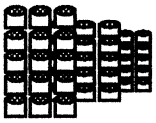
Data General

DG/UX Technical Brief

January 27, 1993

Information About AViiON® Systems from Data General's UNIX® Development Group



In This Issue:

Operator Initiated Failover in the DG/UX™ 5.4.2 Operating System

Contents

Data General's introduction of dual-initiator configuration disk subsystems such as the CLARiiON™ Disk Array Storage System has been well received by customers. Customers need and appreciate the ability to transfer physical disks to a secondary system when a primary system fails.

Customers with dual-initiator configurations have asked:

- How do I transfer physical disks and applications from a failed host to a secondary host as quickly and efficiently as possible?
- How do I design applications to use the physical disk failover functionality?
- What safeguards does the physical disk failover software provide?

To help customers better use the dual-initiator configuration disk subsystem features, Data General has developed "Operator Initiated Failover" software (OIF). This brief describes how the OIF software helps users get the most from their dual-initiator configuration investment. The following topics are discussed:

- An overview of the dual-initiator configuration and its advantages
- The software designed to help customers manage these complex systems
- The hardware and software requirements necessary to ensure a consistent and reliable physical disk failover
- Examples of Operator Initiated Failovers
- How to write your own "watch-dog" processes to automate the failover process

Produced on a Data General AViiON AV4000 with FrameMaker® 3.1X.
012-004186-01

Terminology

Here are some terms that are used in this technical brief.

Array

A collection of one or more of disk modules and one or more SCSI busses that participate in a Redundant Array of Inexpensive Disks (RAID) redundancy scheme.

Disk module (or spindle)

A self contained disk-drive unit—as opposed to the generic term "disk," which could refer to a logical disk or a physical disk.

Dual-initiator configuration

A configuration in which a physical disk is connected to a SCSI bus that can have two initiators.

Failover

The transfer of one or more dual-initiator configuration disk modules and zero or more applications from one machine to another machine sharing the dual-initiator configuration disk module.

Initiator

A SCSI device that has the capability to initiate operations with a SCSI target, such as a host bus adapter.

Logical disk

A software abstraction that enables the DG/UX operating system to manage files the same way, regardless of how the files are stored physically. Logical disks are built on physical disks and a logical disk can use pieces from as many as 32 physical disks.

Physical disk

What the operating system recognizes as a single disk. A physical disk can be a single disk module or a group of disk modules in a CLARiiON Disk Array Storage System.

RAID

Redundant Array of Inexpensive Disks. RAID technology provides redundant disk resources. RAID level 5 (RAID 5) distributes user and parity data among all of the disk modules in an array.

Target

A SCSI device, typically a disk or tape drive, which can be selected as the target of a given SCSI bus operation.

SwitchOver

The use of the Operator Initiated Failover software to transfer a physical disk while both systems are running.

Dual-Initiator Configuration Disk Subsystems

The CLARiiON Disk Array Storage System (CLARiiON disk array) is a dual-initiator configuration disk subsystem that provides SCSI-2 interfaces. This means that you can connect a disk array to two different AViiON computer systems. The systems to which you can connect a CLARiiON disk array must be capable of supporting the CLARiiON disk array I/O processor and they must be server machines (Figure 1).

The CLARiiON disk array is housed in its own cabinet and can support up to 20 disk modules. This configuration of two servers and a CLARiiON disk array provides enormous flexibility as far as distributing disks between the two servers.

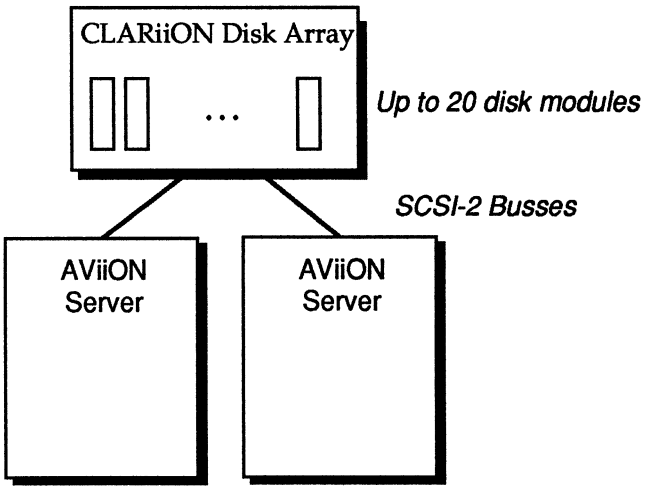


Figure 1 CLARiiON Disk Array Configuration

Dual-initiator configuration physical disks support the fast exchange of physical disks between systems.

When a physical disk is part of a dual-initiator configuration, only one host is said to "own" the physical disk. The machine that has the physical disk open is the owner of the physical disk. Disk devices are opened when the physical disk module is registered, usually through diskman(1M). Only the owner of the physical disk can access the contents of the physical disk, reading or writing directly to the logical disks, or mounting file systems on the logical disks.

Dual-initiator configuration physical disks enable the system administrator to transfer the physical disk from one system to another. This configuration gives the administrator a useful tool to reduce the time that data is inaccessible when one system "crashes," is in need of maintenance, or is upgraded. When a problem arises on one system, the administrator can simply transfer the physical disk to the other system and restart the applications that were running on the failed disk.

Hardware/Software Requirements

The OIF software was released for general distribution in DG/UX 5.4.2. Here are the requirements for using the OIF software:

- DG/UX 5.4.2 or later revisions
- A TCP/IP communications link between each host in the dual-initiator configuration
- Dual-initiator configuration CLARiiON disk array (Figure 2)

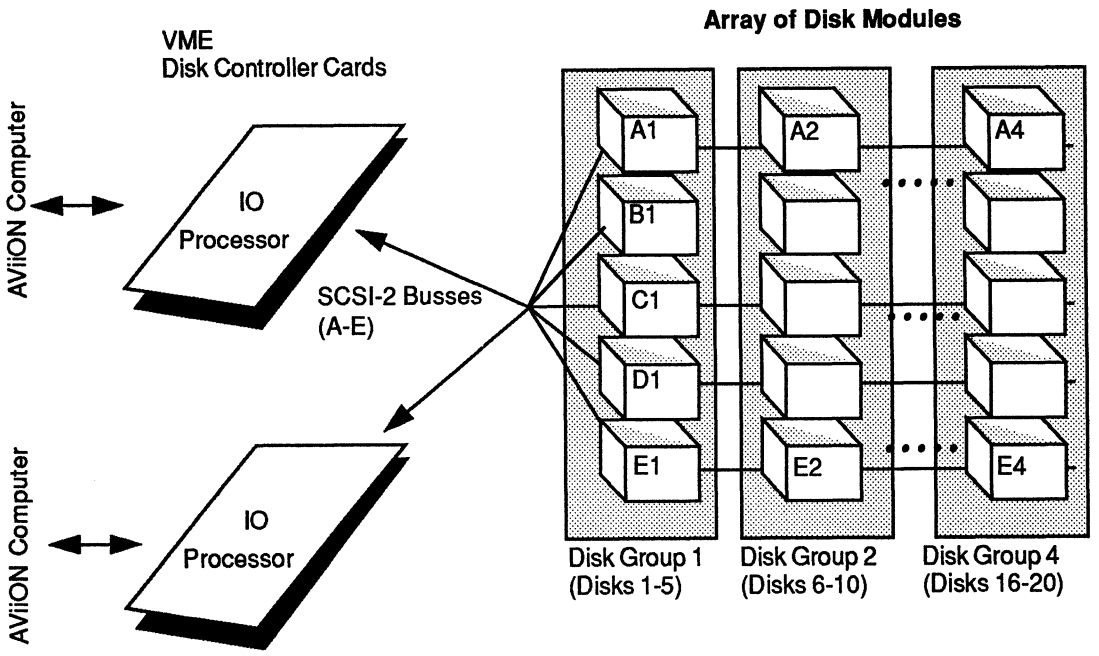


Figure 2 A High Availability Disk Dual-Initiator Configuration

Operator Initiated Failover

The OIF software automates the transfer of ownership of physical disks and ensures that transfers are complete.

The OIF software is designed not only to automate the transfer of ownership of physical disks from one host to another, but to ensure that a transfer is complete. When a system fails, the operator would be displeased to discover that a physical disk that had been transferred contained a logical disk piece on a disk not transferred. This would result in an important file system or application being unable to restart on the secondary system because all of the logical disk pieces were not present. It is this kind of check, among others, that the OIF software performs.

The OIF software consists of several administrative commands, a failover database, a daemon process for controlling remote operations, and a sysadm interface for ease of use. The software uses many existing DG/UX features such as listen(1M), admfilesystem(1M), fsck(1M), fuser(1M), and others. However, all of these are managed by the OIF software. The administrator need only know how to use the OIF software to successfully perform physical disk failover.

OIF Commands

The OIF commands are:

- **admfailoverapplication**—used to edit and maintain the failover application database
- **admfailoverdisk**—used to maintain the failover databases, synchronize failover databases, perform consistency checks, and perform physical disk transfers to and from other hosts
- **admfailovergiveaway**—used to edit and maintain the failover giveaway database
- **admfailoverhosts** —used to edit and maintain the failover hosts database
- **admfailovertakeaway** —used to edit and maintain the failover takeaway database

OIF Operation

The OIF software uses a simple give-and-take scheme. The system that currently owns the physical disk, and has set up the failover databases can give the physical disk away to the other host. The other host in the dual-initiated configuration can (once synchronized) take the physical disks it does not own, but is set up to manage.

To set up the physical disks to be failed over, the operator performs the *add* operation of the admfailoverdisk command on the system that currently owns the physical disk(s).

The *add* operation can be performed through sysadm (Device->Disk->Failover->Add) or by using the command line interface. The *add* operation requires the operator to enter the following information:

- The physical disk specification of the disk to set up, as it appears on this host. This specification is used internally to ensure that the disk is registered, to read its contents, and to help identify failover database entries.
- The physical disk specification of the disk to set up, as it appears on the other (secondary) host in the dual-initiator configuration. This specification is required to properly identify and access the disk on the other host. Depending on how the systems are set up, the remote specification may or may not be the same as the local specification. If for example, the CLARiiON disk array I/O processors are jumpered differently on each machine (jumpered as id 0 on hostA and id 1 on hostB, a physical disk that is referred to as "sd(dgsc(0),0,0)" on hostA would be referred to as "sd(dgsc(0),1,0))" on hostB. This information is also used to identify entries in the failover databases. All failover database entries are tagged with this specification so that when a system fails, the information can be read from the failover databases and not read from the physical disk(s).
- The name of the other host. This information will be used to establish communications, identify failover database entries, and identify this host's failover partner.
- Optionally, a flag specification indicating that a *sync* operation should be performed upon successful completion of the add operation. The sync operation is how the giveaway entries in the current host's failover database become takeaway entries in the other host's failover database. The sync operation also transports failover application database entries from the current host to the other host for addition to its failover application database.
- Optionally, a pathname to an application start-up script that will be executed when the physical disk is failed over. The script should be written so that it does not fail or return a status to be used elsewhere. This script is executed by the OIF software and any return status is ignored.

Once this information is specified, the OIF software will perform the tasks shown in Figure 3.

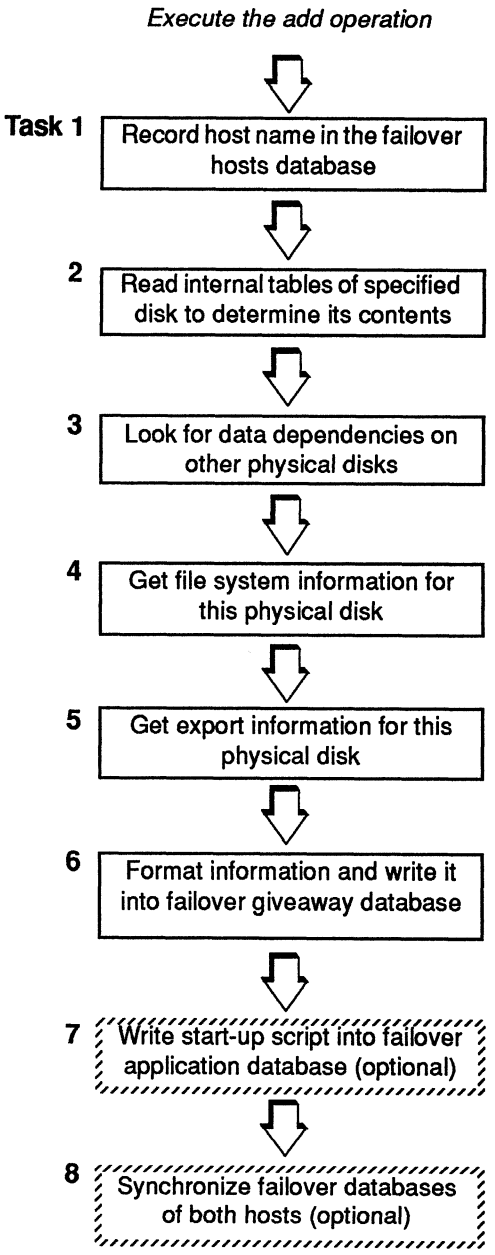


Figure 3 Tasks Performed by the OIF Software

These tasks, performed by the *add* operation, are designed to automate and ensure consistency in the failover process. Looking more closely at these tasks shows how this is achieved.

Task 1

The first task is to make an entry into the failover host's database. This database contains information about the other hosts in the dual-initiator configuration. This information is used to set up communications to the host and to indicate the current synchronization status. If the host has not been entered into the failover hosts database, an entry will be made with a synchronization status indicating that a *sync* operation needs to be performed before failover of a physical disk can be performed. If the failover host's entry already exists (either from a previous *add* operation, or because it was added with the admfailoverhosts command) the synchronization status will be set to indicate a *sync* operation is needed.

The two hosts communicate via TCP/IP. The communications are monitored and controlled by the listen(1M) portmonitor. When the "failover" service is requested by a client, a process called *failoverd* is created on the specified host to service the client requests. The TCP/IP set up for the failover portmonitor and portservice is performed automatically during the DG/UX 5.4.2 upgrade.

Tasks 2 - 6

These tasks ensure that the operator is setting up a complete failover configuration. By reading the contents of the physical disks, the OIF software can find all the logical disks on the physical disk. This is very important in answering the following questions:

- Is this the only piece of the logical disk?
- If not, what piece number(s) is (are) missing, and are they present on this physical disk?
- Is this logical disk a member of a DG/UX software mirror?
- If so, what is the mirror name, how many pieces does it have, are any of these pieces on this physical disk?

Inconsistencies occur when a multi-piece logical disk (or mirror) has a piece on a physical disk that has not yet been added to the failover databases. If there are inconsistencies, warnings are displayed from the *add* operation informing the operator to *add* the physical disks that contain the necessary pieces.

With the information gathered from the physical disk tables, the file system tables are then searched. The file system tables (fstab and exports files) are searched to gather any file system information related to the logical disks found on this physical disk. This information is used to format complete description records for each logical disk that are written to the failover giveaway database. This information is sufficient to issue fuser, fsck, or admfilesystem commands to clear processes from logical disks, and to check, add, mount, and export file systems. Use of this information is further discussed in the "Failover Examples" section.

Entries for logical disks that do not contain file systems or do not export their file systems have the token NONE in fields pertaining to file system information. Entries for logical disks that contain file systems have all of the file system information assigned to the piece 1 logical disk entry.

Task 7

If an application start-up script pathname is specified, it is added to the failover application database. The application scripts listed in the failover application database are executed when the physical disk is failed over.

Task 8

If the synchronize flag is specified, then upon successful completion of the *add* operation, a *sync* operation is performed. The failover databases are checked for consistency and if consistent, are synchronized with the specified host.

Once the failover databases of the two hosts are synchronized, physical disk failover can be performed. Both systems are prepared to transfer the physical disk with either the *give* operation (for the host that currently owns the disk) or the *take* operation (for the system that does not own the disk).

Note – The failover databases contain information about the physical disks at the time they are added to the failover databases. If you change the layout of the physical disk, or information about file systems on these physical disks, you must update the failover databases. These configuration changes are not automatically detected and updated.

Remote-Mounted File Systems

To reduce restart time and errors, pre-mount remote file systems on both hosts in the dual-initiator configuration.

The OIF software and failover databases are used to manage information about file systems residing on the disks in the dual-initiator configuration. Should the application require or use file systems that are remote mounted from other hosts (i.e., NFS mounted from hosts not in the dual-initiator configuration) those file systems should be pre-mounted on both hosts in the dual-initiator configuration. Doing so reduces the time to restart the application and reduces the risk of error. The application may not be able to restart if the host exporting the file system has not granted the backup host permission to mount the file system. A problem such as this would be discovered and corrected before system failure and subsequent physical disk failover occurred.

You should avoid cross mounting file systems between the two hosts in a dual-initiator configuration. This means that a file system that is mounted local on hostA should not be NFS mounted on hostB if the disk it resides on can be failed over to hostB. The admfailoverdisk command will attempt to unmount the NFS file system and make it a local file system, but the results are not guaranteed. A user accessing a file in the NFS hierarchy may cause the local mount to fail.

If this occurs, the administrator will have to determine which process is preventing the mount from succeeding, and terminate it. The file system will then have to be mounted by the administrator.

Failover Databases

The failover databases are in a new directory called /etc/failover. This directory contains four files.

application	containing entries for application start-up scripts to be executed when a physical disk is brought on line
giveaway	containing entries describing physical disks this system can give away to another host
hosts	containing entries describing hosts in dual-initiator configurations with this host
takeaway	containing entries describing physical disks this system can take away from another host

Application

The failover application database stores the pathname to start-up scripts that are executed when a physical disk is failed over. There may be any number of scripts to execute when a physical disk is failed over.

Additional failover application database entries can be made by using the admfailoverapplication command. After all of the physical disks are brought on line (registered, file systems checked and mounted) the application database is searched to see if there are any applications to start. The scripts are executed one after another in the order they are found in the database. If there are dependencies on application start-up scripts, they should be coded into one script that controls the dependencies.

Note – When a physical disk is failed over to a secondary system, the secondary system will have a different CPU ID than the primary system. Applications that validate against the CPU ID, such as some applications that use license managers, may not start up on the secondary system. You should discuss this issue with your software distributor, who may be able to provide a "backup" CPU ID or a floating license.

Giveaway

The failover giveaway database contains descriptive information about the physical disks that this host can give away (or have taken by another host). The entries in this file correspond to any and all file system information that was gathered at the time the disk was added. This information includes the name of the logical disk that is used by fuser to ensure no process is accessing the logical disk (directly or via file system). The information also includes the name of the file system to unexport, unmount, and delete during a *give* operation, or add, mount, and export during a *take* operation.

Hosts

The failover host's database contains information about the other hosts in the dual-initiator configuration. This information includes the name of the host, the communication path to that host (currently "network" is the only supported path), and the current synchronization status. The synchronization status is important because it determines whether a *give* or *take* operation can be performed.

Takeaway

The failover takeaway database entries have the same record formats and level of description as the giveaway database. The difference is that the takeaway database describes physical disks that are owned by the other host in the dual-initiator configuration. These entries are created and maintained by the admfailoverdisk *sync* operation. When a sync is performed, giveaway database entries are copied from the host that owns the physical disk, to the takeaway database of the other specified host in the dual-initiator configuration. The hostname field is replaced with the name of the host that initiated the *sync* operation, and the diskname fields are reversed before writing the entries to the takeaway database of the other host.

Physical Disk SwitchOver

The OIF software is designed to be useful in non-system failure situations. The most obvious use of OIF software is if one system crashes, panics, or "hangs." The secondary system can then take control of the physical disks and restart the applications, minimizing application downtime. However, there are other situations in which the OIF software can be used to SwitchOver the disks to another system, such as:

- load balancing
- system repair
- system upgrades

Load Balancing

Should applications be identified as causing system performance to degrade to an intolerable level, the OIF software can be used to transfer the physical disks and applications to a secondary system. The OIF software can transfer physical disks with both systems up and running in production mode.

System Repair

It is possible that a problem in the root file system could prevent the system from being booted. With the OIF software, the system disk (provided it is on a dual-initiator configuration disk subsystem) can be mounted on the secondary system as a non-root file system, so that it can be examined and repaired. This usage does require careful planning and naming of logical disks when the systems are built. In order to access the root and usr logical disks, they should be named something like rootA and usrA for hostA, and rootB and usrB for hostB. This enables the logical disks to be accessible when the disk is registered on the secondary host.

The file system information in the takeaway database of the remote host needs to be modified (using the admfailovertakeaway command) so that when the disk is taken the root and usr file systems can be mounted. The mount points should be something other than / and /usr.

When the file systems are repaired, the disk can be released using the -S switch on the admfailoverdisk *give* operation.

System Upgrades

The OIF software can be used to transfer physical disks and applications to a secondary system to keep the application processing while the primary system is upgraded with new software. When used in conjunction with a CLARiiON disk array, users can maintain multiple environments. This enables transfer of applications and disks while older or newer operating system releases are booted.

All of these scenarios require careful planning on the part of system administrators. Applications and users should be balanced across physical disks so that transferring one application and 10 users does not force the transfer of all applications and all users. Additionally, accessing system logical disks and file systems requires unique logical disk names for root, usr, and swap for each host.

Failover Examples

The following examples illustrate the set up and usage of Operator Initiated Failover. The first example illustrates its usage in the event of a system crash. The second is a follow-up example showing the transfer of disks back after the failed primary host is rebooted.

For these examples, assume the following configuration information:

- Two AViiON AV6225 (rack-mount dual processor) servers named hostA and hostB
- Both servers are running DG/UX 5.4.2
- There is a local area network to which both servers are connected
- Each server has a CLARiiON disk array I/O processor in the first position
- There is one CLARiiON disk array connected to the two servers
- The CLARiiON disk array has two RAID-5 disk arrays that will contain the application and are to be failed over
- HostA currently owns the RAID-5 arrays
- The first array "sd(dgsc(0,7),1,0)" will be used by the "Acme" Database Manager performing raw I/O on a logical disk spanning the entire array
- The Acme Database Manager is started with a script called acmeup that is in /usr/bin
- The second array "sd(dgsc(0,7),2,0)" will contain three file systems, all of which are set up as fast recovery file systems, one will not be exported, the other two will be exported without restrictions

Example 1—System Crash

Preparation

Set up the arrays on hostA (i.e., use gridman to bind disk modules into arrays and use sysadm to create logical disks and file systems).

Failover Set Up

1) Select the Sysadm->Device->Disk->Failover->Add option on hostA.

   This operation guides you through the required and optional information for the *add* operation of the admfailoverdisk command.

2) Set up the "sd(dgsc(0,7),1,0)" disk. After filling in all of the queries, the following admfailoverdisk command is issued:

```
admfailoverdisk -o add -h hostB -r "sd(dgsc(0),1,0)" -a
/usr/bin/acmeup "sd(dgsc(0,7),1,0)"
```

   The admfailoverdisk command performs the following:

   a. confirms that hostB is a valid host (by using the admhost(1M) command to check for its existence in either the local or NIS (YP) host's database).

   b. confirms that "sd(dgsc(0,7),1,0)" is registered on this host. If the hostname does not exist or the disk is not registered, an error is returned. Because the CLARiiON disk array IOPs are in the same position on each host, the physical disk specifications of the disks we are using are the same on both hosts.

   c. looks for an entry for hostB in the failover host's database on hostA. There isn't one, so it adds one with a synchronization status indicating that a *sync* operation is needed before a *give* or *take* operation can be performed.

   d. gathers information from the internal tables on the physical disk. The physical disk "sd(dgsc(0,7),1,0)" has one logical disk, and no file system information (fstab and exports) is found. An entry is formatted and written to the failover giveaway database.

   e. performs consistency checks and no inconsistencies are found.

   f. processes the -a switch. A failover application database entry is formed indicating that the script "/usr/bin/acmeup" is to be executed when "sd(dgsc(0,7),1,0)" is failed over. This entry is then written to the failover application database.

3) Select the Sysadm->Device->Disk->Failover->Add option on hostA again, to add "sd(dgsc(0,7),2,0)." Since this is the last disk to *add*, select the -s synchronize database option. The resulting command line is:

```
admfailoverdisk -o add -h hostB -r "sd(dgsc(0),2,0)" -s
"sd(dgsc(0,7),2,0)"
```

(See the inset below for information about the "-s" synchronize database option.)

This command performs the same sequence of events as the command in Step 2; it confirms that the hostname and local disk specification are valid, checks the failover host's database, reads the disk, and gathers file system information. This time, file system information is found, so it is formatted and written to the failover giveaway database.

About The "-s" Synchronize Database Option

For the operation in Step 3, we selected the synchronize database option to the admfailoverdisk command. The equivalent command line for this option is:

```
admfailoverdisk -o sync hostB
```

Several things happen in synchronizing failover databases. The admfailoverdisk command becomes a client of a failoverd process on hostB by requesting and connecting to the "failover" service on hostB. A handshake message is then sent to the failoverd daemon on hostB to verify that communications are possible. The failoverd process receives the message, sends a response back to the client (admfailoverdisk command on hostA) and waits for further instructions from the client.

The admfailoverdisk client then sends a message indicating that a *sync* operation is to be performed. The failoverd process acknowledges the request and prepares to receive the necessary files. The admfailoverdisk client then begins reading and sending records from the failoverd giveaway database on hostA to the failover on hostB. The records are received by the failoverd process and written to a temporary file. When the admfailoverdisk client is done sending failover giveaway database entries, a message indicating this is sent to the failoverd process on hostB. The failoverd process then prepares to receive the failover application database from hostA. These records are written to another temporary file.

When both files have been transported to hostB, the admfailoverdisk client on hostA informs the failoverd process on hostB that it can perform its part of the synchronization. The server portion of the synchronization is to gather the records from the temporary files that pertain to hostB, and add them to the appropriate files. Application entries go into the application database, and giveaway entries go into the takeaway database. When this is done, the temporary files are deleted, and the failover host's database entry on hostB for hostA is modified so that its synchronization status indicates that the databases are "in sync."

4) Update the failover host's database entry for hostB on hostA to indicate that the databases are "in sync."

Starting the Application

1) Start the application on hostA.

   HostA panics. The users get no response on their terminals and inform the system administrator.

2) As system administrator, select the Sysadm->Device->Disk->Failover->Take option from hostB.

   This option queries for the name of the host from which to take physical disks. Based on this name, select from a list of possible disks to take. This list is built by reading the failover takeaway database and displaying entries that apply to the selected host.

   Because hostA has panicked, you must select the use trespass option. The trespass option causes the failover software to unconditionally take control of the specified physical disks. This option should be used only when you are sure the other system is unreachable.

3) Depending on whether you selected the disks individually from the list, or selected the "all" option, one of the following commands is executed:

```
admfailoverdisk -o take -h hostA -T "sd(dgsc(0),1,0)"
"sd(dgsc(0),2,0)"
```

   or

```
admfailoverdisk -o take -h hostA -T all
```

   The admfailoverdisk command performs the following:

   a. confirms the hostname by verifying that it is listed in the local or NIS (YP) hosts databases, and that it exists in the failover hosts database with a synchronization status that indicates the *take* operation is to be permitted.

   b. builds a list of disks to search for in its failover takeaway database. For each disk found, the DSKIOC_TRESPASS ioctl system call is issued before attempting to register the physical disk. While in a panic state, hostA's IOP continues to own the physical disks. The DSKIOC_TRESPASS ioctl enables hostB to unconditionally take control of the physical disks.

c. examines the failover takeaway entries for file system information (once all of the physical disks are registered).

   For "sd(dgsc(0),1,0)" there are no file systems. For "sd(dgsc(0),2,0)" there are file systems to bring on line. The first file system operation is to run fsck(1M) on the file system. The fsck utility is invoked with the -l (fast recovery), -x (examine before checking), and -y (automatically fix inconsistencies) options. Since the file systems were set up as fast recovery file systems, the fsck is done in about 5 seconds. Had the file systems not been fast recovery, the fsck time would have been considerably longer.

d. uses the failover takeaway entry information to format an admfilesystem(1M) command that adds an entry to the /etc/fstab and /etc/exports (if required) files, mounts the file system, and exports the file system (if required).

e. moves the entries in the failover takeaway database for the disks that were just brought on line into the failover giveaway database. Since hostB now owns the physical disks it can now *give* them away.

f. executes the application scripts. The failover application database is searched for application scripts that apply to hostA and the two physical disks just brought on line. In this example there is only one script to execute. This is the /usr/bin/acmeup script to bring up the Acme Database Manager.

4) Reboot hostA. Note that the rc.failover script executes, causing the failover database on hostA to be cleaned up. This involves listing the disks currently registered on hostA, and then looking through the giveaway database on hostA for entries corresponding to disks not in this list. These disks are the ones that were failed over, and their associated entries are moved to the failover takeaway database on hostA.

   The physical disks are *not* automatically moved back to hostA when hostA is brought back on line. If you want the physical disks transferred back to hostA then you must do so with the load balancing scenario described in the next example.

Example 2—Load Balancing (SwitchOver)

Preparation

No additional set up or intervention is needed to perform this example right after a system crash. The physical disks are on hostB and the failover databases on hostA were cleaned up when hostA was rebooted. The system informs you that the disks must be moved back to hostA now.

Load Balancing

1) Broadcast a log off message to users.

   The shutdown procedures for the Acme Database Manager should also be executed to eliminate the need for extensive database recovery time.

2) Initiate the load balance scenario from either hostA by selecting the Sysadm->Device->Disk->Failover->Take option, or from hostB by selecting the Sysadm->Device->Disk->Failover->Give option.

   Depending on which option is chosen and assuming you selected "all" physical disks, one of the following command lines is executed:

   ```
   admfailoverdisk -o take -h hostB all
   ```

   or . . .

   ```
   admfailoverdisk -o give -h hostA all
   ```

   Both commands achieve the same result of moving the physical disk from hostB to hostA. Since the *take* operation has already been described, the *give* operation is described here.

   a. The admfailoverdisk command validates the hostname and confirms that the specified physical disks are registered. The synchronization status of the failover hosts database are checked to ensure that both hosts are "in sync."

   b. The admfailoverdisk command then becomes a client of the failoverd process on hostA. A handshake is performed and the admfailoverdisk client sends a message to hostA asking it to verify that it is "in sync" with hostB. The giveaway entries on hostB are then examined and checked. This is to ensure that there are giveaway entries for all logical disk pieces.

   c. The local physical disks are now ready to be "taken down." All of the giveaway entries for the physical disks to be moved are gathered. To ensure that the physical disk can be deregistered, all processes accessing any of the logical disks need to be terminated. For each giveaway entry with a logical disk piece number of one,

the admfailoverdisk command runs fuser(1M). The -k switch is specified to terminate any process found accessing the logical disk. This ensures that the file systems can be unmounted and the physical disks can be deregistered.

d. After the logical disks are cleared, the file systems are unexported, unmounted, and deleted. The fstab and exports entries are deleted to avoid errors if attempts to mount non-existent file systems are made (e.g., mount -a or run level changes).

e. The physical disks are deregistered from hostB.

f. The admfailoverdisk client on hostB sends a message to the failoverd on hostA informing it to perform its part of the load balancing. This involves "bringing up" the physical disks on hostA. The failoverd server executes the appropriate admfailoverdisk command on hostA to bring the disks and applications on line.

g. The admfailoverdisk command on hostA registers the physical disks. The DSKIOC_TRESPASS ioctl is not required because the physical disks were deregistered by hostB. Once the physical disks are registered, the file system operations are performed. The -x switch causes fsck to check the superblock, which will indicate that no check is necessary (since the file system was shut down in an orderly fashion on hostB) and exit.

h. The admfilesystem command lines are constructed to add, mount, and export (if required) the file systems according to the specifications in the failover takeaway database.

i. The takeaway entries on hostA are moved to the giveaway database and the admfailoverdisk command exits. The failoverd process gets the exit status and returns it to the admfailoverdisk command on hostB. The admfailoverdisk command on hostB sees that the operation on hostA was successful, and it moves its giveaway entries to its takeaway database and exits.

j. The applications are executed, restarting the Acme Database Manager with the /usr/bin/acmeup script.

Machine-Initiated Failover Daemons—An Example

Operator Initiated Failover works very well for environments that are monitored frequently. When a failure occurs, the operator can quickly perform the failover and get the application back up and running. But what about those applications that crash at 4am when a system is unattended? What about those failures when the operator has gone to lunch?

Data General is developing a Machine Initiated Failover (MIF) enhancement to the OIF software.

Data General is developing a follow-on to Operator Initiated Failover that further automates the failover process by providing monitors to detect when a system failure occurs and perform the necessary actions. This functionality is called machine initiated failover (MIF) and will be available in a future release of DG/UX.

Until MIF support is available, this section is provided to help administrators develop their own simple monitoring processes that can be used to detect system failure and use the functionality provided by the OIF software.

There are many ways to implement monitoring processes. You could use shell scripts that ping(1M) the primary system every so often, or you could use programs that send messages via TCP/IP back and forth. The direction of the message is a design choice. Should the primary system send "heartbeat" messages at a certain rate (without waiting for a reply) with the secondary system taking action when the heart beat is lost? Or, should the secondary system send messages and wait for a reply (more like a conversation)?

This example uses two programs, **are_you_there.c** and **i_am_here.c**, both written in C. The programs use UDP to access a remote service. These programs are documented on pages 5-6 through 5-11 in the "Programming with TCP/IP on the DG/UX System" manual.

The server program **i_am_here.c**, is written to be put in the background. After you invoke it, the server accepts a message from any client sending one to its port and running in the Internet domain. When the server receives a message, it sends a message to the client process indicating that it is alive.

The client program **are_you_there.c**, binds to any address, sends a message to the server program **i_am_here.c**, and waits for a reply. When it receives a reply, the client program prints a message to the screen indicating that the server is running.

Both the client and server program use the **gethostbyname** routine to return a hostent structure for mapping the hostname supplied on the command line to the host address. If your system is running NFS, this entry may be in a YP database. If your system is running the domain name system, the entry may come from a name server.

Both the client and server programs also use **getservbyname** to request the service specification indicated in **/etc/services**. If your system is running NFS, this entry may be in a YP database.

These two programs can be used in conjunction with the watch_dog.sh shell program. This program takes three arguments:

- the name of the host to monitor
- the interval (in seconds) to wait between sending messages to the server
- the number of times to retry communications before taking action.

The action in this case is the admfailoverdisk -o take command with the -T (trespass) option.

Once you have compiled and linked the C programs, invoke the server program **i_am_here** on the system that has the disks registered (and is set up to give them away). Then on the secondary system (the one set up to take them away) invoke the **watch_dog.sh** script with the proper arguments. It will use the **are_you_there** program to monitor the primary host.

When communications are lost, the shell script executes the admfailoverdisk command to take the disks. This is a simple monitoring process that introduces some of the aspects of machine initiated failover. The script and programs could be changed to attempt to communicate on another LAN or TCP/IP medium should the configuration have multiple LAN connections. The C programs and shell script are listed here.

The are_you_there.c Program

This is a client program that sends a message to a server program to see whether that server is running. If this program receives a response, it prints a message to the terminal indicating the server is running, and exits with a status code of 0.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <signal.h>

extern int errno;
void alarmed();

main(argc, argv)
int    argc;
char   *argv[];
{
 int                s, ns, i, cc, flags, fromlen;
 char               c, *cp, buf[1024], msg[17];
 char               *name = "test_port";
 struct sockaddr_inaddr_base;
 struct sockaddr_in  *addr = &addr_base;
 struct sockaddr_in  from;
 struct sockaddr_in  to;
 struct servent      *sp;
 struct hostent      *hp;
```

```
strcpy(&msg[0], "Hello!");

if (argc != 2) {
      fprintf(stderr, "Usage: are_you_there <hostname>\n");
      exit(1);
}

hp = gethostbyname(argv[1], NULL);
if (hp == NULL) {
      printf("no host named %s\n", argv[1]);
      exit(1);
}

/*
* Service must be in /etc/services.
*/
sp = getservbyname(name, NULL);
if (hp == NULL) {
      printf("no service named %s\n", name);
      exit(1);
}

/*
* We are a client, so we ask to be bound to any port
* ( addr_base.sin_port = 0). We don't care what port we are on,
* only what port the server is on. Bind will assign us a port.
*/

memset((void *)addr, 0, sizeof(struct sockaddr_in));
addr_base.sin_family = hp->h_addrtype;

s = socket(AF_INET, SOCK_DGRAM, 0);
if (s == -1) {
      perror("socket");
      exit(1);
}

if (bind(s, addr, sizeof(struct sockaddr_in)) == -1) {
      perror("bind");
      exit(1);
}

/*
* Next we want to send a message to the server. The theory is that
* he will send a message back and we'll know that he is alive.
*
* In preparation for this, we'll zero out the 'to' struct. This
* makes sure there is no garbage to interfere with our call. We
* set up the "to" struct with the correct family (AF_INET) and
* the port number and address of the server. We got the port
* number from the service name (see call to getservbyname()).
* Both of these parameters came from the command line.
*/

flags = 0;
memset((void *)&to, 0, sizeof(to));
to.sin_family = AF_INET;
to.sin_port = sp->s_port;
to.sin_addr.s_addr = *(int *)hp->h_addr;

cc = sendto(s, msg, sizeof(msg), flags, &to, sizeof(to));
if (cc == -1) {
      perror("sendto");
      exit(1);
}

/*
* Finally, we'll wait for the server to return our call.
*
```

```
 * Once again we start by zeroing out the "from" structure and
 * setting "fromlen" to the length of that struct. The we let
 * recvfrom() do the rest.
 */

 printf("waiting for response from %s\n", hp->h_name);
 signal(SIGALRM, alarmed);
 alarm(5);

 memset((void *)&from, 0, sizeof(from));
 fromlen = sizeof(from);

 cc = recvfrom(s, buf, sizeof(buf), flags, &from, &fromlen);
 if (cc == -1) {
       perror("recvfrom");
       exit(1);
 }

 printf("%s is alive and kicking\n", hp->h_name);
 exit(0);
}

void
alarmed()
{
       printf("Whoops - no answer!\n");
       exit(1);
}
```

The i_am_here.c Program

This is a server program that receives messages from a remote client process and sends a message to that process, informing the client that it is up and running.

```
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/socket.h>
#include <stdio.h>

extern int errno;

main(argc, argv)
int    argc;
char   *argv[];
{
 int                 s, ns, buflen, i, cc, flags, *fromlen, tolen;
 char                c, *cp, buf[1024], *msg;
 char                *name = "test_port";
 char                hostname[14];
 struct sockaddr_in  addr_base;
 struct sockaddr_in  *addr;
 struct sockaddr_in  *from;
 struct sockaddr_in  *to;
 struct servent      *sp;
 struct hostent      *hp;
 struct sockaddr_in  from_container;
 int                 fromlen_container;
```

```
if (argc != 1) {
      fprintf(stderr, "Usage: \ti_am_here\n");
      exit(1);
}

if (gethostname(hostname, sizeof(hostname)) == -1) {
      printf("Can't gethostname(), %s\n", strerror(errno));
      exit(1);
}

hp = gethostbyname(hostname, NULL);
if (hp == NULL) {
      printf("Can't find host %s\n", hostname);
      exit(1);
}

/*
* Service name must be in /etc/services.
*/

sp = getservbyname(name, NULL);
if (hp == NULL) {
      printf("no service named %s\n", name);
      exit(1);
}

addr_base.sin_port = sp->s_port;
addr_base.sin_family = AF_INET;
addr_base.sin_addr.s_addr= INADDR_ANY;

s = socket(AF_INET, SOCK_DGRAM, 0);
if (s == -1) {
      perror("socket");
      exit(1);
}

addr = &addr_base;
if (bind(s, addr, sizeof(struct sockaddr_in)) == -1) {
      perror("bind");
      exit(1);
}

from = &from_container;
fromlen_container = sizeof(from_container);
fromlen = &fromlen_container;

buflen = sizeof(buf);
for (;;) {
      cc = recvfrom(s, buf, buflen, flags, from, fromlen);
      if (cc == -1) {
            perror("recvfrom");
            exit(1);
      }

      cc = sendto(s, buf, buflen, flags, from, *fromlen);
      if (cc == -1) {
            perror("sendto");
            exit(1);
      }
}

}
```

The watch_dog.sh Shell Script

This shell script is used to monitor a specified host and upon loss of communications, invoke the admfailoverdisk command to take control of the disks from that host.

```
#!/bin/sh
#
# To use this Machine Initiated Failover Example:
#
#       1) on the primary host, invoke the i_am_here program and
#        put it in the background.
#
#       2) invoke this script, it requires 3 arguments, they are:
#
#              hostname, the name of the primary system
#
#              interval, in seconds to sleep between sending
#              heartbeat messages
#
#              retries, the number of times to retry the message
#              before performing the failover action
#
HOSTNAME=$1
INTERVAL=$2
RETRIES=$3
COMMAND="/usr/bin/admfailoverdisk -o take"
retries=0
heartbeat_gone=0

while (true)
do
       ./are_you_there $HOSTNAME

       if [ "$?" -eq "1" ]
       then
              if [ "$retries" -eq "$RETRIES" ]
              then
                     if [ "$heartbeat_gone" -eq "1" ]
                     then
                            :
                     else
                            heartbeat_gone=1
                            echo "\texecuting failover command"
                            echo "\t$COMMAND -h $HOSTNAME -T"
                            $COMMAND -h $HOSTNAME -T
                     fi
              else
                     echo "Failure detected retrying"
                     retries=`expr $retries + 1`
              fi
       else
              if [ "$heartbeat_gone" -eq "1" ]
              then
                     echo "heart beat is back"
                     heartbeat_gone=0
                     retries=0
              fi
              sleep $INTERVAL
       fi
done
```

FYI—Device Numbering for Dual-Initiator Configuration SCSI Devices

Since dual-initiator configuration SCSI-2 disk drives such as the CLARiiON disk array all reside on the same SCSI bus, the system administrator must be careful when assigning SCSI ID's to devices. The assignment of SCSI ID's for dual-initiator configuration devices is left up to the system administrator to establish. This section contains suggestions that may help in planning dual-initiator configurations.

When planning and setting up a SCSI-2 dual-initiator configuration, the SCSI ID numbers for each device to be connected to the SCSI bus should be assigned before booting and building the kernels. The system administrator must ensure that no two entities on the bus, be they disk, tape, or host bus adapter, attempt to use the same SCSI ID value.

The format for a SCSI device specification is as follows:

device(adapter-type@device-code(adapter-address, adapter-SCSI-ID),device-SCSI-ID,LUN)

Here is an example of a SCSI device specification:

sd(dgsc@7(FFFB0000,7),0,0)

Here is a description of the example specification:

device	: sd	Indicates a SCSI disk
adapter-type	: dgsc	Data General SCSI adapter
device-code	: 7	
adapter-address	: FFFB0000	
adapter-SCSI-ID	: 7	SCSI-ID of adapter
device-SCSI-ID	: 0	SCSI-ID of device
LUN	: 0	Logical Unit Number

There can be up to eight SCSI-ID's assigned to a SCSI bus. The adapter requires one SCSI-ID, and each device requires a SCSI-ID.

When a dual-initiator configuration of two AV4600's is set up for Operator Initiated Failover, it is recommended that the SCSI-ID's on each bus be assigned or reserved as follows:

SCSI-ID	Device	Description
7	adapter	hostA dgsc adapter
6	adapter	hostB dgsc adapter
5	tape	hostB tape drive
4	tape	hostA tape drive
3	disk	Failover Disk
2	disk	Failover Disk
1	disk	hostB system disk
0	disk	hostA system disk

To change the adapter SCSI-ID's, edit the system.<name> file that is invoked when building a new kernel in Sysadm->System->Kernel->Build. To set the disk and tape SCSI-ID's, adjust the "jumpers" on the disk and tape devices accordingly.

Here is an example for setting up two AV4600 systems that have a physical disk set up for Operator Initiated Failover.

Assumptions

- Each AV4600 has one disk drive for system usage
- Each AV4600 has one tape drive
- Each AV4600 has an dgsc adapter in the first position
- Each AV4600 has DG/UX 5.4.2 pre-installed
- The physical disk to failover is in its own peripheral housing unit (PHU)

Steps

1) Take the first AV4600 out of the box and boot it.

   Follow the installation steps to build and boot a custom kernel.

2) Using Sysadm->System->Kernel->Build, change the disk and tape specifications in the system file. These specifications initially are as follows:

   sd(dgsc(0),0)## SCSI disk 0 on Data General SCSI adapter 0
   st(dgsc(0),4)## SCSI tape 4 on Data General SCSI adapter 0

   Change the specifications to the following, then rebuild and reboot the system:

   sd(dgsc(0,7),0)## SCSI disk 0 on Data General SCSI adapter 0
   st(dgsc(0,7),4)## SCSI tape 4 on Data General SCSI adapter 0

   This change configures the SCSI adapter at SCSI ID 7. Although this is the default value for the adapter SCSI-ID, specifically configuring this is a good idea to ensure things are configured to be the way you want them.

3) Take the second AV4600 out of the box and change the "jumpers" on the disk and tape drive so that the disk drive has SCSI-ID 1 and the tape drive has SCSI-ID 5.

   Boot the second machine.

   Follow the installation steps to build and boot a custom kernel.

4) Using Sysadm->System->Kernel->Build, change the disk and tape specifications in the system file. These specifications initially are as follows:

   sd(dgsc(0),1)## SCSI disk 1 on Data General SCSI adapter 0
   st(dgsc(0),5)## SCSI tape 5 on Data General SCSI adapter 0

   Change the specifications to the following, then rebuild and reboot the system:

   sd(dgsc(0,6),1)## SCSI disk 1 on Data General SCSI adapter 0
   st(dgsc(0,6),5)## SCSI tape 5 on Data General SCSI adapter 0

   This change configures the SCSI adapter for SCSI ID 6. Remember that SCSI-ID 7 (the default for adapter-SCSI-ID's) is used by the adapter on the other host. This change must be done to ensure that all devices on the SCSI bus have a unique SCSI-ID when the two hosts are connected.

5) Change the "jumpers" on the disk drive in the PHU so that it has a SCSI-ID of 2. With both systems powered down, connect the disk in the PHU to both AV4600's. Now power on the PHU and both hosts.

6) The physical disk can now be registered on either host as:

   hostA:sd(dgsc(0,7),2)
   hostB:sd(dgsc(0,6),2)

By taking these steps, all disk and tape drives on hostA are assigned the system adapter-SCSI-ID of 7. All disk and tape drives on hostB are assigned the system adapter-SCSI-ID of 6. These specifications are very important when setting this configuration up for failover.

There is only one SCSI-ID that remains unused on this bus—SCSI-ID 3. This SCSI-ID could be used if another physical disk or CD-ROM device were added to the SCSI bus. Because tape drives in this configuration are "shared," not owned, a third physical disk could be added to the bus by removing one of the tape drives and adding another disk (jumpered to the SCSI-ID of the tape that was removed). Both hosts will be able to access the remaining tape drive on a first come, first served basis.

Additional disks could be added by adding another SCSI bus adapter, with a physically separate SCSI bus connecting both systems. The devices would then have specifications of the form:

sd(dgsc(1,7),0)## SCSI disk 0 on Data General SCSI adapter 1
st(dgsc(1,7),4)## SCSI tape 4 on Data General SCSI adapter 1

Figure 4 shows a dual-initiator SCSI-2 configuration with device specifications.

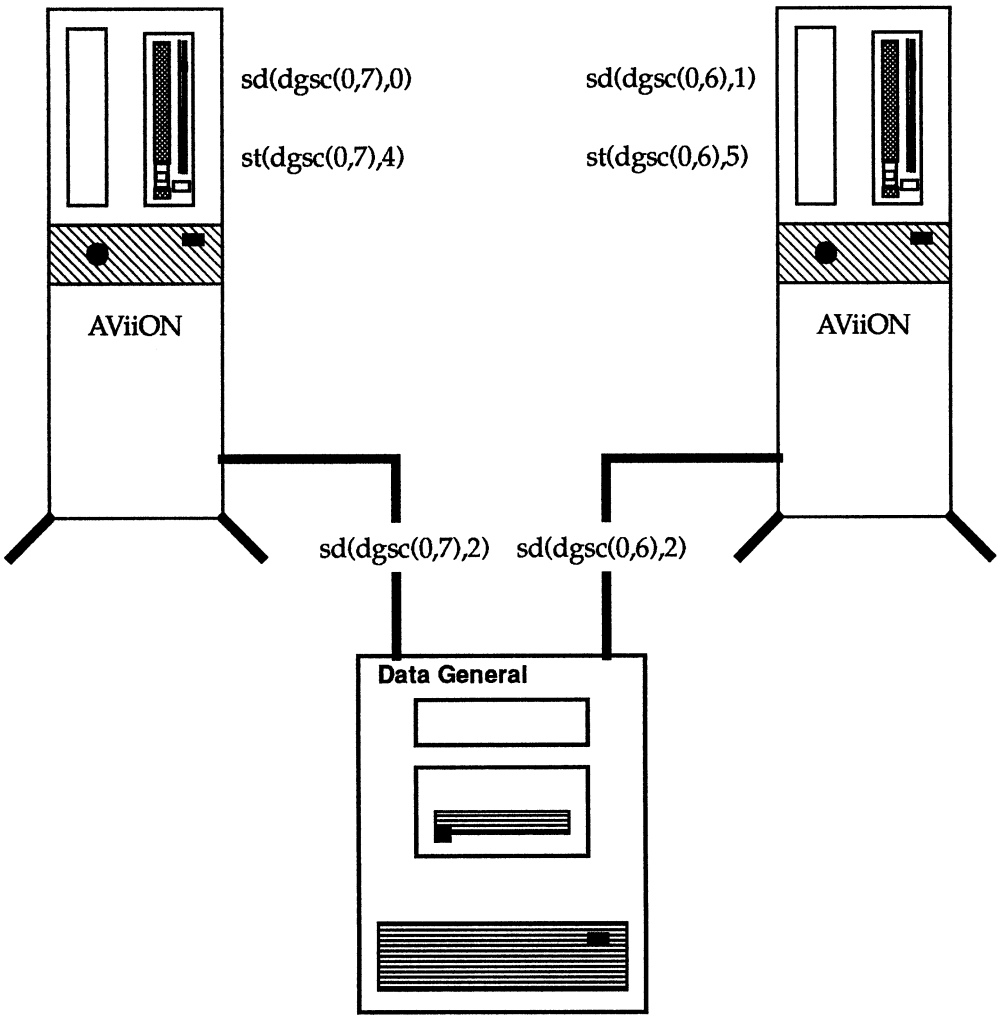


Figure 4 Dual-Initiator Configuration AV4600