.inodes.created		◆	Accumulated count of inodes created on this host (includes mirrored inodes that represent existing files on the server).
PERF.legacy.inodes.deleted		◆	Accumulated count of inodes deleted on this host.
PERF.files.created		◆	Accumulated count of regular files created on server file systems.
PERF.files.deleted		◆	Accumulated count of regular files deleted on server file systems.
PERF.files.open		◆	Current count of DVS files open.
PERF.symlinks.created		◆	Accumulated count of symlinks created on server file systems.
PERF.symlinks.deleted		◆	Accumulated count of symlinks deleted on server file systems.
PERF.directories.created		◆	Accumulated count of directories created on server file systems.
PERF.directories.deleted		◆	Accumulated count of directories deleted on server file systems.

DVS Messages

The following table shows the valid `req` values to be used in the RQ statistics, as described in the table *Aggregate (AGG) and Per-mount (MNT) Statistics Collected by DVS* on page 77. They represent messages passed between the DVS client and server.

Table 3. Valid `req` Values

req	**Meaning**
RQ_CLOSE	Close open file on the server.

req	Meaning
RQ_CREATE	Create a regular file on the server.
RQ_FASYNC	
RQ_FLUSH	
RQ_FSYNC	
RQ_GETATTR	Get file attributes on the server.
RQ_GETEOI	
RQ_GETXATTR	Get file extended attributes on the server.
RQ_IOCTL	Perform a general ioctl on the server.
RQ_LINK	
RQ_LISTXATTR	List extended attributes on the server.
RQ_LOCK	
RQ_LOOKUP	
RQ_MKDIR	Create a directory on the server.
RQ_MKNOD	Create a special device file on the server.
RQ_OPEN	Open a file on the server.
RQ_PARALLEL_READ	
RQ_PARALLEL_WRITE	
RQ_PERMISSION	
RQ_READDIR	
RQ_READLINK	Read the contents of a symlink on the server.
RQ_READPAGE_ASYNC	
RQ_READPAGE_DATA	
RQ_READPAGES_RP	
RQ_READPAGES_RQ	
RQ_REMOVEXATTR	Remove extended attributes on the server.
RQ_RENAME	Rename a file on the server.
RQ_RMDIR	Remove a directory on the server.
RQ_RO_CACHE_DISABLE	
RQ_SETATTR	
RQ_SETXATTR	
RQ_STATFS	Perform a file stat on the server.

req	Meaning
RQ_SYMLINK	Create a symlink on the server.
RQ_SYNC_UPDATE	
RQ_TRUNCATE	Truncate an open file on the server.
RQ_UNLINK	Unlink (delete) a file object on the server.
RQ_VERIFYFS	
RQ_WRITEPAGES_RP	
RQ_WRITEPAGES_RQ	

Virtual File System Operations

The following table shows the valid *fileop* values to be used in IPC statistics, as described in the table *Aggregate (AGG) and Per-mount (MNT) Statistics Collected by DVS* on page 77. They are virtual file system (VFS) operations requested by the Linux VFS framework, and they represent file system actions to be performed by the kernel on behalf of a user-space application. For example, a call to `open()` in a user application results in a call to the open handler (*fileop* = open in the following table), requesting that the DVS file system open a file. VFS operations may operate locally (for example, out of local cache memory), or they may result in one or more request messages sent to the server. The d_, f_, and l_ prefixes are vestigial references to the file system object on which the operation is performed. They will be removed in a future release.

Table 4. Valid fileop Values

fileop	Meaning
aio_read	Perform a vectorized read from an open file.
aio_write	Perform a vectorized write to an open file.
d_create	Create a new regular file in a directory.
d_getattr	Get attributes for a directory.
d_getxattr	Get extended attributes for a directory.
d_link	Create a new dentry (link) for an inode.
d_listxattr	List extended attributes for a directory.
d_lookup	Find the dentry corresponding to an inode.
d_mkdir	Create a subdirectory within a directory.
d_mknod	Create a special device file within a directory.
d_permission	Check access permissions for an object in a directory for current user.
d_removexattr	Remove extended attributes for a directory.
d_rename	Rename an object within a directory.
d_revalidate	Revalidate an object within a directory.

fileop	Meaning
d_rmdir	Remove a subdirectory within a directory.
d_setattr	Set attributes a directory.
d_setxattr	Set extended attributes for a directory.
d_symlink	Create a symlink within a directory.
d_truncate	OBSOLETE
d_unlink	Delete an object within a directory.
direct_io	Perform vectorized I/O bypassing Linux cache.
evict_inode	
f_create	NOT USED
f_getattr	
f_getxattr	
f_link	NOT USED
f_listxattr	
f_mkdir	NOT USED
f_mknod	NOT USED
f_permission	NOT USED
f_removexattr	
f_rename	NOT USED
f_rmdir	NOT USED
f_setattr	
f_setxattr	
f_symlink	NOT USED
f_truncate	OBSOLETE
f_unlink	NOT USED
fasync	Control asynchronous I/O notifications.
flock	BSD file locking not supported.
flush	
fsync	Perform fsync on an open file.
l_follow_link	Follow a symlink.
l_getattr	
l_put_link	Free kernel memory after following a symlink.

fileop	Meaning
l_readlink	Read the contents of a symlink.
l_setattr	
llseek	Seek to a position in an open file.
lock	Control discretionary file locks.
mmap	
open	Open a file system object.
put_super	Free kernel memory for superblocks.
read	Perform sequential read from an open file.
readdir	
readpage	
readpages	
release	
show_options	
statfs	Get status of a file path.
unlocked_ioctl	
write	Perform sequential write to an open file.
write_begin	Prefetch for a cached write.
write_end	Perform a cached write.
writepage	
writepages	
write_super	OBSOLETE

6.4 DVS Can Log Requests Sent to Servers

To aid in debugging problems that may arise, DVS request logging is enabled by default. To reduce the overhead to each client request, DVS logs only those requests that take more than a certain number of seconds to complete. For each request that exceeds the threshold, DVS writes a single line to `/proc/fs/dvs/request_log`. That minimum threshold can be changed (default is 15 seconds) to see more or fewer requests in the log.

There are two ways to make changes to DVS request logging. When the system is running, an administrator can ssh to a node and make changes dynamically by writing values to certain `/proc` files. Alternatively, to set the behavior from the moment the DVS kernel modules load, administrators can set kernel module parameters prior to boot.

Control DVS Request Logging and the Log Buffer Size

To disable, enable, or reset DVS request logging and to change the buffer size, use these kernel module parameters. Because these parameters cannot be set using the configurator, use the procedure *Change Kernel Module Parameters Prior to Boot using Modprobe.d Files and Simple Sync* on page 32 and the following information, both from the topic *Configure DVS using Modprobe or Proc Files* on page 30.

dvs_request_log_enabled

Logs each DVS request sent to servers.

- Default value: 1 (enabled)
- To view read-only:
  cat `/sys/module/dvsproc/parameters/dvs_request_log_enabled`
- To change prior to boot, add these lines to
  <simple sync path>`/etc/modprobe.d/dvs-local.conf`:

```
# Disable DVS request log
options dvsproc dvs_request_log_enabled=0

# Enable DVS request log
options dvsproc dvs_request_log_enabled=1
```

- To change dynamically:

```
hostname# echo 0 > /proc/fs/dvs/request_log
hostname# echo 1 > /proc/fs/dvs/request_log
hostname# echo 2 > /proc/fs/dvs/request_log
```

  The value `2` resets the log.

dvs_request_log_size_kb

Size (KB) of the request log buffer.

- Default value: 16384 KB (16384 * 1024 bytes)
- To view read-only:
  cat `/sys/module/dvsproc/parameters/dvs_request_log_size_kb`
- To change prior to boot, add these lines to
  <simple sync path>`/etc/modprobe.d/dvs-local.conf`:

```
# Set size (in kb) of the request log buffer
options dvsproc dvs_request_log_size_kb=17000
```

- To change dynamically:

```
hostname# echo 17000 > /proc/fs/dvs/request_log_size_kb
```

  To determine the current buffer size, cat the file. For example:

```
hostname# cat /proc/fs/dvs/request_log_size_kb
16384
```

dvs_request_log_min_time_secs

Defines a threshold of time for data to be logged to the `request_log`.

- Default value: 15 seconds
- To view read-only:
  cat `/sys/module/dvsproc/parameters/dvs_request_log_time_min_secs`
- To change prior to boot, add these lines to
  <simple sync path>`/etc/modprobe.d/dvs-local.conf`:

```
# Set threshold (in seconds) for time a DVS request
# takes before logged. Requests taking fewer seconds
# will not be logged.
options dvsproc dvs_request_log_time_min_secs
```

- To change dynamically:

```
hostname# echo 15 > /proc/fs/dvs/request_log_time_min_secs
```

Example DVS Request Log

Here is an example DVS request log.

```
hostname# cat /proc/fs/dvs/request_log
2016-12-7 21:59:54-UTC: pid=34548 cmd=pkg-config path=/var/opt/cray/imps/image_roots/pe_compute_cle_6.0-latest_sles_12/
opt/cray/pe/pmi/5.0.11-20.0000.c0dc009.413.0.ari/lib64/pkgconfig type=dir req=RQ_OPEN count=1 node=c0-0c2s15n0
time=0.000
2016-12-7 21:59:54-UTC: pid=34548 cmd=pkg-config path=/var/opt/cray/imps/image_roots/pe_compute_cle_6.0-latest_sles_12/
opt/cray/pe/pmi/5.0.11-20.0000.c0dc009.413.0.ari/lib64/pkgconfig type=dir req=RQ_READDIR count=1 node=c0-0c2s15n0
time=0.000
2016-12-7 21:59:54-UTC: pid=34548 cmd=pkg-config path=/var/opt/cray/imps/image_roots/pe_compute_cle_6.0-latest_sles_12/
opt/cray/pe/pmi/5.0.11-20.0000.c0dc009.413.0.ari/lib64/pkgconfig type=dir req=RQ_READDIR count=1 node=c0-0c2s15n0
time=0.004
2016-12-7 21:59:54-UTC: pid=34548 cmd=pkg-config path=/var/opt/cray/imps/image_roots/pe_compute_cle_6.0-latest_sles_12/
opt/cray/pe/pmi/5.0.11-20.0000.c0dc009.413.0.ari/lib64/pkgconfig type=dir req=RQ_CLOSE count=1 node=c0-0c2s15n0
time=0.000
2016-12-7 21:59:54-UTC: pid=34548 cmd=pkg-config path=/var/opt/cray/imps/image_roots/pe_compute_cle_6.0-latest_sles_12/
opt/cray/pe/craype/2.5.10.1/pkg-config type=dir req=RQ_OPEN count=1 node=c0-0c2s15n0 time=0.000
2016-12-7 21:59:54-UTC: pid=34548 cmd=pkg-config path=/var/opt/cray/imps/image_roots/pe_compute_cle_6.0-latest_sles_12/
opt/cray/pe/craype/2.5.10.1/pkg-config type=dir req=RQ_READDIR count=1 node=c0-0c2s15n0 time=0.000
2016-12-7 21:59:54-UTC: pid=34548 cmd=pkg-config path=/var/opt/cray/imps/image_roots/pe_compute_cle_6.0-latest_sles_12/
opt/cray/pe/craype/2.5.10.1/pkg-config type=dir req=RQ_READDIR count=1 node=c0-0c2s15n0 time=0.000
2016-12-7 21:59:54-UTC: pid=34548 cmd=pkg-config path=/var/opt/cray/imps/image_roots/pe_compute_cle_6.0-latest_sles_12/
opt/cray/pe/craype/2.5.10.1/pkg-config type=dir req=RQ_CLOSE count=1 node=c0-0c2s15n0 time=0.000
2016-12-7 21:59:54-UTC: pid=34548 cmd=pkg-config path=/var/opt/cray/imps/image_roots/pe_compute_cle_6.0-latest_sles_12/
opt/cray/pe/iobuf/2.0.7/lib/pkgconfig type=dir req=RQ_OPEN count=1 node=c0-0c2s15n0 time=0.000
2016-12-7 21:59:54-UTC: pid=34548 cmd=pkg-config path=/var/opt/cray/imps/image_roots/pe_compute_cle_6.0-latest_sles_12/
opt/cray/pe/iobuf/2.0.7/lib/pkgconfig type=dir req=RQ_READDIR count=1 node=c0-0c2s15n0 time=0.000
[...]
```

Each line in the log contains this information:

date/time	Date and time (in UTC) of request.
pid	User process ID of the process initiating the request.
cmd	Command being executed by the user process ID.
path	Path on the server node that is being referenced.
type	Type of path (file, directory, link, etc.).
req	Type of DVS request.
count	Number of servers the request was sent to. Some DVS client operations send a request to a single server, and others send a request to multiple servers. The former will log a line with count=1, and the latter will log a line with count=N, where N is the number of servers the request was sent to. When the request has been sent to multiple servers, the single line in the log is a summary of the requests.

node Node is set to the node that the DVS request was sent to. If multiple nodes were targeted (i.e., the count field is > 1), then the value displayed is `[multiple]`.

time Time is set to the amount of time in seconds it took for the request to be sent to the server, execute, and for the reply to be received by the client. Granularity is milliseconds, so 0.000 is displayed for requests that take less than a millisecond.

6.5 DVS Can Log Details About File System Calls

DVS can log details about the calls that the DVS server node makes into the underlying file system. The log files are created in `/proc/fs/dvs/fs_log` and can provide the administrator with useful information, such as:

- The pid and name of the thread making the file system call
- The operation being executed (open, close, read, write, getattr, ...)
- The uid of the thread making the file system call, which DVS sets to the uid of the process on the DVS client
- The APID of the process that made the request on the client node (if ALPS was used as the job launcher)
- The c-name of the client node that sent the request to the server
- The path corresponding to the operation, if any

The file system operations logged are a larger set than the list of DVS requests. This is because it is sometimes necessary to make multiple and different file system calls to complete a single DVS request. For these secondary file system operations, the operation logged contains `op[func]` where `op` is the file system operation performed and `func` is the DVS function that contained the call. This lets the reader distinguish the primary from the secondary file system calls.

```
hostname# cat /proc/fs/dvs/fs_log_enable
2017-5-1 12:47:20-UTC: pid=31764 cmd=DVS-IPC_msg op=open uid=13299 apid=491654 node=c0-0c0s11n0
time=0.001 path=/cray
```

Note: For some code paths, there is only access to the "basename" portion of the path.

Control DVS Log Details and Adjust Buffer Content

The following kernel module parameters are available to create log files and adjust buffer content if desired. Because these parameters cannot be set using the configurator, use the procedure *Change Kernel Module Parameters Prior to Boot using Modprobe.d Files and Simple Sync* on page 32 and the following information, both from the topic *Configure DVS using Modprobe or Proc Files* on page 30.

dvs_fs_log_enabled

Logs information on I/O operations made from DVS to the underlying file system on DVS server nodes.

- Default value: 1 (enabled)
- To view read-only:
  cat `/sys/module/dvsproc/parameters/dvs_fs_log_enabled`

- To change prior to boot, add these lines to *<simple sync path>*/etc/modprobe.d/dvs-local.conf:

```
# Disable DVS fs log
options dvsproc dvs_fs_log_enabled=0

# Enable DVS fs log
options dvsproc dvs_fs_log_enabled=1
```

- To change dynamically:

```
hostname# echo 0 > /proc/fs/dvs/fs_log
hostname# echo 1 > /proc/fs/dvs/fs_log
hostname# echo 2 > /proc/fs/dvs/fs_log
```

dvs_fs_log_size_kb

Size (KB) of the log buffer.

- Default value: 32768 KB (32768 * 1024 bytes)
- To view read-only:
  cat /sys/module/dvsproc/parameters/dvs_fs_log_size_kb
- To change prior to boot, add these lines to *<simple sync path>*/etc/modprobe.d/dvs-local.conf:

```
# Set size (in kb) of the fs log buffer
options dvsproc dvs_fs_log_size_kb=17000
```

- To change dynamically:

```
hostname# echo 17000 > /proc/fs/dvs/fs_log_size_kb
```

To determine the current buffer size, cat the file. For example:

```
hostname# cat /proc/fs/dvs/fs_log_size_kb
32768
```

dvs_fs_log_min_time_secs

Defines a threshold of time for data to be logged to the fs_log.

- Default value: 15 seconds
- To view read-only:
  cat /sys/module/dvsproc/parameters/dvs_fs_log_time_min_secs
- To change prior to boot, add these lines to *<simple sync path>*/etc/modprobe.d/dvs-local.conf:

```
# Set threshold (in seconds) for time a DVS requests
# takes before logged. Requests taking fewer seconds
# will not be logged.
options dvsproc dvs_fs_log_time_min_secs
```

- To change dynamically:

```
hostname# echo 15 > /proc/fs/dvs/fs_log_time_min_secs
```

6.6 DVS Lists Outstanding Client Requests

DVS provides a list of outstanding requests on client nodes in `/proc/fs/dvs/ipc/requests`, which lists the DVS server node, the request, the DVS file system path, `uid`, time that the request has been waiting for a response, and the associated `apid`. If the request is from a process that was not spawned through `aprun`, the request `apid` is 0. An example output of the file looks like:

```
% cat /proc/fs/dvs/ipc/requests
server: c0-0c0s0n0 request: RQ_LOOKUP path: /cray_home user: 12795 time: 0.000 sec
apid: 3871
```

The file appears on DVS servers but returns an error when a user tries to access it.

6.6.1 DVS Provides a Plugin for Node Health Checker

The Node Health Checker (NHC) in CLE performs specified tests to determine if compute nodes allocated to an application are healthy enough to support running subsequent applications. These tests are enabled by various plugin scripts. These scripts run according to the node health configuration, typically after an abnormal termination of a user application, as evidenced by a crash or a non-zero exit code.

Cray DVS provides `dvs_request`, a plugin script that outputs information on the oldest outstanding client request in the `/proc/fs/dvs/ipc/requests` file, including the `cname` of the DVS server processing the request. This plugin is included by default on systems that have an initial install (as opposed to an update) of CLE 5.2.UP00 or later.

For more information on configuring and using NHC, see "Configure the Node Health Checker (NHC)" in *XC System Administration Guide*.

7 DVS Troubleshooting

Here are some issues that could arise when using DVS.

DVS Does Not Start after Data Store Moved to External Lustre File System

If DVS fails after the Cray system's data store is moved to a shared external Lustre file system, verify that DVS has the correct `lnd_name`.

`lnd_name` uniquely identifies the LNet network that DVS will use. DVS communicates it to the LNet service when DVS is being initialized. It must match the `cray_lnet.settings.local_lnet.data.lnet_name` value set in the `cray_lnet` service for DVS to boot properly. To find that value, search the CLE config set (this example searches in config set `p0` and finds lnet_name = gni4):

```
smw# cfgset search --term lnet_name \
--state all --service cray_lnet p0
# 1 match for 'lnet_name' from cray_lnet_config.yaml
#---------------------------------------------------
cray_lnet.settings.local_lnet.data.lnet_name: gni4
```

If `lnd_name` does not match `lnet_name` from the `cray_lnet` service, change it. Because `lnd_name` is a kernel module parameter that cannot be set using the configurator, add these lines to *<simple sync path>*`/etc/modprobe.d/dvs-local.conf`, substituting for *gnix* the value found from the config set search:

```
# Set identifier of LNet network DVS will use
options dvsipc_lnet lnd_name=gnix
```

For information about what *<simple sync path>* should be, see the procedure *Change Kernel Module Parameters Prior to Boot using Modprobe.d Files and Simple Sync* on page 32, which is found in *Configure DVS using Modprobe or Proc Files* on page 30.

ALPS Kills a Process to Avoid Potential Data Loss

DVS forwards file system writes from clients to servers. The data written on the DVS server may reside in the server's page cache for an indeterminate time before the Linux kernel writes the data to backing store. If the server crashes before the data is written to backing store, this data is lost. To prevent silent data loss, DVS kills the processes on the clients that wrote the data. If the Application Level Placement Scheduler (ALPS) was used to launch the application, the system displays the following message to the terminal before `aprun` exits: "DVS server failure detected: killing process to avoid potential data loss."

To avoid this error message, do one of the following:

- Add the `datasync` option to the `options` setting of the `client_mount` setting for that client mount in the `cray_dvs` service, or use the `DVS_DATASYNC` user environment variable. This avoids the error message

because each write operation is followed by `fsync` before it is considered complete. However, be aware that this also exacts a substantial performance penalty.

- Add the `nokillprocess` option to the `options` field of the `client_mount` setting for that client mount in the `cray_dvs` service or set the `DVS_KILLPROCESS` user environment variable to `off`. When a server fails, processes that have written data to the server are not killed. If a process continues to perform operations with an open file descriptor that had been used to write data to the server, the operations fail (with `errno` set to `EHOSTDOWN`). A new open of the file is allowed, and subsequent operations with the corresponding file descriptor function normally.

Application Hangs as a Result of NFS File Locking

Applications may hang when NFS file systems are projected through DVS and file locking is used. To avoid this issue, add the `nolock` option to the `options` field of the `client_mount` setting for the NFS client mount in the `cray_dvs` service. See the `nfs(5)` man page for more information on the `nolock` option.

DVS Ignores User Environment Variables

If the `nouserenv` option has not been specified when configuring a DVS client mount, and a DVS user environment variable that was set does not override the associated DVS mount option, it appears as if DVS is ignoring user environment variables. This can be caused by the addition of a large number of user environment variables. Due to the nature of Linux, if a user adds a large number of user environment variables (large enough that the kernel needs to store that information somewhere other than the usual location), DVS may not be able to find and apply those user environment variables, producing unexpected results.

To define a large number of user environment variables, Cray recommends that users include those definitions in the user's shell so that they are available at startup and stored where DVS can always locate them.

8 DVS Caveats

Read the following caveats and limitations to avoid common pitfalls when configuring and using Cray DVS.

When Configuring DVS

GPFS (Spectrum Scale) `blksize`

When projecting a general parallel file system (Spectrum Scale), the client mount option `blksize` must match or be a multiple of the Spectrum Scale file system blocksize. When projecting multiple Spectrum Scale file systems that have different block sizes, configure a separate DVS client mount for each file system.

For example, projecting two Spectrum Scale file systems, one with a 64 kilobyte (KB) block size, and another with a 1024KB block size, the client mount settings might have these values:

```
1) 'gpfs1'
    a) mount_point: /gpfs1
    b) spath: /dvs1
    c) servers:
           c0-0c0s0n2
    d) clients: (none)
    e) loadbalance: False
    f) attrcache_timeout: 14400
    g) readonly: True
    h) options: blksize=65536

2) 'gpfs2'
    a) mount_point: /gpfs2
    b) spath: /dvs2
    c) servers:
           c0-0c0s0n3
    d) clients: (none)
    e) loadbalance: False
    f) attrcache_timeout: 14400
    g) readonly: True
    h) options: blksize=1048576
```

expanded file system support

Setting up and mounting target file systems on Cray service nodes is the sole responsibility of the customer or an agent of the customer. Cray Custom Engineering is available to provide a tailored file system solution. Please contact a Cray service representative for more information.

When Using DVS Programmatically

client consistency	DVS supports close-to-open consistency, which means that files on client and server are consistent at `open()` and `close()`. However, while a file is open, DVS does not guarantee that the file on the client and the file on the server are consistent.
`flock()` not supported	DVS does not support `flock()` system calls and will return an error. DVS will set `errno` to `ENOTSUPP` when a `flock()` call is attempted for a DVS-projected file system. DVS supports file locking with `fcntl()`. For more information, see the `fcntl(2)` man page.

9 Supplemental Information

With the SMW 8.0 / CLE 6.0 release, Cray has changed the way software is installed, configured, and managed on XC Series systems. The new Cray Management System (CMS) leverages standard Linux and common open source tools (e.g., zypper/yum for RPMs, Ansible, YAML/JSON configuration data), and it centralizes configuration, keeping it separate from software images until it is applied to nodes at boot time or whenever cray-ansible is run.

The core elements of this new management system are:

IMPS Image Management and Provisioning System (IMPS) is responsible for creating and distributing repository content and for prescriptive image creation. Note that although filepaths for configuration data and tools include `imps`, this is an artifact of an early implementation that grouped both image and configuration management under IMPS. IMPS is now image management only.

CMF Configuration Management Framework (CMF) comprises the configuration data (stored in config sets on the SMW), tools to manage and distribute that data (e.g., the configurator and the IMPS Distribution System (IDS)), and software to apply the configuration data to the running image (Ansible plays).

NIMS Node Image Mapping Service (NIMS) is responsible for keeping track of which images get booted on which nodes, what additional kernel parameters to pass to nodes at boot time, and which load file to use within a boot image.

The following topics have been added to this DVS guide to provide additional information that may help in understanding the DVS-specific procedures that comprise the bulk of this guide.

- *Cray XC System Configuration* on page 93
- *About the Configurator* on page 95
- *Config Set Create/Update Process* on page 96
- *About Simple Sync* on page 100
- *About Node Groups* on page 105
- *About Config Set Caching* on page 109

9.1 Cray XC System Configuration

To configure Cray XC systems and manage configuration content, system administrators use the Cray configuration management framework (CMF). The CMF comprises configuration data, the tools to manage and distribute that data, and software to apply the configuration data to the running image at boot time. Its major components include configuration service packages, config sets, the IMPS distribution service (IDS), the configurator, cray-ansible, and Ansible.

Configuration Starts with Configuration Service Packages

Configuration content (data and software) is installed as configuration service packages on the management node of Cray XC systems (in `/opt/cray/imps_config/<service package>/default/configurator` by default). Each service package delivers configuration content for one or more system services. The contents of each service package reside in the following subdirectories:

ansible	Drop zone for Cray-provided Ansible play content.
callbacks	Pre- and post-configuration scripts.
dist	Drop zone for other Cray-provided content, such as static files required for the configuration of a service.
template	Configuration templates that define the configuration settings to be set and provide some default values. These templates are never modified by administrators or other users.

Configuration service packages are installed for system upgrades and updates as well as for initial installation.

Configuration Information is Stored in Config Sets

Administrators use the `cfgset` command to manage configuration information. It takes configuration content delivered in service packages and invokes the configurator tool to combine that content with site-specific configuration content gathered from administrators either interactively or through bulk import. The results are used by `cfgset` to create a configuration set or config set. A config set is a central repository that stores all configuration information necessary to operate the system. Config sets reside on the management node (e.g., the SMW) in `/var/opt/cray/imps/config/sets` by default. The contents of each config set reside in the following subdirectories:

ansible	Drop zone for local site-provided Ansible play content to be distributed with the config set. When the config set is created, `cfgset` copies Ansible content from service packages to this location. Whenever the config set is updated, `cfgset` copies Ansible content from service packages again, overwriting the previous service-package Ansible content and leaving the site-provided content unchanged.
changelog	YAML change logs from previous sessions with the configurator.
config	Configuration templates containing configuration information. When the config set is created, the configurator copies service package templates to this location. Administrators can modify the content of these templates using `cfgset` and the configurator. Whenever the config set is updated, the configurator merges service package templates with the templates in this location.
dist	Drop zone for other site-provided content, such as static files required for the configuration of a service. When the config set is created, `cfgset` copies dist content from service packages to this location. Whenever the config set is updated, `cfgset` copies dist content from service packages again, overwriting the previous service-package dist content and leaving the site-provided content unchanged.
files	Files necessary for system configuration that are generated by configuration callback scripts or manually and distributed with the config set (e.g., `/etc/hosts`).
worksheets	Configuration worksheets generated by the configurator using data stored in the configuration templates in the `config` subdirectory of the config set. Administrators copy these worksheets to a location outside the config set, edit them with site-specific configuration data, and then import them to create a new config set or update an existing one.

An administrator may create multiple config sets to support partitions or alternate configurations. Typically a config set of type `cle` is created for each partition to store partition- and CLE-specific content, and another config set of type `global` is created to store management node and global configuration data.

IDS Distributes Config Sets to Nodes

IDS, a read-only network share of content from the management node to the rest of the system, distributes config sets to every node in the system. All config sets are shared throughout the system, but only one `cle` config set is active on a given node at a time (in addition to an active `global` config set, which is applied to the entire system). Currently, IDS leverages the 9P network file system and the Linux automounter facility as its distribution mechanism; however, the content and use of the config sets is independent of the distribution mechanism.

Ansible Plays Apply Configuration during System Boot

Prior to booting the system, each node will have an image, the `global` config set, and the `cle` config set. When the system boots, each node boots an unconfigured software image. Then Ansible plays, which can be located in both the image and the config set (config set is the preferred location for site-supplied Ansible plays), apply configuration to that image, bringing up the services pertinent to each node.

Administrators Configure/Reconfigure the System on an Ongoing Basis

Configuration happens at times other than initial installation. New configuration service packages can be installed during system upgrades and updates, sites can decide to enable a new service or change the configuration of an existing service, and so forth. In all of these scenarios, an administrator uses the `cfgset` command to manage config sets and the `cray-ansible` script to apply any configuration changes. The `cfgset` command and its associated subcommands and options enable administrators to perform a variety of operations on config sets in addition to create and update, such as search, diff, list, show, validate, push, and remove. See the `cfgset` man page for a description of its subcommands and options and some examples of each.

9.2 About the Configurator

The configurator plays a major role in Cray XC system configuration. The configurator gathers configuration data from several sources (including the user, with helpful prompts and default values), merges and validates it, and stores it in a central location on the management node, where it is used during boot to configure the entire system. The configurator is invoked by the `cfgset` command to:

- handle all configuration template and worksheet operations
- perform steps 4, 5, and 6 of the *Config Set Create/Update Process*, including providing a user interface to gather and modify configuration data interactively or through the import of configuration worksheets

The configurator is invoked with the `cfgset` subcommands `create` (except when the `--clone` option used) and `update`. It is invoked also with the `search` subcommand, because that involves searching data stored in the configuration templates, but no changes are made to the config set using `search`. The options selected for the `create` and `update` subcommands determine the mode in which the configurator is run (with or without user interaction), which settings can be viewed and set by a user, and whether callback scripts are run before and after the configurator session. The configurator is not involved when the remaining `cfgset` subcommands are used: `diff`, `list`, `push`, `remove`, `show`, and `validate`. See the `cfgset` man page for a description of its subcommands and options and some examples of each, or use `cfgset` *`SUBCOMMAND`* `-h` to see information about just one of the subcommands.

Choose How to Interact with the Configurator: Modes

The `mode` option of the `cfgset` command determines how the configurator interacts with a user. Mode can be specified only with subcommands `create` and `update`.

`--mode` | `-m` Possible values: `auto` (default), `interactive`, `prepare`

`auto` The configurator searches through all available configuration templates in the config set and automatically presents all configuration settings that meet state and level filtering criteria. It presents the configuration settings in a certain order (taking into account dependencies among services) one at a time until all have been presented to the user, and then it automatically ends the session and saves the config set.

`interactive` The configurator searches through templates as with auto mode, but in interactive mode, it presents a menu of all available services (or a menu of all available settings, when a service has been selected) that meet state and level filtering criteria. This mode enables the user to navigate through the services and settings to view and modify the settings as needed. The configuration session ends when the user exits the session. The user chooses whether to save any changes to the config set upon exit.

`prepare` The configurator prepares configuration worksheets, one for each service. Each worksheet contains all configuration settings (unfiltered) for that service, and the worksheet can be edited offline and then imported later to create or update a config set. In this mode, the configurator does not open an interactive session with the user.

Choose What to See with the Configurator: Filters

Two `cfgset` command options act as filters to determine which settings are available to view and set or update. These options can be specified only with subcommands `create`, `update`, and `search`.

`--state` | `-S` Possible values: `unset` (default), `set`, `all`

`--level` | `-l` Possible values: `required`, `basic` (default), `advanced`

`required` Settings that must be set or the system will not function. The config set will not validate if any required settings are skipped (i.e., left unset). Specify level `required` in a `cfgset` command to filter for required settings only.

`basic` Settings that are likely to be used by most sites. If a `basic` setting is left unset, the template-provided default is used. Specify level `basic` in a `cfgset` command to filter for both basic and required settings.

`advanced` Settings that are likely to be used only by advanced users to tune a service. If an `advanced` setting is left unset, the template-provided default is used. Specify level `advanced` in a `cfgset` command to filter for all settings: advanced, basic, and required.

9.3 Config Set Create/Update Process

Config sets are created and updated using the `cfgset` command with the `create` and `update` subcommands, respectively. Invoking `cfgset` with one of those subcommands initiates the following process, which defines how

configuration content is discovered from service packages installed on the management node and used, along with site-supplied content, to create or update a config set.

1. `cfgset` searches for service packages in `/opt/cray/imps_config`.
2. `cfgset` copies to the config set (for `create`) or overwrites in the config set (for `update`) ansible and dist content from each service package. Note that it is only content from service packages that is overwritten; content placed in those directories manually is unchanged.

   NOTE: Manual changes to service package content in this directory will be overwritten!
3. `cfgset` runs pre-configuration callback scripts from each service package. Scripts act on the config set to create content necessary for system configuration, which they place into the `files` subdirectory of the config set.
4. `cfgset` invokes the configurator to do steps 4 through 6.

   Configurator finds configuration templates from each service package that match the config set type, and then copies them into the config set (for `create`) or merges them with the templates already in the config set (for `update`).
5. Configurator takes *one* of these actions to further modify config set template data, depending on the command-line options used:

interacts with user	Initiates an interactive session with the user and modifies config set template data based on the values supplied by the user. Occurs when `--mode interactive` option used or no mode option used, which defaults to `auto` mode.
does not interact with user	Does not initiate an interactive session and does no further modification to config set template data beyond the copy/merge of service package data already done in step 4. Occurs when `--mode prepare` option used. Note that although this action is associated with preparing worksheets, all three actions result in worksheets being written in step 6.
imports worksheets	Imports configuration worksheets and modifies config set template data based on the values in each service worksheet. Occurs when `--worksheet-path` *`FILEPATH`* option used.
6. Configurator writes configuration template data, configuration worksheets, and a changelog to the config set. Note that the configurator never modifies the configuration templates in service packages, which are found in `/opt/cray/imps_config/`*`SERVICE PACKAGE`* for each service package.
7. `cfgset` runs post-configuration callback scripts from each service package.
8. `cfgset` autosaves the config set to a time-stamped clone.

The following three figures illustrate how this eight-step process is used to create a CLE config set. They differ in how configuration data in a config set is further modified in step 5, corresponding to the three different actions: interacting with the user (modification through user interaction), not interacting with the user (no further modification), and importing worksheets (modification through bulk import of configuration worksheets). Black lines indicate `cfgset` actions, and red lines indicate actions taken by the configurator when invoked by `cfgset`.

This first figure shows how the configurator creates config set templates (in the `config` subdirectory) from service package templates in step 4, enables the user to enter new or modify existing configuration data in step 5, and then saves the new/modified data to the config set templates and worksheets in step 6.

Figure 8. Process to Create a Config Set Interactively

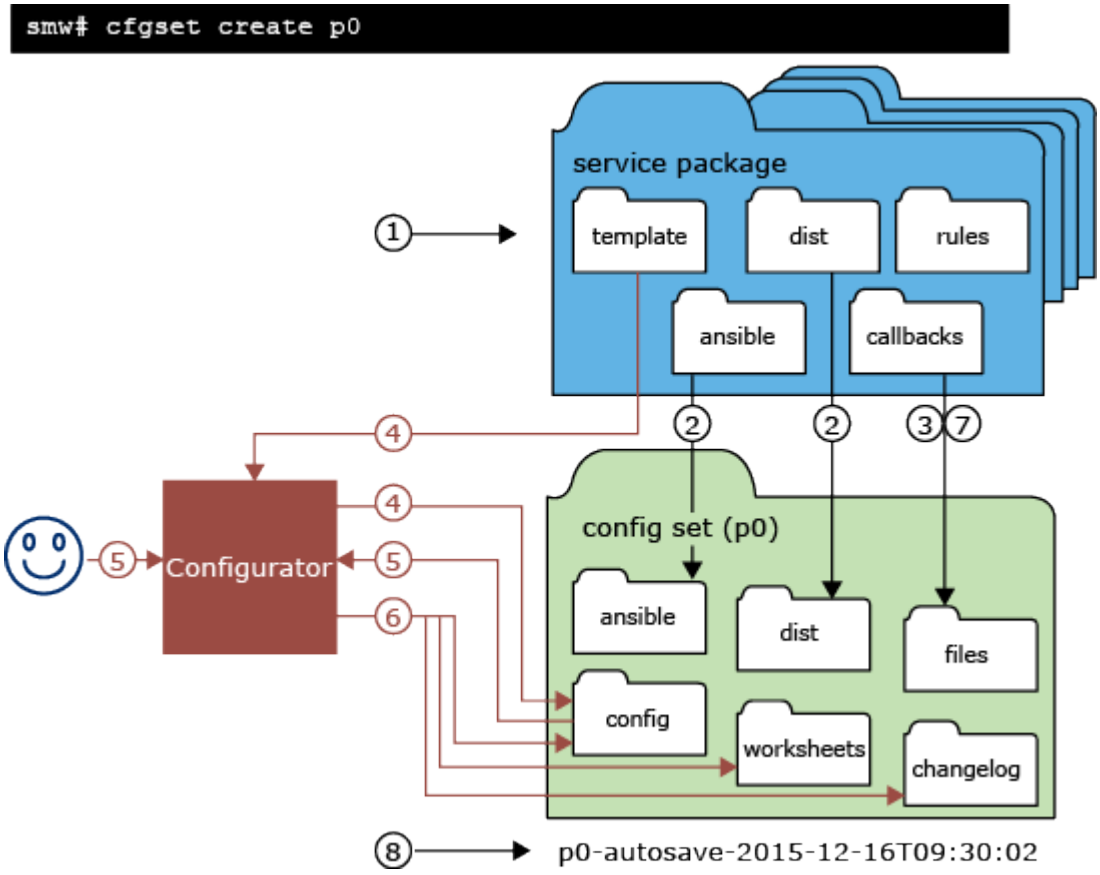
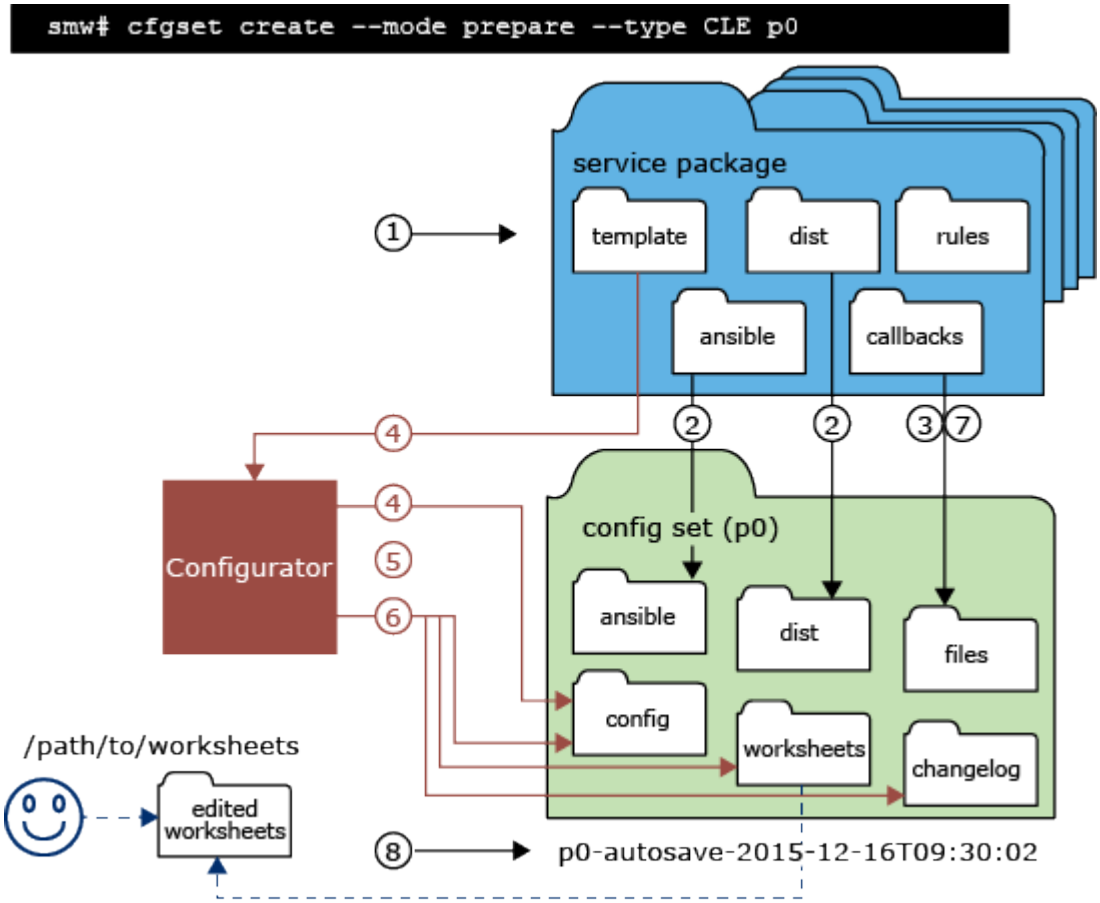
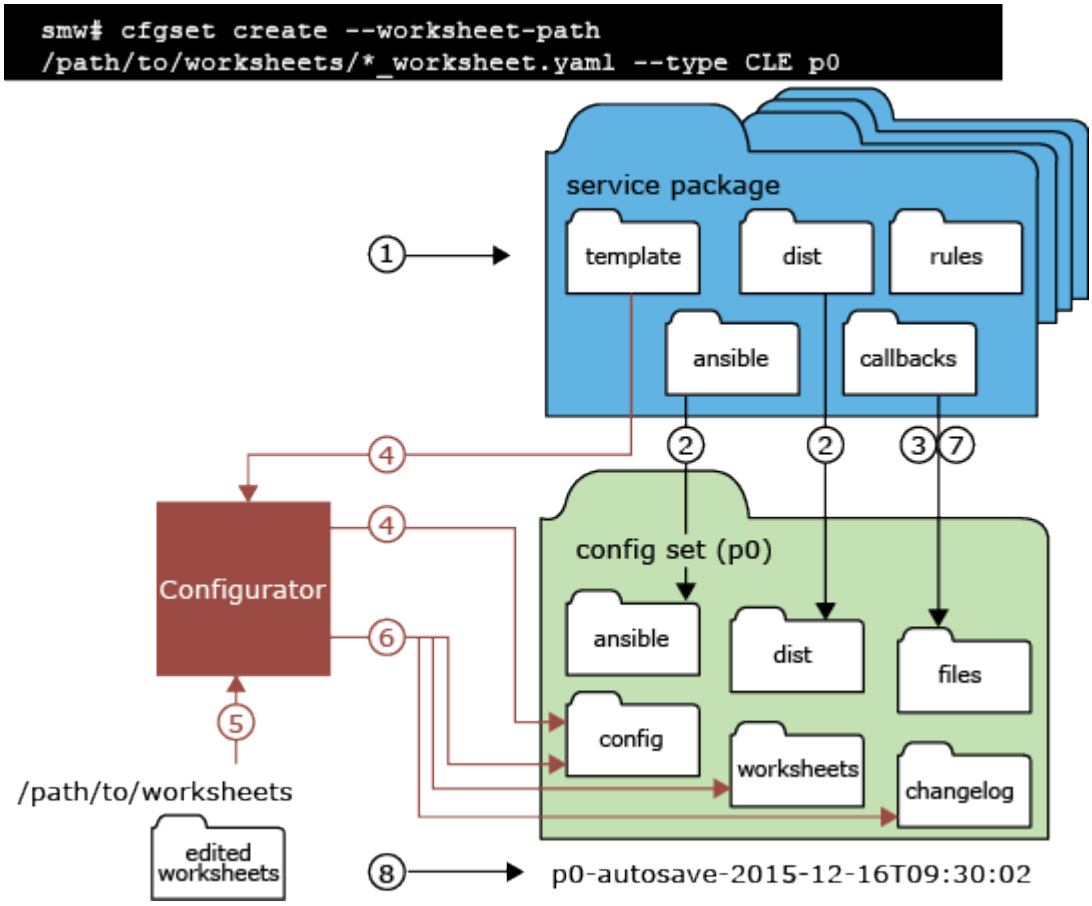


Figure 9. Process to Create a Config Set and Prepare Worksheets



The prepare-mode figure shows how the configurator creates config set templates from service package templates in step 4, does nothing to that configuration data in step 5, and then saves the data from step 4 to config set templates and worksheets in step 6. The blue dashed line indicates an action taken by the user after `cfgset` has completed the create/update process to prepare worksheets. The user (usually an installer or system administrator) copies the worksheets prepared by the configurator to a location outside the config set and edits them (or has other site staff edit them) with site-specific configuration values. It is these edited worksheets that are used when creating (or updating) a config set from worksheets (shown in worksheets figure).

Figure 10. Process to Create a Config Set from Worksheets



The worksheets figure shows how the configurator creates config set templates from service package templates in step 4, imports new or modified configuration data from worksheets in step 5, and then saves the new/modified data to the config set templates and worksheets in step 6.

9.4 About Simple Sync

The Cray Simple Sync service (cray_simple_sync) provides a simple, generic mechanism for copying user-defined content to internal and external nodes in a Cray XC system. When executed, the service automatically copies files found in source directories in the config set to one or more target nodes. The Simple Sync service is enabled by default and has no additional configuration options. It can be enabled or disabled during the initial installation using worksheets or with the `cfgset` command at any time. For more information, see man cfgset(8).

With regard to external nodes like eLogin nodes, the exclusions specified in the cray_cfgset_exclude configuration service are applied when the CLE config set is transferred to the node, and some portions of the Simple Sync directory in the config set are excluded. The "Files Excluded from eLogin Nodes" section contains more details.

Simple Sync is a simple tool and not intended as the sole solution for making configuration changes to the system. Writing custom Ansible plays might provide better maintainability, flexibility, and scalability in the long term.

How Simple Sync Works

When enabled, the Simple Sync service is executed on all internal CLE nodes and eLogin nodes at boot time and whenever the administrator executes `/etc/init.d/cray-ansible start` on a CLE node or eLogin node. When Simple Sync is executed, files placed in the following directory structure are copied to the root file system (`/`) on the target nodes.

The Simple Sync directory structure has this root:
`smw:/var/opt/cray/imps/config/sets/<config_set>/files/simple_sync/`

Below that root are the directories listed on the left. Files placed in those directories are copied to their associated target nodes.

`./common/files/`	Targets all nodes, both internal CLE nodes and eLogin nodes.
`./hardwareid/<hardwareid>/files/`	Targets a specific node with that hardware ID, which is the cname of a CLE node or the output of the `hostid` command (e.g., 1eac0b0c) on other nodes. An admin must create both the `<hardwareid>` directory and the `files` directory.
`./hostname/<hostname>/files/`	Used ONLY for eLogin nodes. Targets a node with the specified host name. An admin must create both the `<hostname>` directory and the `files` directory.
`./nodegroups/<node_group_name>/files/`	Targets all nodes in the specified node group. The directories for this `nodegroups` directory are automatically stubbed out when the config set is updated after node groups are defined and configured in the cray_node_groups service.
`./platform/[compute|service]/files/`	Targets all compute nodes or all service nodes, depending on whether they are placed in `platform/compute/files` or `platform/service/files`. Each time the config set is updated, the HSS data store is queried to update which nodes are service and which are compute.
`./README`	Provides brief guidance on using Simple Sync and a list of existing node groups in the order in which files will be copied. This ordering enables an administrator to predict behavior in cases where a file may be duplicated within the Simple Sync directory structure.

Simple Sync copies content into place prior to the standard Linux startup (`systemd`) and before `cray-ansible` runs any other services.

The ownership and permissions of copied directories and files are preserved when they are copied to root on the target nodes. An administrator can run `cray-ansible` multiple times, as needed, and only the files that have changed will be copied to the target nodes.

Because of the way it works, Simple Sync can be used to configure services that have configuration parameters not currently supported by configuration templates and worksheets. An administrator can create a configuration file with the necessary settings and values, place it in the Simple Sync directory structure, and it will be distributed and applied to the target nodes.

Files Excluded from eLogin Nodes

Because eLogin nodes use the cray_cfgset_exclude configuration service, some directories within the Simple Sync directory structure on the SMW can be excluded from transfer to eLogin nodes. The default “elogin_security” profile will exclude the following config set directories from being transferred to an eLogin node when the CLE config set is pushed to the node from the SMW.

- `files/simple_sync/common/files/etc/ssh`
- `files/simple_sync/common/files/root/.ssh`

To specify other areas within the Simple Sync directory structure that should not be transferred to eLogin nodes, create a customized site profile in cray_cfgset_exclude.

Simple Sync and Configuration File Management

Configuration files can be managed in one of three ways:

- Managed entirely by a site system administrator

  Such config files are considered non-conflicting because there is no potential conflict between administrator-provided content and Cray-managed content.

- Managed entirely by Cray configuration services

  Where possible, such config files have a comment at the top indicating that the file is completely under the management of the Cray service. Files that have been changed by Cray services can be identified by checking the change logs on the running node in `/var/opt/cray/log/ansible`. Simple Sync does not provide a mechanism to override changes made by Cray services. To override changes made by Cray services, refer to the documentation for the specific service.

- Jointly managed by a system administrator and by Cray config services

  These config files can contain both administrator-managed content and Cray-managed content, so there is potential for conflict. Administrator changes to Cray-managed content can be overridden. Content that is not managed by Cray is considered non-conflicting because any admin changes to it will not conflict with changes made by Cray services.

  Because Simple Sync copies administrator-provided files into place before `cray-ansible` runs, any Cray services that make small changes to jointly managed files will operate on the administrator-provided files. Afterwards, that file will contain both non-conflicting administrator-provided content as well as the changes made by the Cray service. Because these changes happen prior to Linux startup, the changes will be in place when the services start up.

Characteristics of Simple Sync

Simple Sync is:	**Simple Sync is NOT:**
for simple and straightforward use cases	a comprehensive system management solution
for copying a moderate number of moderately sized files*	intended to transfer large objects or a large volume of files
	an interface to configure Cray "turnkey" services such as ALPS, Node Health or Lightweight Log Manager (LLM)

* Bear in mind that anything in the Simple Sync directory structure is part of a config set, and a SquashFS copy of the current config set is distributed to all nodes in the system. Even though it is a reduced-size config set that is distributed, it is good practice to not add very large files to a config set, hence the use of "moderate" here.

Simple Sync:

- runs as early in the Ansible execution sequence as possible (it runs BEFORE other `cray-ansible` plays, so it can be used to make changes to files that Cray updates, like `sshd_config`)
- runs during the netroot setup sequence, so it can be used to change LNet and DVS settings, if needed
- supports node groups for targeting which system nodes to copy files to (see *About Node Groups* on page 105)

Simple Sync does not support:

- removing files
- appending to files
- changing file ownership and permissions (the permissions of the file in the config set are mirrored on-node)
- backing up files
- overriding Cray-set values (it cannot be used to change files that Cray completely overwrites, such as `alps.conf`, or change values in files that Cray modifies such as PermitRootLogin in `/etc/ssh/sshd_config`)

Cautions about the Use of Simple Sync

- Simple Sync copies files from the config set, which in the case of nodes without a persistent root file-system is cached in a compressed form, locally, in memory. As a result, each file stored in the config set uses some memory on the node. Therefore, using Simple Sync to copy binary files or large numbers of files is inadvisable.
- Be aware of differences in node environments when using Simple Sync. For example, systems configured with direct-attached Lustre (DAL) have nodes running CentOS instead of SLES. Administrators would have to be very careful to avoid putting an inappropriate configuration file into place when using the Simple Sync platform/service target in such a situation.
- Storage and distribution of verbatim config files through Simple Sync creates the potential for unintentional impact to the system when config files evolve due to software changes. Making minimal necessary changes through a site-local Ansible playbook provides more flexibility and minimizes the potential for unintended consequences.

Use Cases

Copy a non-conflicting file to all nodes

1. Place `etc/myfile` under `./common/files/` in the Simple Sync directory structure.
2. Simple Sync copies it to `/etc/myfile` on all nodes.

Copy a non-conflicting file to a service node

1. Place `etc/servicefile` under `./platform/service/files/` in the Simple Sync directory structure.

2. Simple Sync copies it to `/etc/servicefile` on all service nodes.

Copy a non-conflicting file to a compute node

1. Place `etc/computefile` under `./platform/compute/files/` in the Simple Sync directory structure.
2. Simple Sync copies it to `/etc/computefile` on all compute nodes.

Copy a non-conflicting file to a specific node

1. Place `etc/mynode` under `./hardwareid/c0-0c0s0n0/files/` in the Simple Sync directory structure.
2. Simple Sync copies it to `/etc/mynode` on c0-0c0s0n0.

Copy a non-conflicting file to a user-defined collection of nodes

1. Create a node group called "my_nodes" containing a list of nodes.
2. Update the config set.

```
smw# cfgset update p0
```

3. Place `etc/mynodes` under `./nodegroups/my_nodes/files/` in the Simple Sync directory structure.
4. Simple Sync copies it to `/etc/mynodes` on all nodes listed in node group my_nodes.

Copy to a node a file that has Cray-maintained content

To reduce the number of authentication tries from the default of six,

1. Place a version of `sshd_config` (entire file) that includes “MaxAuthTries 3” under `./nodegroups/login_nodes_x86_64/files/etc/ssh/` and `./nodegroups/login_nodes_aarch64/files/etc/ssh/` in the Simple Sync directory structure.
2. The booted system will contain both:
   - “MaxAuthTries 3” (from the files copied by Simple Sync)
   - “PasswordAuthentication yes” (from modification of file by Cray)

Copy to a node a file that is exclusively maintained by Cray

Files exclusively maintained by Cray such as `alps.conf` cannot be updated using Simple Sync. Please refer to the owning service (such as ALPS) for information on how to update the contents.

Copy to a node a file that resides on a file system that will be mounted during Linux boot

No special operational changes are necessary. However, Simple Sync will put the file in place early in the boot sequence, and then it will be over-mounted by the file system. Because Simple

Sync runs again later, it will copy the file into the mounted file system. Due to the ordering of operations, the file will not be present between the time the file system was mounted and the late execution of Ansible.

On netroot login nodes, modify an LNet modprobe parameter

1. Generate a file *my*_lnet.conf containing `options lnet router_ping_timeout=100`.
2. Place *my*_lnet.conf under `./nodegroups/login/files/etc/modprobe.d/` in the Simple Sync directory structure.
3. The `lnet router_ping_timeout` value will be 100.

Note that normally Simple Sync does not allow the user to override Cray values, but this procedure takes advantage of the standard Linux mechanism to override Kernel module options.

Copy a file with incompatible content to a node file that has Cray-maintained content

While Simple Sync allows an administrator to make changes to configuration files that are modified by Cray, be very careful to avoid introducing syntax errors or incompatible values that may cause the system to fail to operate correctly.

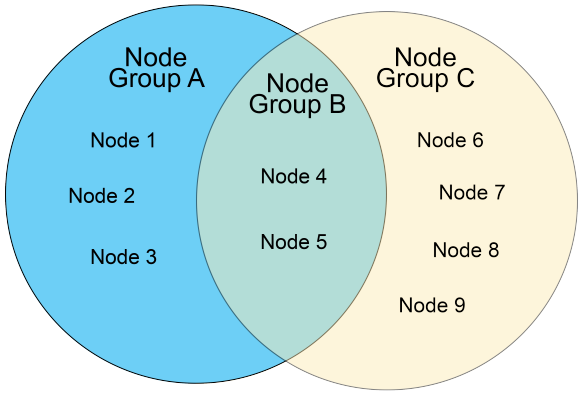
9.5 About Node Groups

The Cray Node Groups service (cray_node_groups) enables administrators to define and manage logical groupings of system nodes. Nodes can be grouped arbitrarily, though typically they are grouped by software functionality or hardware characteristics, such as login, compute, service, DVS servers, and RSIP servers.

Node groups that have been defined in a config set can be referenced by name within all CLE services in that config set, thereby eliminating the need to specify groups of nodes (often the same ones) for each service individually and greatly streamlining service configuration. Node groups are used in many Cray-provided Ansible configuration playbooks and roles and can be also used in site-local Ansible plays. Node groups are similar to but more powerful than the class specialization feature of releases prior to CLE 6.0. For example, a node can be a member of more than one node group but could belong to only one class.

The figure below demonstrates how several nodes may belong to more than one node group. In this example, node group A contains nodes 1-5, node group B contains nodes 4-5, and node group C contains nodes 4-9. Nodes 4 and 5 belong to node groups A, B, and C. In this example, if nodes 1-5 are the desired target for an Ansible play, the play can target node group A instead of specifying each node individually.

Figure 11. Node Group Member Overlap



Sites are encouraged to define their own node groups and specify their members. Administrators can define and manage node groups using any of these methods:

- Edit and upload the node groups configuration worksheet (`cray_node_groups_worksheet.yaml`).
- Use the `cfgset` command to view and modify node groups interactively with the configurator.
- Use the `cfgset get` and `cfgset modify` CLI commands to view and modify node groups at the command line. Note that CLI modifications must be followed by a config set update.

After using any of these methods, remember to validate the config set.

Characteristics of Node Groups

- Node group membership is not exclusive, that is, a node may be a member of more than one node group.
- Node group membership is specified as a list of cnames. However, if the SMW is part of a node group, it is specified with the output of the `hostid` command. Also, host names are used for eLogin nodes that are to be included in node groups.
- All compute nodes and/or all service nodes can be added as node group members by including the keywords “platform:compute” and/or “platform:service” in a node group.
- Any CLE configuration service is able to reference any defined node group by name.
- The Configuration Management Framework (CMF) exposes node group membership of the current node through the local system "facts" provided by the Ansible runtime environment. This means that each node knows what node groups it belongs to, and that knowledge can be used in Cray and site-local Ansible playbooks.

Default Node Groups

Default node groups are groups of nodes that

- are likely to be customized and used by many sites
- support useful default values for many of the migrated services

Several of the default node groups require customization by a site to provide the appropriate node membership information. This table lists the Cray default groups and indicates which ones require site customization.

Note that as of CLE 6.0.UP06, Cray no longer supports a single node group for all login nodes. Instead, there are two architecture-specific login node groups: one for all login nodes with the x86-64 architecture and one for all login nodes with the AArch64 architecture. To specify all login nodes in the system, use both of those node groups.

Table 5. cray_node_groups

Default Node Group	Requires Customization?	Notes
compute_nodes	No	Contains all compute nodes in the given partition. The list of nodes is determined at runtime.
compute_nodes_x86_64	No	Contains all x86-64 compute nodes in the given partition. The list of nodes is determined at runtime.
compute_nodes_aarch64	No	Contains all AArch64 compute nodes in the given partition. The list of nodes is determined at runtime.
service_nodes	No	Contains all service nodes in the given partition. The list of nodes is determined at runtime.
service_nodes_x86_64	No	Contains all x86-64 service nodes in the given partition. The list of nodes is determined at runtime.
service_nodes_aarch64	No	Contains all AArch64 service nodes in the given partition. The list of nodes is determined at runtime.
smw_nodes	Yes	Add the output of the `hostid` command for the SMW. For an SMW HA system, add the host ID of the second SMW also.
boot_nodes	Yes	Add the cname of the boot node. If there is a failover boot node, add its cname also.
sdb_nodes	Yes	Add the cname of the SDB node. If there is a failover SDB node, add its cname also.
login_nodes_x86_64	Yes	Add the cnames of all x86-64 internal login nodes on the system.
login_nodes_aarch64	Yes	Add the cnames of all AArch64 internal login nodes on the system. Leave empty (set to `[]`) if there are none.
elogin_nodes	Yes	Add the host names of external login nodes on the system. Leave empty (set to `[]`) if there are no eLogin nodes.
all_nodes	Maybe	Contains all compute nodes and service nodes on the system. Add external nodes (e.g., eLogin nodes), if needed.
all_nodes_x86_64	No	Contains all x86-64 nodes in the given partition. The list of nodes is determined at runtime.
all_nodes_aarch64	No	Contains all AArch64 nodes in the given partition. The list of nodes is determined at runtime.

Default Node Group	Requires Customization?	Notes
tier2_nodes	Yes	Add the cnames of nodes that will be used as tier2 servers in the cray_scalable_services configuration.

Why is there no "tier1_nodes" default node group? Cray provides a default tier2_nodes node group to support defaults in the cray_simple_shares service. Cray does not provide a tier1_nodes node group because no default data in any service requires it. Because it is likely that tier1 nodes will consist of only the boot node and the SDB node, for which node groups already exist, Cray recommends using those groups to populate the cray_scalable_services tier1_groups setting rather than defining a tier1_nodes group.

About eLogin nodes. To add eLogin nodes to a node group, use their host names instead of cnames, because unlike CLE nodes, eLogin nodes do not have cname identifiers. If eLogin nodes are intended to receive configuration settings associated with the all_nodes group, add them to that group, or change the relevant settings in other configuration services to include both all_nodes and elogin_nodes.

Additional Platform Keywords

Cray uses these two platform keywords to create default node groups that contain all compute or all service nodes.

```
platform:compute
platform:service
```

New in CLE 6.0.UP06:

- Additional platform keywords are used to create pre-populated node groups that contain all compute or service nodes with the x86-64 or AArch64 architecture.

  ```
  platform:compute-X86
  platform:service-X86
  platform:compute-ARM
  platform:service-ARM
  ```

- All platform keywords, such as `platform:compute`, `platform:service-ARM`, and `platform:compute-HW12`, include nodes that have been disabled.
- A new `platform:disabled` keyword can be used by administrators to identify disabled nodes.
- (not new but previously undocumented) Groups of nodes can be excluded using a negation operator: `~` (the tilde symbol). For example, a node group that contains all enabled compute and service nodes would have the following list as its members:

  ```
  - platform:compute
  - platform:service
  - ~platform:disabled
  ```

  The ordering of the list does not matter: all non-negated keywords are resolved first, then negated ones are removed.

Sites that need finer-grained groupings can use additional platform keywords to create custom node groups. For a node group that contains all compute or service nodes with a particular processor/core type, use one of the following platform keywords.

```
platform:compute-XX##
platform:service-XX##
```

For *XX##*, substitute a processor/core code, such as KL64 or KL68, which designate two Intel® Xeon Phi™ "Knights Landing" (KNL) processors with different core counts. To find the code associated with each node on a Cray system, use the `xtcli status p0` command and look in the "Core" column of the output, as shown in the following example.

```
smw# xtcli status p0
Network topology: class 0
Network type: Aries
Nodeid: Service Core Arch| Comp state [Flags]
----------------------------------------------------
c0-0c0s0n0: service BW18 X86| ready [noflags|]
c0-0c0s0n1: service BW18 X86| ready [noflags|]
c0-0c0s0n2: service BW18 X86| ready [noflags|]
c0-0c0s0n3: service BW18 X86| ready [noflags|]
c0-0c0s1n0: service BW18 X86| ready [noflags|]
c0-0c0s1n1: service BW18 X86| ready [noflags|]
c0-0c0s1n2: service BW18 X86| ready [noflags|]
c0-0c0s1n3: service BW18 X86| ready [noflags|]
c0-0c0s2n0: - HW12 X86| ready [noflags|]
c0-0c0s2n1: - HW12 X86| ready [noflags|]
c0-0c0s2n2: - HW12 X86| ready [noflags|]
c0-0c0s2n3: - HW12 X86| ready [noflags|]
```

The following table lists some of the common processor/core codes supported by Cray.

Table 6. Cray Supported Intel Processor/Core (XX##) Codes

Processor (XX)	Core (##)	Intel Code Name
BW	12, 14, 16, 18, 20, 22, 24, 28, 32, 36, 40, 44	"Broadwell"
HW	04, 06, 08, 10, 12, 14, 16, 18, 20, 24, 28, 32, 36	"Haswell"
IV	02, 04, 06, 08, 10, 12, 16, 20, 24	"Ivy Bridge"
KL	60, 64, 66, 68, 72	"Knights Landing"
SB	04, 06, 08, 12, 16	"Sandy Bridge"
SK	04, 08, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56	"Skylake"

9.6 About Config Set Caching

Config sets are defined and reside on the Server of Authority, which on XC systems is the SMW. Config set content is made available to all nodes in the system by means of Cray Scalable Services.

To make the sharing of config set content both quick and reliable, the cray-cfgset-cache service was created. It caches config sets locally on nodes (compressed for a smaller footprint). On the SMW, it does the following:

- notices changes to config sets on the SMW
- refreshes the local caches dynamically

- detects failures and retries automatically

The cray-cfgset-cache service ensures that config set content gets refreshed on all nodes whenever config sets are created or updated on the SMW. It is triggered when `cray-ansible` is run on a node with the `start`, `restart`, or `link` commands.

> **ATTENTION:** If the cray-cfgset-cache service is stopped, config set content in node-local memory will not get refreshed when `cray-ansible` is run. If that happens, nodes will continue to use the most recent compressed copy of the config set data created before the service was stopped.

What Gets Cached

The cray-cfgset-cache service does not copy an entire config set to node-local memory. Instead, it uses the config set on the SMW to create these two files in the root of the config set:

- a compressed copy of the config set using SquashFS tools, (typically < 3 MB)
- a checksum of the compressed copy of the config set

The compressed copy is made available (effectively copied) to node-local RAM, and the checksum is used to know when the config set in node-local memory no longer matches the config set on the SMW. Even though Scalable Services makes the entire config set directory structure on the SMW available to the rest of the system, only the compressed copy and its associated checksum are used by nodes. They are the key to the performance, scalability, and reliability improvements provided by config set caching.

When `cray-ansible` is run on a node, the node will do the following:

1. Check to see if the cached node-local version of the compressed config set is out of date.
2. If it is stale, replace it with a newer version available on the SMW and start using that newer version.