Introduction to Cray Data Virtualization Service

S–0005–4002

RECORD OF REVISION

S–0005–4002 Published December 2011 Supports the Cray Linux Environment (CLE) 4.0.UP02 and the System Management Workstation (SMW) 6.0.UP02 releases.

S–0005–3102 Published December 2010 Supports the Cray Linux Environment (CLE) 3.1.UP02 and the System Management Workstation (SMW) 5.1 releases.

3.1 Published June 2010 Supports the Cray Linux Environment (CLE) 3.1 and the System Management Workstation (SMW) 5.1 releases.

3.0 Published March 2010 Supports the Cray Linux Environment (CLE) 3.0 and the System Management Workstation (SMW) 5.0 releases.

2.2 Published July 2009 Supports general availability versions of Cray Linux Environment (CLE) release running on Cray XT systems. Document updated to reflect installation and configuration changes.

1.0 Published January 2008 Supports limited availability versions of Cray DVS for the UNICOS/lc 2.0 release running on Cray XT systems.

Changes to this Document

Introduction to Cray Data Virtualization Service S–0005–4002

Changes to this manual reflect features implemented since the CLE 4.0 release:

- Added information on DVS Fairness of Service in DVS Fairness of Service on page 21.

Contents

Page

Procedures

Examples

Tables

Figures

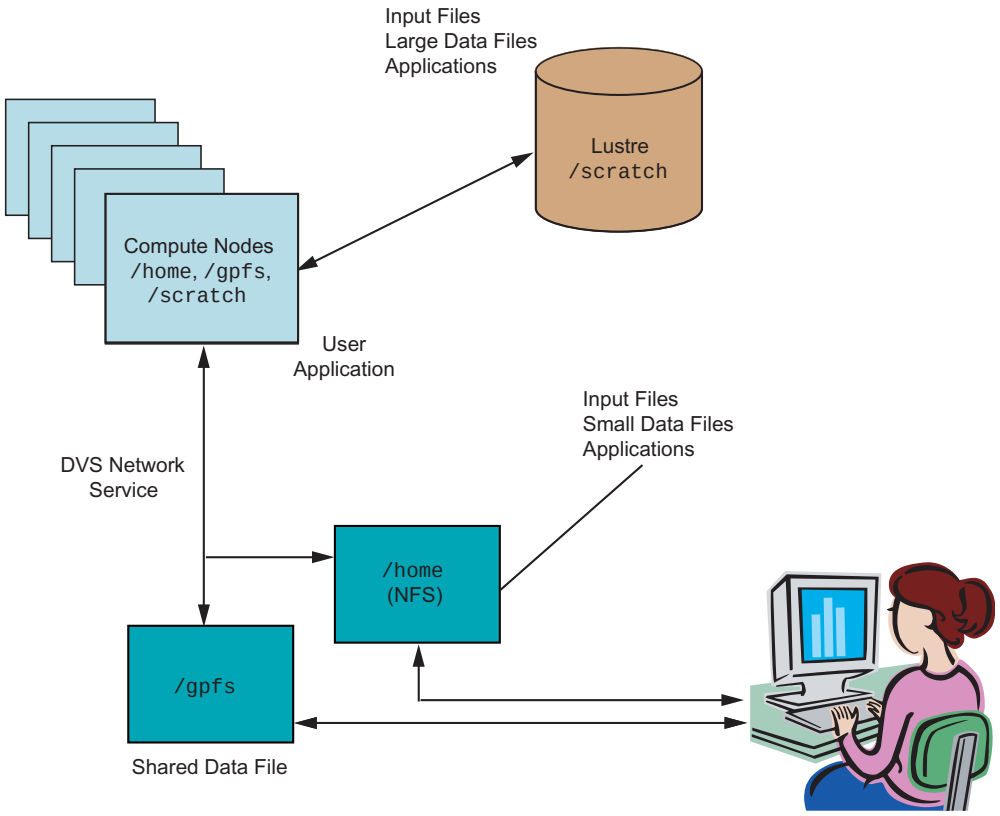
Introduction [1]

The Cray Data Virtualization Service (Cray DVS) is a distributed network service that provides transparent access to file systems residing on the service I/O nodes and remote servers in the data center. Cray DVS provides a service analogous to NFS. It 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 options to the `mount` command enable clients (compute nodes) access to a file system being projected by DVS servers. Thus, Cray DVS, while not a file system, represents a software layer that provides scalable transport for file system services.

Cray DVS provides I/O performance and scalability to a large numbers of nodes, far beyond the typical number of clients supported by a single NFS server. Impact on compute node memory resources, as well as operating system noise, is minimized in the Cray DVS configuration.

Cray DVS provides support for access to Linux Virtual File System Switch (VFS)-based file systems. DVS clients use Resiliency Communication Agent (RCA) events to determine when server nodes have failed or when DVS has been unloaded from a server node and when server nodes have been booted and DVS is re-loaded. 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 re-routing I/O operations away from downed servers and back to rebooted servers. Figure 1 presents a typical Cray DVS use case.

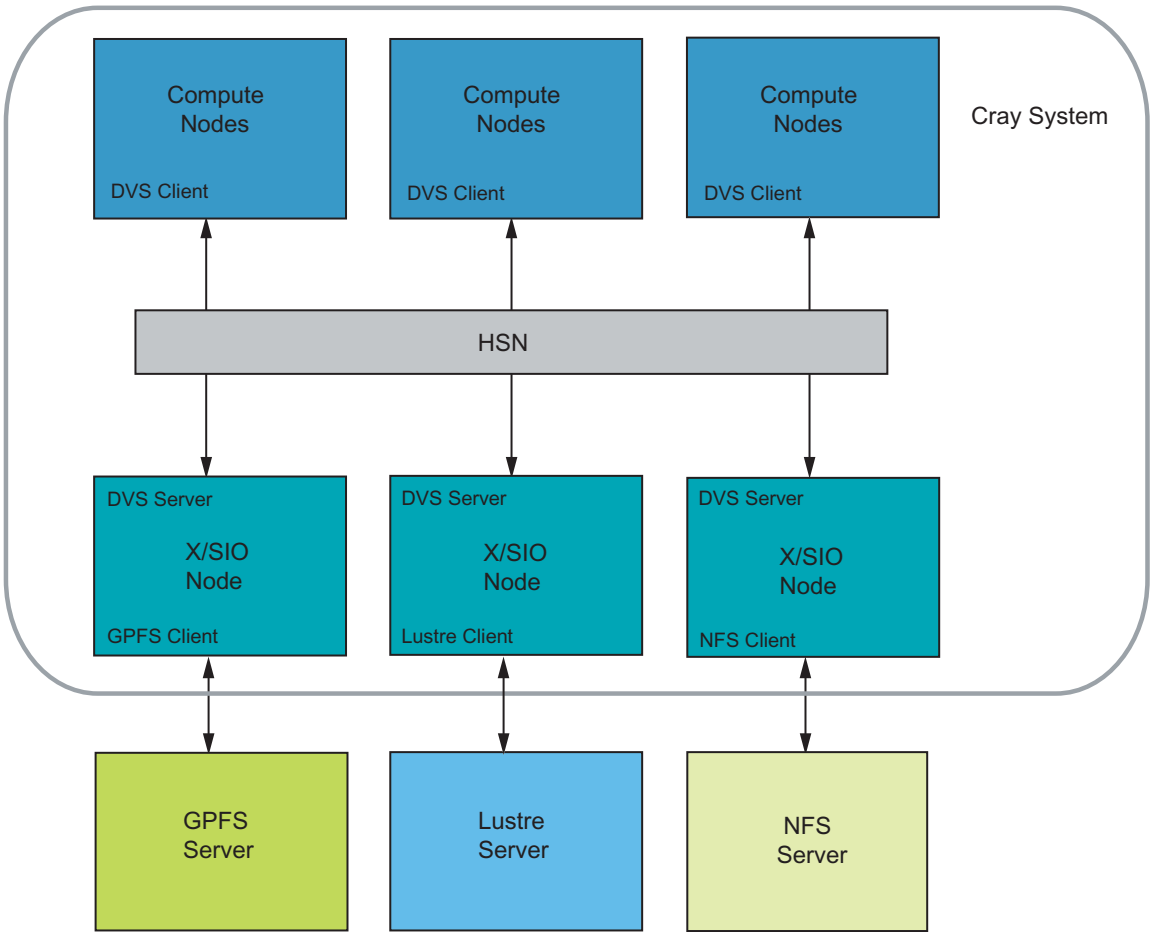
Figure 1. Cray DVS Use Case



See the `mount`(8) and `dvs`(5) man pages for more information.

Figure 2 illustrates the system administrator's view of Cray DVS. Administration of Cray DVS is very similar to configuring and mounting any Linux file system.

Figure 2. Cray DVS In a Cray System



DVS Modes [2]

Cray DVS uses the Linux-supplied 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 (GPFS). If you would like to use Cray DVS as a file systems solution for your Cray system, contact your Cray service representative for more information.

There are two ways to use Cray DVS: in either serial or parallel access modes. In serial mode, one DVS server on a Cray service node projects a file system to a number of compute node clients. Parallel modes comprise multiple servers in configurations that vary in purpose, layout, and performance.

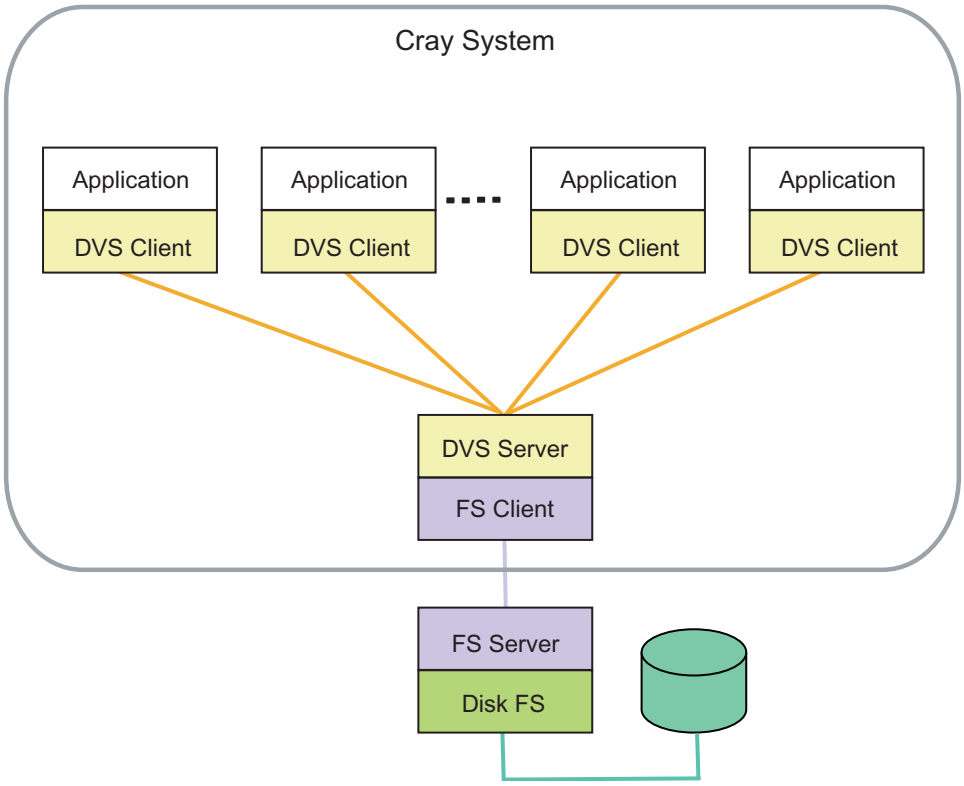
Table 1. Cray DVS Access Modes

Mode	Access Level	Pattern
Serial	Read/Write	Many clients, one server
Parallel	Read/Write	Many clients, many servers

2.1 Serial Mode

Serial mode is the simplest implementation of DVS where each file system is projected from a single DVS server. DVS can project multiple file systems in serial mode by assigning a new or an existing DVS server to each additional file system in serial access mode and entering the appropriate mount point on the clients. DVS projecting one file system is shown in Figure 3.

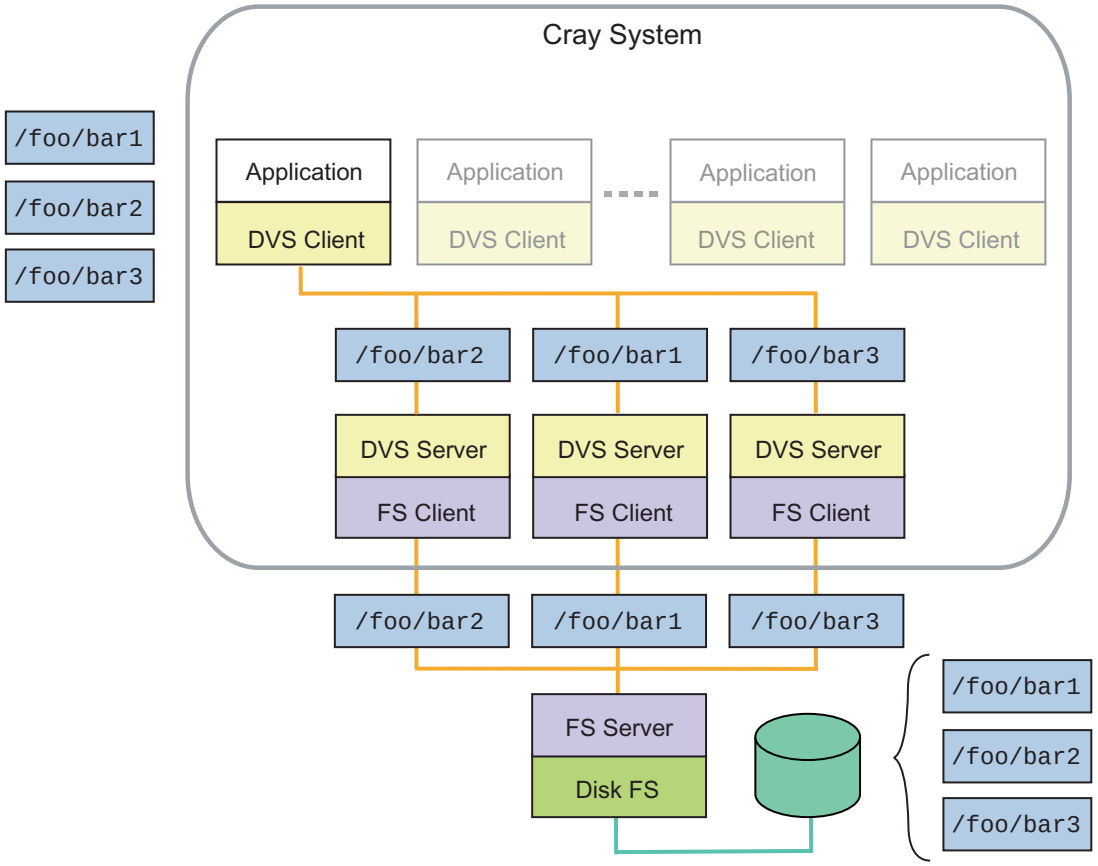
Figure 3. Cray DVS Serial Access Mode



2.2 Cluster Parallel Mode

In cluster parallel access mode (shown in Figure 4), a single client interacts with multiple servers. The server used to perform the read, write, or metadata operation is selected using an internal hash involving the underlying file or directory inode number. All I/O from all clients involving the same file will route to the same server to prevent file system coherency thrash.

Figure 4. Cray DVS Cluster Parallel Access Mode

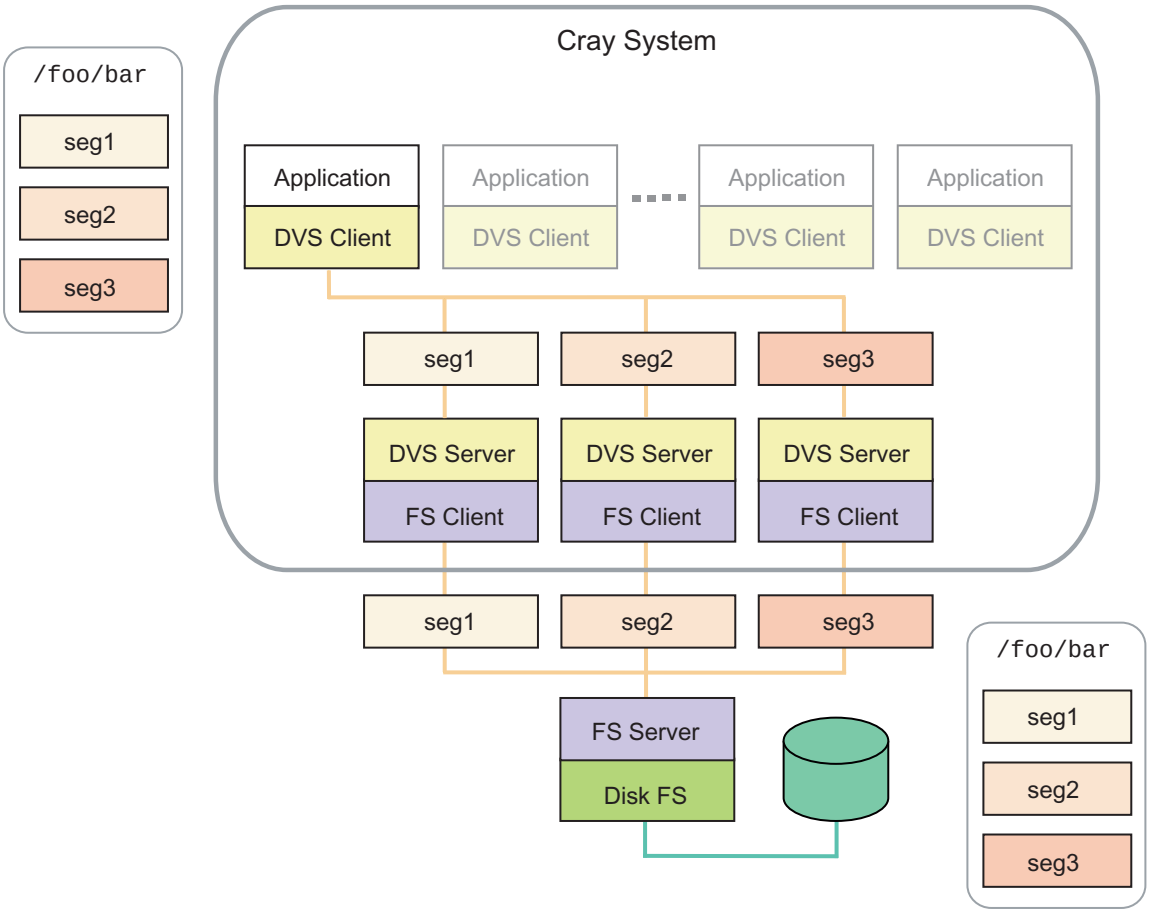


2.3 Stripe Parallel Mode

Stripe parallel mode provides an extra level of parallelized I/O forwarding for clustered file systems. Each DVS server 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. Stripe parallel mode provides the opportunity for greater aggregate I/O bandwidth when forwarding I/O from a coherent cluster file system. GPFS has been tested extensively using this mode. All I/O from all clients involving the same file will route each DVS block size of file data to the same server to prevent file system coherency thrash.

Warning: NFS cannot be used in Stripe parallel mode as NFS implements close-to-open cache consistency, and thus striping data across the NFS clients could result in data integrity issues.

Figure 5. Cray DVS Stripe Parallel Mode



2.3.1 Atomic Stripe Parallel Mode

Stripe parallel mode provides parallelism within a file at the granularity of the DVS block size. It does not, however, strictly adhere to POSIX read/write atomicity rules. POSIX read/write atomicity means that all bytes associated with a read or write are guaranteed to not be interleaved with bytes from other read or write operations. When application-level file locking is not used, stripe parallel mode can not guarantee POSIX read/write atomicity. 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 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. 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, but also allows DVS clients to access different servers for subsequent I/O requests if they have different starting offsets within the file. Administrators select atomic stripe parallel mode by adding the 'atomic' mount option to an otherwise normal stripe parallel mount point. Users can request atomic stripe parallel mode by setting the `DVS_ATOMIC` user environment variable to `on`. Serial, cluster parallel, and loadbalance modes all adhere to POSIX read/write atomicity rules.

2.4 Loadbalance Mode

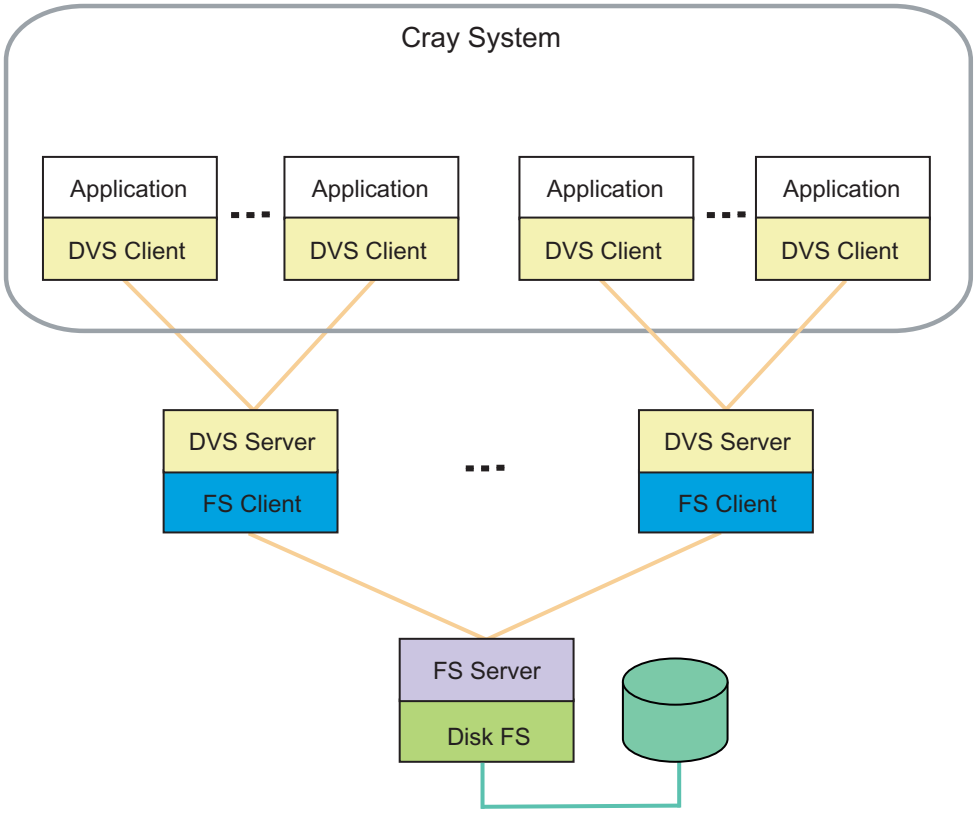
Loadbalance mode is a client access mode for DVS used exclusively for the compute node root runtime environment to use with dynamic shared objects (see Compute Node Root Runtime Environment on page 17 for more information). The clients, Cray system compute nodes, automatically select the server based on a DVS-internal node ID (NID) from the list of available server nodes specified on the `/etc/fstab` line. Loadbalance mode is only valid for read-only mount points. In the case of compute node root servers, the underlying file system is the NFS-exported shared root. Loadbalance mode automatically enables failover to another DVS server specified on the `/etc/fstab` line.

DVS automatically enables the `cache` mount option in loadbalance mode as it is a read-only mode. This means that a DVS client will pull data from the DVS server the first time it is referenced, but then the data will be stored in the client's page cache. All future references to that data will be local to the client's memory and DVS will not be involved at all. If the node runs low on memory, the Linux kernel may remove these pages however, and at that point the client will have to re-fetch the data from the DVS server on the next reference to repopulate the client's page cache.

Administrators may also use the `attrcache_timeout` mount option to enable attribute caching for `loadbalance` mount points. 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 as the file system is read-only and thus attributes are not likely to change.

Note: When one of the compute node root servers is not responsive, requests will failover to other nodes in the list.

Figure 6. Cray DVS Loadbalance Mode



2.4.1 Compute Node Root Runtime Environment

Dynamic shared objects are supported for Cray Linux Environment (CLE). This means that applications developers can compile, link and load dynamic shared objects. Applications that are compiled with dynamic objects and libraries (DSLs) do not need to be recompiled if these libraries are updated. Memory footprint and application size are reduced by using DSLs. This feature is facilitated by projecting the shared root to compute nodes using DVS in loadbalanced mode. The compute nodes then use the shared root instead of `initramfs` to compile and run applications. The system administrator is also allowed to repurpose compute nodes as DVS servers for the purpose of projecting the NFS shared root to other compute nodes. If the administrator elects to re-purpose compute nodes as DVS servers, end-users will no longer be allowed to use this as a traditional compute node. This framework is called the compute node root runtime environment (CNRTE).

2.4.2 Cluster Compatibility Mode (CCM)

Cluster Compatibility Mode (CCM) is a software solution that provides the services needed to run most cluster-based independent software vendor (ISV) applications out-of-the-box with some configuration adjustments. CCM supports ISV applications running in four simultaneous cluster jobs on up to 256 compute nodes per job instance. It is built on top of CNRTE and by extension Cray DVS. For more information, see *Workload Management and Application Placement for the Cray Linux Environment* and *Managing System Software for Cray XE and Cray XK Systems*.

2.5 Failover

Cray DVS supports failover and failback by default for loadbalance mode as well as cluster, and stripe, atomic stripe parallel with the addition of the `failover` option to the `/etc/fstab` entry.

DVS failover and failback is done in an active-active manner. Multiple servers must be specified in the `/etc/fstab` entry for failover and failback to function. When a server fails, it is taken out of the list of servers to use for the mount point until it is rebooted. All open and new files will use the remaining servers as described by the cluster, stripe, and atomic stripe parallel sections. Files not using the downed server are not affected.

When failover occurs the following will happen:

- If all servers fail, I/O will be retried as described by the `retry` option in Additional `/etc/fstab` Options on page 25.
- Any mount point using loadbalance mode will automatically re-calibrate the existing client-to-server routes to ensure the clients are evenly distributed across the remaining servers. When failback occurs, this process will be repeated.

- Any mount point using cluster parallel mode will automatically redirect I/O to one of the remaining DVS servers for any file that previously routed to the now-down server. The failover server will be determined in the same manner any server is: using the inode value of the underlying file hashed with the number of available servers. When failback occurs, these files will be re-routed back to their original server.
- Any mount point using stripe parallel, or atomic stripe parallel mode will automatically re-stripe I/O across the remaining DVS servers in an even manner. The striping will be determined in the same manner any stripe pattern is: using the inode value of the underlying file and the offsets of data into the file relative to the DVS `blksize` value and the number of available servers. When failback occurs, files will be re-striped back to their original pattern.

2.5.1 Examples of Messages Indicating a Failover Event

Example 1. Client System Console Message: "DVS: file_node_down: removing *c0-0c2s1n3* from list of available servers for *2* mount points"

When you see the following message:

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

it indicates that a DVS server has failed. In this example, *c0-0c2s1n3* is the DVS server and has been removed from the list of available mount points provided in the `/etc/fstab` entry for the DVS projection.

Once the issue is resolved, the following message will be 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
```

2.6 Cray DVS Statistics

DVS statistics are available for both client and server nodes in CLE.

A count of file system operations are available via the `/proc/fs/dvs/stats` file. Each line of this file displays a file system operation and a count of successful and failed operations of that type. The `/proc/fs/dvs/stats` file is used for file system operations that can not be correlated to a specific DVS mount point, and is thus most interesting on DVS servers.

The same type of information is also available in per-mount point files on DVS clients (`/proc/fs/dvs/mounts/0/stats`, `/proc/fs/dvs/mounts/1/stats`, etc.) Each of these files contains counts of successful and failed file system operations for that specific mount point only. More information about each of these mount points can be obtained by viewing the mount file that resides in the same directory (e.g., `/proc/fs/dvs/mounts/0/mount`).

In addition, the `/proc/fs/dvs/ipc/stats` file displays DVS IPC statistics such as bytes transferred and received, NAK counts, etc. It also displays message counts broken down by type and size.

DVS statistics are enabled and collected by default. Each DVS statistics file can be disabled by a privileged user by writing a zero into the file (e.g., '**`echo 0 > /proc/fs/dvs/stats`**'). To re-enable a DVS statistic file, write a 1 into the file (e.g., '**`echo 1 > /proc/fs/dvs/stats`**'). To reset the statistic values to zero, write a 2 into the file (e.g., '**`echo 2 > /proc/fs/dvs/stats`**').

Cray DVS Configuration [3]

3.1 DVS on Systems Using Gemini

DVS uses Lustre Networking (LNET) to communicate on Gemini networks, and LNET in turn uses the Generic Lustre Network Driver (gnilnd). The corresponding RPMs must be installed on Cray XE systems for DVS to function properly. In addition, the following line must be in `/etc/modprobe.conf.local` on DVS servers and in `/etc/modprobe.conf` on DVS clients:

```
options lnet networks=gni
```

3.2 DVS Fairness of Service

In CLE 4.0.UP02, DVS is updated to create user or job specific request queues for clients. Originally, DVS had used one queue to handle requests in a FIFO (first-in, first out) fashion. This means 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) completes. Fairness of Service creates a pool of request processing threads for the server and a list of queues—one for each client and/or job. The list 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 the chosen queue. This helps to distribute the workload and potentially helps contending applications perform better. Fairness of Service is enabled by default.

Procedure 1. Disabling DVS Fairness of Service

1. Stop DVS on all servers.
2. Enter `xtopview` on the boot node in either `default` or a customized DVS server view if you have one.
3. Edit `/etc/modprobe.d/dvs` and enter the following line:

   ```
   options dvsipc_single_msg_queue=1
   ```
4. Exit `xtopview`.
5. Restart all DVS servers.

3.3 Modifying DVS Configuration for CLE

This section assumes that you have already installed the appropriate RPMs using the `CLEinstall` program. Administration of Cray DVS is very similar to configuring and mounting any Linux file system. For more information, see the `dvs`(5) man page.

Caution: DVS servers use unlimited amounts of CPU and memory resources based directly on the I/O requests sent from DVS clients. For this reason, DVS servers should be isolated and not share nodes with other services (e.g., Lustre nodes, login nodes, etc.).

Procedure 2. Configuring the system to mount DVS file systems

After Cray DVS software has been successfully installed on both the service and compute nodes, you can `mount` a file system on the compute nodes that require access to the network file system that is mounted on DVS server nodes. When a client mounts the file system, all of the necessary information is specified on the `mount` command.

Note: The node that is projecting the file system needs to mount it. Therefore, if the file system is external to the Cray, the DVS server must have external connectivity.

At least one DVS server must be active when DVS is loaded on the client nodes to ensure that all DVS mount points are configured to enable higher-level software, such as the compute node root runtime environment (CNRTE), to function properly.

The following example configures a DVS server at `c0-0c0s4n3` (node 23 on a Cray XE system) to project the file system that is served via NFS from *nfs_serverhostname*. For more information about Cray DVS mount options, see the `dvs`(5) man page.

Note: To make these changes for a system partition, rather than for the entire system, replace `/opt/xt-images/templates` with `/opt/xt-images/templates-p`*N*, where *N* is the partition number.

1. Enter `xtopview` with the node view for your DVS server and create the `/dvs-shared` directory that you will be projecting */nfs_mount* from.

```
boot:~ # xtopview -n 23
node/23:/ # mkdir /dvs-shared
```

2. Specialize the `/etc/fstab` file for the server and add a DVS entry to it.

```
node/23:/ # xtspec -n 23 /etc/fstab
node/23:/ # vi /etc/fstab
nfs_serverhostname:/nfs_mount /dvs-shared nfs tcp,rw 0 0
node/23:/ # exit
```

3. Log into the DVS server and mount the file system:

```
boot:~ # ssh nid00023
nid00023:/ # mount /dvs-shared
nid00023:/ # exit
```

4. Create mount point directories in the compute image for each DVS mount in the `/etc/fstab` file. For example, type the following command from the SMW:

```
smw:~ # mkdir -p /opt/xt-images/templates/default/dvs
```

5. (Optional) Create any symbolic links that are used in the compute node images. For example:

```
smw:~ # cd /opt/xt-images/templates/default
smw:/opt/xt-images/templates/default # ln -s dvs link_name
```

6. To allow the compute nodes to mount their DVS partitions, add an entry in the `/etc/fstab` file in the compute image and add entries to support the DVS mode you are configuring.

```
smw:~# vi /opt/xt-images/templates/default/etc/fstab
```

For serial mode, add a line similar to the following example which mounts `/dvs-shared` from DVS server `c0-0c0s4n3` to `/dvs` on the client node.

```
/dvs-shared /dvs dvs path=/dvs,nodename=c0-0c0s4n3
```

For cluster parallel mode, add a line similar to the following example which mounts `/dvs-shared` from multiple DVS servers to `/dvs` on the client node. Setting `maxnodes` to 1 indicates that each file hashes to only one server from the list.

```
/dvs-shared /dvs dvs path=/dvs,nodename=c0-0c2s1n0:c0-0c2s1n3:c0-0c2s2n0,maxnodes=1
```

For stripe parallel mode, add a line similar to the following example which mounts `/dvs-shared` from the DVS servers to `/dvs` on the client nodes. Specifying a value for `maxnodes` greater than 1 or removing it altogether makes this stripe parallel access mode.

```
/dvs-shared /dvs dvs path=/dvs,nodename=c0-0c2s1n0:c0-0c2s1n3:c0-0c2s2n0
```

For atomic stripe parallel mode, add a line similar to the following example which mounts `/dvs-shared` from the DVS servers to `/dvs` on the client nodes. Specifying atomic makes this atomic stripe parallel mode as opposed to stripe parallel mode.

```
/dvs-shared /dvs dvs path=/dvs,nodename=c0-0c2s1n0:c0-0c2s1n3:c0-0c2s2n0,atomic
```

For loadbalance mode, add a line similar to the following example to project `/dvs-shared` from multiple DVS servers to `/dvs` on the client node. The `ro` and `cache` settings specify to mount the data read-only and cache it on the compute node. The `attrcache_timeout` option specifies the amount of time

in seconds that file attributes remain valid on a DVS client after they are fetched from a DVS server. Failover is automatically enabled and does not have to be specified.

```
/dvs-shared /dvs dvs path=/dvs,nodename=c0-0c2s1n0:c0-0c2s1n3:c0-0c2s2n0,\
loadbalance,cache,ro,attrcache_timeout=14400
```

7. If you set `CNL_dvs=yes` in `CLEinstall.conf` before you ran the `CLEinstall` program, update the boot image (by preparing a new compute and service node boot image.)

   Otherwise, you must first edit the `/var/opt/cray/install/shell_bootimage_`*label*`.sh` script and set `CNL_DVS=y` and then update the boot image.

   Note: It is important to keep `CLEinstall.conf` consistent with changes made to your configuration in order to avoid unexpected changes during upgrades or updates. Remember to set `CNL_dvs` equal to `yes` in `CLEinstall.conf`.

3.4 Starting DVS Automatically

If you have configured your system to run DVS, you can start DVS servers using `chkconfig`. If `c0-0c0s4n3` is being used as a DVS server, use the `chkconfig` command on the DVS server node's view in `xtopview`:

```
boot:~ # xtopview -n nid00019 -m "chkconfig dvs on"
       # chkconfig --force dvs on
       # exit
```

Additional Mount Options and Environment Variables [4]

4.1 Additional User Environment Variables and Client Mount Options

4.1.1 Additional `/etc/fstab` Options

There are several options that can be inserted into DVS client mount points:

- `atomic` enables atomic stripe parallel mode. This ensures that stripe parallel requests adhere to POSIX read/write atomicity rules. DVS clients will 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 `nodename` or `nodefile` lists multiple DVS servers, and neither loadbalance or cluster parallel mode is specified, DVS will stripe I/O requests across multiple servers and not necessarily adhere to POSIX read/write atomicity rules if file locking is not used. This is the default behavior.
- *nodefile* is equivalent to *nodename* but allows the administrator to specify a list of server nodes in a file instead of placing them on the mount line directly. This provides more convenience for large sites that may employ many DVS server nodes. Node names are separated by a new line or a colon `(:)` character.
- `attrcache_timeout` enables client-side attribute caching. File attributes and `dentries` for `getattr` requests, pathname lookups, etc. will be read from DVS servers and cached on the DVS client for *n* seconds. Additional lookups or `getattr` requests will use the cached attributes until the timeout expires, at which point they will be read and cached again on first reference.

  Attribute caching can have an extremely positive impact on 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.

- `blksize=`*n* sets the DVS block size to *n* bytes. The default value is 16384.
- `cache` enables client-side read caching. The client node will perform caching of reads from the DVS server node and provide data to user applications from the page cache if possible, instead of performing a data transfer from the DVS server node.

  Note: Cray DVS is not a clustered file system; No coherency is maintained between 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 caching. This is the default behavior.
- `closesync` enables data synchronization upon last close of the file. When the DVS server receives the last close, it will wait until all data has been written back to the projected file system. This functionality serves to prevent lost data in the event of a server node crash after an application has terminated.
- `noclosesync` is the default behavior of DVS. In the default case, DVS will return a `close()` request immediately.
- `datasync` enables data synchronization. The DVS server node will wait until data has been written to the underlying media before indicating that the write has completed.
- `nodatasync` disables data synchronization. The DVS server node will return from a write request as soon as the user's data has been written into the page cache on the server node. This is the default behavior.
- `deferopens` defers DVS client open requests to DVS servers for a given set of conditions. When a file is open in stripe parallel or atomic stripe parallel modes DVS clients will only send the open request to a single DVS server. Additional open requests will be 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` (since 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 or atomic stripe parallel mode, DVS clients will send open requests to all DVS servers denoted by nodename or nodefile. This is the default behavior.

- `failover` enables failover and failback of DVS servers. If multiple DVS servers are listed for a single DVS mount point and one or more of the servers fails, operations for that mount point will continue by using the subset of servers still available. When the downed servers are rebooted and start DVS, any client mount points that had performed failover operations will failback to once again include the servers as valid nodes for I/O forwarding operations. The failover option can not be specified at the same time as the `noretry` option. If all servers fail, operations for the mount point will behave as described by the `retry` option until the at least one server is rebooted and has loaded DVS. This is the default behavior.
- `nofailover` disables failover and failback of DVS servers. If one or more servers for a given mount point fail, operations for that mount point will be behave as described by the corresponding `retry` or `noretry` option specified for the mount point.
- `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. This is the default behavior.
- `noretry` disables retries of user I/O requests when the DVS server receiving the request is down.
- `killprocess` enables the killing processes that have written data to a DVS server when a server fails. This ensures that a process can not be affected by silent data loss if there was data residing in the Linux kernel page cache and not backing store when the server failed. See ALPS: "DVS server failure detected: killing process to avoid potential data loss" on page 31 for additional information. This is the default behavior.
- `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 will not be killed. Should a process continue to perform operations with an open file descriptor that had been used to write data to the server, the operations will fail (with `errno` set to `EHOSTDOWN`). A new open of the file will be allowed and addition operations with the corresponding file descriptor will function normally.
- `userenv` argument informs DVS to honor end user environment variable overrides for DVS `mount` options. This is the default case with DVS.
- `nouserenv` argument allows the administrator to block end user environment variable overrides for DVS mount options.

- `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 will verify that the underlying file system has a magic value that matches the specified value. If not, the DVS client will exclude that DVS server from the list of servers it utilizes for the mount point and print 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 will 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
  Lustre servers 0x0bd00bd1
  Lustre clients 0x0bd00bd0
  ```

4.1.2 `/etc/fstab` Examples

Example 2. Configuring cluster parallel with a *nodefile*

From the SMW, enter the `/etc/fstab` file for the compute node boot images:

```
smw:~# vi /opt/xt-images/templates/default/etc/fstab
```

`c0-0c0s2n3` and `c0-0c0s3n0` are the DVS servers for a single cluster file system. For a `nodefile`, `/opt/xt-images/templates/default/etc/`*nodefile*`,` enter an entry similar to the following:

```
/dvs-shared /dvs dvs path=/dvs,nodefile=/etc/nodefile,maxnodes=1
```

Use your preferred method for updating this file. For example, simple editing works for small sets of nodes:

```
smw:~# vi /opt/xt-images/templates/default/etc/nodefile
```

File contents for *nodefile* in the example look like this:

```
c0-0c0s2n3
c0-0c0s3n0
```

Or

```
c0-0c0s2n3:c0-0c0s3n0
```

Example 3. Configuring cluster parallel access mode with the `closesync` option

From the SMW, enter the `/etc/fstab` file for the compute node boot images:

```
smw:~# vi /opt/xt-images/templates/default/etc/fstab
```

```
/dvs-shared /dvs dvs path=/dvs,nodefile=/etc/nodefile,\
maxnodes=1,closesync
```

4.1.3 Environment Variables

By default, user environment variables allow client override of options specified in the `/etc/fstab` entry. However, if the `nouserenv` option is specified in the DVS entry, then user environment variables are disabled.

The following are the environment variables that you can use in the default case:

Table 2. Cray DVS User-environment Variables

Variable Name	Options	Purpose
DVS_DATASYNC	on|off	Sets the behavior for the `datasync` or `nodatasync` mount options.
DVS_CLOSESYNC	on|off	Sets the behavior for the `closesync` or `noclosesync` mount options.
DVS_CACHE	on|off	Sets the behavior for the `cache` or `nocache` mount options. This enables client-side read caching.
DVS_BLOCKSIZE	n	This non-zero number, `n`, overrides the `blksize` mount option.
DVS_MAXNODES	n	This non-zero 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.
DVS_ATOMIC	on|off	Sets the behavior for the `atomic` or `noatomic` mount options.
DVS_KILLPROCESS	on|off	Sets the behavior for the `killprocess` or `nokillprocess` mount options.
DVS_DEFEROPENS	on|off	Sets the behavior for the `deferopens` or `nodeferopens` mount options.

Troubleshooting [5]

The following information may be useful in case of errors related to DVS.

5.1 ALPS: "DVS server failure detected: killing 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 were to crash before the data is written to backing store, this data will be lost. To prevent silent data loss, DVS will kill the processes on the clients which wrote the data. If the Application Level Placement Scheduler (ALPS) was used to launch the application, the following message will be displayed to the user's terminal before `aprun` exits: "DVS server failure detected: killing process to avoid potential data loss."

The `datasync` option for the client `/etc/fstab` entry or the `DVS_DATASYNC` user environment variable will avoid this because each write operation will be followed by a `fsync` before it is considered complete.

Using the `nokillprocess` option for the client `/etc/fstab` entry or setting the `DVS_KILLPROCESS` user environment variable to off will also avoid this. In these scenarios, when a server fails, processes that have written data to the server will not be killed. Should a process continue to perform operations with an open file descriptor that had been used to write data to the server, the operations will fail (with `errno` set to `EHOSTDOWN`). A new open of the file will be allowed and addition operations with the corresponding file descriptor will function normally.

5.2 Application Hangs As a Result of NFS File Locking

This type of scenario is encountered when file locking is used. It is specific to NFS file systems projected through DVS. If you encounter this issue, specify the `nolock` option in the NFS mount point on DVS servers. See the `nfs`(5) man page for more information on the `nolock` option.

5.3 Caveats and Limitations

5.3.1 DVS `blksize` Must Match Or Be a Multiple of GPFS Block Size

If you are projecting a General Parallel File System (GPFS) cluster the client `mount` option, `blksize` must match or be a multiple of the GPFS blocksize. If you are projecting multiple GPFS file systems that have different block sizes, it's necessary to have different `/etc/fstab` entries for each file system.

For example, with two GPFS file systems, one with a 64 kilobyte (KB) block size, and another with a 1024KB block size, the `/etc/fstab` entries for DVS would look like the following:

```
/gpfs1 /dvs1 dvs path=/dvs1,nodefile=/etc/nidlist1,blksize=65536
/gpfs2 /dvs2 dvs path=/dvs2,nodefile=/etc/nidlist2,blksize=1048576
```

5.3.2 `mmap()` Support

DVS only supports read-only access when the `MAP_SHARED` flag is specified to `mmap()`. This is because writable shared mmaps are cached in the Linux kernel page cache and DVS does not maintain consistency between DVS clients and DVS servers. Thus the file data cached on the DVS client may not be kept coherent with other accesses to the file data from other nodes.

Write access is available when the `MAP_PRIVATE` flags is specified to `mmap()` since the file data is private to the process that performed the `mmap()` and thus coherency with other processes is not a concern.

5.3.3 Client Consistency

DVS supports close-to-open consistency, meaning that files on client and server are consistent at `open()` and `close()`. While the file is open, DVS does not guarantee that the files will be consistent.

5.3.4 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 your Cray service representative for more information.