Chapter 9

System Administration for Guaranteed-Rate I/O

Guaranteed-rate I/O, or GRIO for short, is a mechanism that enables a user application to reserve part of a system's I/O resources for its exclusive use. For example, it can be used to enable "real-time" retrieval and storage of data streams. GRIO manages the system resources among competing applications, so the actions of new processes do not affect the performance of existing ones. GRIO can read and write only files on a real-time subvolume of an XFS filesystem. To use GRIO, the subsystem *eoe.sw.xfsrt* must be installed.

This chapter explains important guaranteed-rate I/O concepts, describes how to configure a system for GRIO, and provides instructions for creating an XLV logical volume for use with applications that use GRIO.

The major sections in this chapter are:

Note: By default, IRIX supports four GRIO streams (concurrent uses of GRIO). To increase the number of streams to 40, you can purchase the High Performance Guaranteed-Rate I/O—5-40 Streams software option. For even more streams, you can purchase the High Performance Guaranteed-Rate I/O—Unlimited Streams software option. See the *grio* Release Notes for information on purchasing these software options and obtaining the required NetLS licenses. NetLS licenses for GRIO are installed in the standard location, */var/nodelock*.

Guaranteed-Rate I/O Overview

The guaranteed-rate I/O system (GRIO) allows applications to reserve specific I/O bandwidth to and from the filesystem. Applications request guarantees by providing a file descriptor, data rate, duration, and start time. The filesystem calculates the performance available and, if the request is granted, guarantees that the requested level of performance can be met for a given time. This frees programmers from having to predict system I/O performance and is critical for media delivery systems such as video-on-demand.

The GRIO mechanism is designed for use in an environment where many different processes attempt to access scarce I/O resources simultaneously. GRIO provides a way for applications to determine that resources are already fully utilized and attempts to make further use would have a negative performance impact.

If the system is running a single application that needs access to all the system resources, the GRIO mechanism does not need to be used. Since there is no competition, the application gains nothing by reserving the resources before accessing them.

Guarantees can be *hard* or *soft*, a way of expressing the tradeoff between reliability and performance. Hard guarantees deliver the requested performance, but with some possibility of error in the data (due to the requirements for turning off disk drive self-diagnostics and error-correction firmware). Soft guarantees allow the disk drive to retry operations in the event of an error, but this can possibly result in missing the rate guarantee. Hard guarantees place greater restrictions on the system hardware configuration.

Applications negotiate with the system to make a GRIO *reservation*, an agreement by the system to provide a portion of the bandwidth of a system resource for a period of time. The system resources supported by GRIO are files residing within real-time subvolumes of XFS filesystems. A reservation can by transferred to any process and to any file on the filesystem specified in the request.

A GRIO reservation associates a data rate with a filesystem. A data rate is defined as the number of bytes per a fixed period of time (called the *time quantum*). The application receives data from or transmits data to the filesystem starting at a specific time and continuing for a specific period. For example, a reservation could be for 1.2 MB every 1.29 seconds, for the next three hours, to or from the filesystem on */dev/dsk/xlv/video1*. In this example, 1.29 seconds is the time quantum of the reservation.

The application issues a reservation request to the system, which either accepts or rejects the request. If the reservation is accepted, the application then associates the reservation with a particular file. It can begin accessing the file at the reserved time, and it can expect that it will receive the reserved number of bytes per time quantum throughout the time of the reservation. If the system rejects the reservation, it returns the maximum amount of bandwidth that can be reserved for the resource at the specified time. The application can determine if the available bandwidth is sufficient for its needs and issue another reservation request for the lower bandwidth, or it can schedule the reservation for a different time. The GRIO reservation continues until it expires or an explicit **grio_unreserve_bw()** or **grio_remove_request()** library call is made (for more information, see the grio_unremove_bandwidth(3X) and grio_remove_request(3X) reference pages). A GRIO reservation is also removed on the last close of a file currently associated with a reservation.

If a process has a rate guarantee on a file, any reference by that process to that file uses the rate guarantee, even if a different file descriptor is used. However, any other process that accesses the same file does so without a guarantee or must obtain its own guarantee. This is true even when the second process has inherited the file descriptor from the process that obtained the guarantee.

Sharing file descriptors between processes in a process group is supported for files used for GRIO, but the processes do not share the guarantee. If a process inherits an open file descriptor from a parent process and wants to have a rate guarantee on the file, the process must obtain another rate guarantee and associate it with the file descriptor. Sharing file descriptors between processes inhibits the automatic removal of GRIO reservations on the last close of a file associated with a rate reservation.

Four sizes are important to GRIO:

Optimal I/O size

Optimal I/O size is the size of the I/O operations that the system actually issues to the disks. All the disks in the real-time subvolume of an XLV volume must have the same optimal I/O size. Optional I/O sizes of disks in real-time subvolumes of different XLV volumes can differ. For more information see the sections "/etc/grio_config File Format" and "/etc/grio_disks File Format" in this chapter.

XLV volume stripe unit size
: The XLV volume stripe unit size is the amount of data written to a single disk in the stripe. The XLV volume stripe unit size must be an even multiple of the optimal I/O size for the disks in that subvolume. See the section "Introduction to Logical Volumes" in Chapter 6 for more information.

Reservation size (also known as the rate)
: The reservation size is the amount of I/O that an application issues in a single time quantum.

Application I/O size
: The application I/O size is the size of the individual I/O requests that an application issues. An application I/O size that equals the reservation size is recommended, but not required(need to verify). The reservation size must be an even multiple of the application I/O size, and the application I/O size must be an even multiple of the optimal I/O size.

The application is responsible for making sure that all I/O requests are issued within a given time quantum, so that the system can provide the guaranteed data rate.

GRIO Guarantee Types

In addition to specifying the amount and duration of the reservation, the application must specify the type of guarantee desired. There are five different classes of options that need to be determined when obtaining a rate guarantee:

- The rate guarantee can be hard or soft.
- The rate guarantee can be made on a per-file or per-filesystem basis.
- The rate guarantee can be private or shared.
- The rate guarantee can be a fixed rotor, slip rotor, or non-rotor type.
- The rate guarantee can have deadline or real-time scheduling, or it can be nonscheduled.

If the user does not specify any options, the rate guarantee has these options by default: hard, shared, non-rotor options, and deadline scheduling. The per-file or per-filesystem guarantee is determined by the **libgrio** calls to make the reservation: either the **grio_reserve_file()** or **grio_reserve_file_system()** library calls.

Hard and Soft Guarantees

A *hard* guarantee means that the system does everything possible to make sure the application receives the amount of data that has been reserved during each time quantum. It also indicates that the hardware configuration of the system does not interfere with the rate guarantees.

Hard guarantees are possible only when the disks that are used for the real-time subvolume meet the requirements listed in the section "Hardware Configuration Requirements for GRIO" in this chapter.

Because of the disk configuration requirements for hard guarantees (see the section "Hardware Configuration Requirements for GRIO" in this chapter), incorrect data may be returned to the application without an error notification, but the I/O requests return within the guaranteed time. If an application requests a hard guarantee and some part of the system configuration makes the granting of a hard guarantee impossible, the reservation is rejected. The application can then issue a reservation request for a soft guarantee.

A *soft* guarantee means that the system tries to achieve the desired rate, but there may be circumstances beyond its control that prevent the I/O from taking place in a timely manner. For example, if a non-real-time disk is on the same SCSI controller as real-time disks and there is a disk data error on the non-real-time disk, the driver retries the request to recover the data. This could cause the rate guarantee on the real-time disks to be missed due to SCSI bus contention.

Per-File and Per-Filesystem Guarantees

A *per-file* guarantee indicates that the given rate guarantee can be used only on one specific file. When a *per-filesystem* guarantee is obtained, the guarantee can be transferred to any file on the given filesystem.

Private and Shared Guarantees

A *private* guarantee can be used only by the process that obtained the guarantee; it cannot be transferred to another process. A *shared* guarantee can be transferred from one process to another. Shared guarantees are only transferable; they cannot be used by both processes at the same time.

Rotor and Non-Rotor Guarantees

The *rotor* type of guaranteed (either fixed or slip) is also known as a VOD (video on demand) guarantee. It allows more streams to be supported per disk drive, but requires that the application provide careful control of when and where I/O requests are issued.

Rotor guarantees are supported only when using a striped real-time subvolume. When an application accesses a file, the accesses are time multiplexed among the drives in the stripe. An application can only access a single disk during any one time quantum, and consecutive accesses are assumed to be sequential. Therefore, the stripe unit must be set to the number of kilobytes of data that the application needs to access per time quantum. (The stripe unit is set using the *xlv_make* command when volume elements are created.) If the application tries to access data on a different disk when it has a slip rotor guarantee, the system attempts to change the process's rotor slot so that it can access the desired disk. If the application has a fixed rotor guarantee it is suspended until the appropriate time quantum for accessing the given disk.

An application with a fixed rotor reservation that does not access a file sequentially, but rather skips around in the file, has a performance impact. For example, if the real-time subvolume is created on a four-way stripe, it could take as long as four (the size of the volume stripe) times the time quantum for the first I/O request after a seek to complete.

Non-rotor guarantees do not have such restrictions. Applications with non-rotor guarantees normally access the file in entire stripe size units, but can access smaller or larger units without penalty as long as they are within the bounds of the rate guarantee. The accesses to the file do not have to be sequential, but must be on stripe boundaries. If an application tries to access the file more quickly than the guarantee allows, the actions of the system are determined by the type of scheduling guarantee.

An Example Comparing Rotor and Non-Rotor Guarantees

Assume the system has eight disks, each supporting twenty-three 64 KB operations per second. For non-rotor GRIO, if an application needs 512 KB of data each second, the eight disks would be arranged in a eight-way stripe. The stripe unit would be 64 KB. Each application read/write operation would be 512 KB and cause concurrent read/write operations on each disk in the stripe. The application could access any part of the file at any time, provided that the read/write operation always started at a stripe boundary. This would provide 23 process streams with 512 KB of data each second.

With a non-rotor guarantee, the eight drives would be given an optimal I/O size of 512 KB. Each drive can support seven such operations each second. The higher rate (7 x 512 KB versus 23 x 64 KB) is achievable because the larger transfer size does less seeking. Again the drives would be arranged in an eight-way stripe but with a stripe unit of 512 KB. Each drive can support seven 512K streams per second for a total of 8 * 7 = 56 streams. Each of the 56 streams is given a time period (also known as a time "bucket"). There are eight different time periods with seven different processes in each period. Therefore, 8 * 7 = 56 processes are accessing data in a given time unit. At any given second, the processes in a single time period are allowed to access only a single disk.

Using a rotor guarantee more than doubles the number of streams that can be supported with the same number of disks. The tradeoff is that the time tolerances are very stringent. Each stream is required to issue the read/write operations within one time quantum. If the process issues the call too late and real-time scheduling is used, the request blocks until the next time period for that process on the disk. In this example, this could mean a delay of up to eight seconds. In order to receive the rate guarantee, the application must access the file sequentially. The time periods move sequentially down the stripe allowing each process to access the next 512 KB of the file.

Real-Time Scheduling, Deadline Scheduling, and Nonscheduled Reservations

Three types of reservation scheduling are possible: *real-time* scheduling, *deadline* scheduling, and *non-scheduled* reservations.

Real-time scheduling means that an application receives a fixed amount of data in a fixed length of time. The data can be returned at any time during the time quantum. This type of reservation is used by applications that do only a small amount of buffering. If the application requests more data than its rate guarantee, the system suspends the application until it falls within the guaranteed bandwidth.

Deadline scheduling means that an application receives a minimum amount of data in a fixed length of time. Such guarantees are used by applications that have a large amount of buffer space. The application requests I/O at a rate at least as fast as the rate guarantee and is suspended only when it is exceeding its rate guarantee and there is no additional device bandwidth available.

Nonscheduled reservations means that the guarantee received by the application is only a reservation of system bandwidth. The system does not enforce the reservation limits and therefore cannot guarantee the I/O rate of any of the guarantees on the system. Nonscheduled reservations should be used with extreme care.

GRIO System Components

Several components make up the GRIO mechanism: a system daemon, support commands, configuration files, and an application library.

The system daemon is *ggd*. It is started from the script */etc/rc2.d/S94grio* when the system is started. It is always started; unlike some other daemons, it is not turned on and off with the *chkconfig* command. A lock file is created in the */tmp* directory to prevent two copies of the daemon from running simultaneously. Requests for rate guarantees are made to the *ggd* daemon. The daemon reads the GRIO configuration files */etc/grio_config* and */etc/grio_disks*.

/etc/grio_config describes the various I/O hardware paths on the system, starting with the system bus and ending with the individual peripherals such as disk and tape drives. It also describes the bandwidth capabilities of each component. The format of this file is described in the section "/etc/grio_config File Format" in this chapter. If you want a soft rate guarantee, you must edit this file. See step 10 in the section "Configuring a System for GRIO" in this chapter for more information.

/etc/grio_disks describes the performance characteristics for the types of disk drives that are supported on the system, including how many I/O operations of each size (64K, 128K, 256K, or 512K bytes) can be executed by each piece of hardware in one second. You can edit the file to add support for new drive types. The format of this file is described in the section "/etc/grio_disks File Format" in this chapter.

The *cfg* command is used to automatically generate an */etc/grio_config* configuration file for a system's configuration. It scans the hardware in the system, the XLV volumes, and the information in the */etc/grio_disks* file so that it can generate a performance tree, which is put into */etc/grio_config*, for use by the *ggd* daemon. This performance tree is based on an optimal I/O size specified as an option to the *cfg* command. A checksum is included at the end of */etc/grio_config* by *cfg*. When the *ggd* daemon reads the configuration information, it validates the checksum. You can also edit */etc/grio_config* to tune the performance characteristics to fit a given application and tell *ggd* to ignore the checksum. See the section "Modifying /etc/grio_config" in this chapter for more information.

The */usr/lib/libgrio.so* libraries contain a collection of routines that enable an application to establish a GRIO session. The library routines are the only way in which an application program can communicate with the *ggd* daemon. The library also includes a library routine that applications can use to check the amount of bandwidth available on a filesystem. This enables them to quickly get an idea of whether or not a particular reservation might be granted—more quickly than actually making the request.

Hardware Configuration Requirements for GRIO

Guaranteed-rate I/O requires the hardware to be configured so that it follows these guidelines:

- Put only real-time subvolume volume elements on a single disk (not log or data subvolume volume elements). This configuration is recommended for soft guarantees and required for hard guarantees.
- The drive firmware in each disk used in the real-time subvolume must have the predictive failure analysis and thermal recalibration features disabled. All disk drives have been shipped from Silicon Graphics this way since March 1994.
- When possible, disks used in the real-time subvolume of an XLV volume should have the RC (read continuous) bit enabled. (The RC bit is a disk drive parameter that is discussed in more detail later in this section.) This allows the disks to perform faster, but at the penalty of occasionally returning incorrect data (without giving an error).
- Disks used in the data and log subvolumes of the XLV logical volume must have their retry mechanisms enabled. The data and log subvolumes contain information critical to the filesystem and cannot afford an occasional disk error.

For GRIO with hard guarantees, these additional hardware configuration requirements must be met:

- Each disk used for hard guarantees must be on a controller whose disks are used exclusively for real-time subvolumes. These controllers cannot have any devices other than disks on their buses. Any other devices could prevent the disk from accessing the SCSI bus in a timely manner and cause the rate to be missed.

- For hard guarantees, the disk drive retry and error correction mechanisms must be disabled for all disks that are part of the real-time subvolume. (Disk drive retry and error correction mechanisms are controlled by drive parameters that are discussed in more detail below.) When the drive does error recovery, its performance degrades and there can be lengthy delays in completing I/O requests. However, when the drive error recovery mechanisms are disabled, occasionally invalid data is returned to the user without an error indication. Because of this, the integrity of data stored on an XLV real-time subvolume is not guaranteed when drive error recovery mechanisms are disabled.

As described in this section, in some situations, disk drive parameters must be altered on some disks used for GRIO. Table 9-1 shows the disk drive parameters that may need to be changed.

Table 9-1 Disk Drive Parameters for GRIO

Parameter	**New Setting**
Auto bad block reallocation (read)	Disabled
Auto bad block reallocation (write)	Disabled
Delay for error recovery (disabling this parameter enables the read continuous (RC) bit)	Disabled

Setting disk drive parameters can be performed on approved disk drive types only. You can use the *fx* command to find out the type of a disk drive. *fx* reports the disk drive type after the controller test on a line that begins with the words “Scsi drive type.” The approved disk drives types whose parameters can be set for real-time operation are shown in Table 9-2.

Table 9-2 Disk Drives Whose Parameters Can Be Changed

Disk Drive Types Approved for Changing Disk Parameters		
SGI	0664N1D	6s61
SGI	0664N1D	4I4I

The procedure for enabling the RC bit and disabling the disk drive retry and error correction mechanisms is described in the section “Disabling Disk Error Recovery” in this chapter.

Configuring a System for GRIO

Caution: The procedure in this section can result in the loss of data if it is not performed properly. It is recommended only for experienced IRIX system administrators.

This section describes how to configure a system for GRIO: create an XLV logical volume with a real-time subvolume, make a filesystem on the volume and mount it, and configure and restart the *ggd* daemon.

1. Choose disk partitions for the XLV logical volume and confirm the hardware configuration as described in the section “Hardware Configuration Requirements for GRIO” in this chapter. This includes modifying the disk drive parameters as described in the section “Disabling Disk Error Recovery” in this chapter.
2. Determine the values of variables used while constructing the XLV logical volume:

vol_name	The name of the volume with a real-time subvolume.
rate	The rate at which applications using this volume access the data. *rate* is the number of bytes per time quantum per stream (the rate) divided by 1K. This information may be available in published information about the applications or from the developers of the applications.
num_disks	The number of disks included in the real-time subvolume of the volume.

stripe_unit	When the real-time disks are striped (required for Video on Demand and recommended otherwise), this is the amount of data written to one disk before writing to the next. It is expressed in 512-byte sectors.
	For non-rotor guarantees:
	`stripe_unit = rate * 1K / (num_disks * 512)`
	For rotor guarantees:
	`stripe_unit = rate * 1K / 512`

extent_size	The filesystem extent size. For non-rotor guarantees: `extent_size = rate * 1K`

For rotor guarantees:

extent_size = *rate* * 1K * *num_disks*

opt_IO_size The optimal I/O size. It is expressed in kilobytes. By default, the possible values for *opt_IO_size* are 64 (64K bytes), 128 (128K bytes), 256 (256K bytes), and 512 (512K bytes). Other values can be added by editing the */etc/grio_disks* file (see the section "/etc/grio_disks File Format" in this chapter for more information).

For non-rotor guarantees, *opt_IO_size* must be an even factor of *stripe_unit*, but not less than 64.

For rotor guarantees *opt_IO_size* must be an even factor of *rate*. Setting *opt_IO_size* equal to *rate* is recommended.

Table 9-3 gives examples for the values of these variables.

Table 9-3 Examples of Values of Variables Used in Constructing an XLV Logical Volume Used for GRIO

Variable	Type of Guarantee	Comment	Example Value
vol_name	any	This name matches the last component of the device name for the volume, /dev/dsk/xlv/*vol_name*	xlv_grio
rate	any	For this example, assume 512 KB per second per stream	512
num_disks	any	For this example, assume 4 disks	4
stripe_unit	non-rotor	512*1K/(4*512)	256
	rotor	512*1K/512	1024
extent_size	non-rotor	512 * 1K	512k
	rotor	512 * 1K * 4	2048k
opt_IO_size	non-rotor	128/1 = 128 or 128/2 = 64 are possible	64
	rotor	Same as *rate*	512

3. Create an *xlv_make* script file that creates the XLV logical volume. (See the section "Creating Volume Objects With xlv_make" in Chapter 7 for more information.) Example 9-1 shows an example script file for a volume.

Example 9-1 Configuration File for a Volume Used for GRIO

```
# Configuration file for logical volume vol_name. In this
# example, data and log subvolumes are partitions 0 and 1 of
# the disk at unit 1 of controller 1. The real-time
# subvolume is partition 0 of the disks at units 1-4 of
# controller 2.
#
vol vol_name
data
plex
ve dks1d1s0
log
plex
ve dks1d1s1
rt
plex
ve -stripe -stripe_unit stripe_unit dks2d1s0 dks2d2s0 dks2d3s0 dks2d4s0
show
end
exit
```

4. Run *xlv_make* to create the volume:

   ```
   # xlv_make script_file
   ```

   script_file is the *xlv_make* script file you created in step 3.

5. Create the filesystem by giving this command:

   ```
   # mkfs -r extsize=extent_size /dev/dsk/xlv/vol_name
   ```

6. To mount the filesystem immediately, give these commands:

   ```
   # mkdir mountdir
   # mount /dev/dsk/xlv/vol_name mountdir
   ```

   mountdir is the full pathname of the directory that is the mount point for the filesystem.

7. To configure the system so that the new filesystem is automatically mounted when the system is booted, add this line to */etc/fstab*:

   ```
   /dev/dsk/xlv/vol_name mountdir xfs rw,raw=/dev/rdsk/xlv/vol_name 0 0
   ```

8. If the file */etc/grio_config* exists, and you see `OPTSZ=65536` for each device and `OPTSZ=524288` (check this) for disks in the real-time subvolume, skip to step 10.

9. Create the file */etc/grio_config* with this command:

   ```
   # cfg -d opt_IO_size
   ```

10. If you want soft rate guarantees, edit */etc/grio_config* and remove this string:

    ```
    RT=1
    ```

    from the lines for disks where software retry is required (see the section "/etc/grio_config File Format" in this chapter for more information).

11. Restart the *ggd* daemon:

    ```
    # /etc/init.d/grio stop
    # /etc/init.d/grio start
    ```

    Now the user application can be started. Files created on the real-time subvolume volume can be accessed using guaranteed-rate I/O.

Additional Procedures for GRIO

The following subsections describe additional special-purpose procedures for configuring disks and GRIO system components.

Disabling Disk Error Recovery

As described in the section "Hardware Configuration Requirements for GRIO" in this chapter, disks in XLV logical volumes used by GRIO applications may have to have their parameters modified.

Caution: Setting disk drive parameters must be performed correctly on approved disk drive types only. Performing the procedure incorrectly, or performing it on an unapproved type of disk drive could severely damage the disk drive. Setting disk drive parameters should be performed only by experienced system administrators.

The procedure for setting disk drive parameters is shown below. In this example all of the parameters shown in Table 9-1 are changed for a disk on controller 131 at drive address 1.

1. Start *fx* in expert mode:

```
# fx -x
fx version 6.2, Oct 10, 1995
```

2. Specify the disk whose parameters you want to change by answering the prompts:

```
fx: "device-name" = (dksc) <Enter>
fx: ctlr# = (0) 131
fx: drive# = (1) 1
fx: lun# = (0)
...opening dksc(131,1,0)


...controller test...OK
```

3. Confirm that the disk drive is one of the approved types listed in Table 9-2 by comparing the next line of output to the table.

```
Scsi drive type == SGI     0664N1D         6s61
----- please choose one (? for help, .. to quit this menu)-----
[exi]t              [d]ebug/             [l]abel/
[b]adblock/         [exe]rcise/          [r]epartition/
```

4. Show the current settings of the disk drive parameters (this command uses the shortcut of separating commands on a series of hierarchical menus with slashes):

```
fx > label/show/parameters

----- current drive parameters-----
Error correction enabled          Enable data transfer on error
Don't report recovered errors     Do delay for error recovery
Don't transfer bad blocks         Error retry attempts          10
Do auto bad block reallocation (read)
Do auto bad block reallocation (write)
Drive readahead  enabled          Drive buffered writes disabled
Drive disable prefetch   65535    Drive minimum prefetch         0
Drive maximum prefetch   65535    Drive prefetch ceiling     65535
Number of cache segments     4
Read buffer ratio        0/256    Write buffer ratio         0/256
Command Tag Queueing disabled


----- please choose one (? for help, .. to quit this menu)-----
[exi]t              [d]ebug/             [l]abel/
[b]adblock/         [exe]rcise/          [r]epartition/
```

The parameters in Table 9-1 correspond to “Do auto bad block reallocation (read),” “Do auto bad block reallocation (write),” and “Do delay for error recovery,” in that order. Each of them is currently enabled.

5. Give the command to start setting disk drive parameters and press **`<Enter>`** until you reach a parameter that you want to change:

```
fx> label/set/parameters
fx/label/set/parameters: Error correction = (enabled) <Enter>
fx/label/set/parameters: Data transfer on error = (enabled) <Enter>
fx/label/set/parameters: Report recovered errors = (disabled) <Enter>
```

6. To change the delay for error recovery parameter to disabled, enter "disable" the prompt:

```
fx/label/set/parameters: Delay for error recovery = (enabled) disable
```

7. Press **`<Enter>`** through other parameters that don't need changing:

```
fx/label/set/parameters: Err retry count = (10) <Enter>
fx/label/set/parameters: Transfer of bad data blocks = (disabled) <Enter>
```

8. To change the auto bad block reallocation parameters, enter "disable" at their prompts:

```
fx/label/set/parameters: Auto bad block reallocation (write) = (enabled) disable
fx/label/set/parameters: Auto bad block reallocation (read) = (enabled) disable
```

9. Press **`<Enter>`** through the rest of the parameters:

```
fx/label/set/parameters: Read ahead caching = (enabled) <Enter>
fx/label/set/parameters: Write buffering = (disabled) <Enter>
fx/label/set/parameters: Drive disable prefetch = (65535) <Enter>
fx/label/set/parameters: Drive minimum prefetch = (0) <Enter>
fx/label/set/parameters: Drive maximum prefetch = (65535) <Enter>
fx/label/set/parameters: Drive prefetch ceiling = (65535) <Enter>
fx/label/set/parameters: Number of cache segments = (4) <Enter>
fx/label/set/parameters: Enable CTQ = (disabled) <Enter>
fx/label/set/parameters: Read buffer ratio = (0/256) <Enter>
fx/label/set/parameters: Write buffer ratio = (0/256) <Enter>
```

10. Confirm that you want to make the changes to the disk drive parameters by entering "yes" to this question and start exiting *fx*:

```
 * * * * * W A R N I N G * * * * *
about to modify drive parameters on disk dksc(131,1,0)! ok? yes

----- please choose one (? for help, .. to quit this menu)-----
[exi]t             [d]ebug/           [l]abel/           [a]uto
[b]adblock/        [exe]rcise/        [r]epartition/     [f]ormat
fx> exit
```

11. Confirm again that you want to make the changes to the disk drive parameters by pressing **<Enter>** in response to this question:

```
label info has changed for disk dksc(131,1,0).  write out changes? (yes) <Enter>
```

Restarting the *ggd* Daemon

After any of the files */etc/grio_disks*, */etc/grio_config*, or */etc/config/ggd.options* are modified, *ggd* must be restarted to make the changes take effect. Give these commands to restart *ggd*:

```
# /etc/init.d/grio stop
# /etc/init.d/grio start
```

When *ggd* is restarted, current rate guarantees are lost.

Modifying */etc/grio_config*

You can edit */etc/grio_config* to tune the performance characteristics to fit a given application. Follow this procedure to make the changes:

1. Using the information in the section "/etc/grio_config File Format" in this chapter, edit */etc/grio_config* as desired.
2. Create or modify the file */etc/config/ggd.options* and add **-d**. This option tells *ggd* to ignore the file checksum in */etc/grio_config*; the checksum is no longer correct because of the editing in step 1. See the section "/etc/config/ggd.options File Format" in this chapter for more information.
3. Restart the *ggd* daemon. See the section "Restarting the ggd Daemon" in this chapter for directions.

Running *ggd* as a Real-time Process

Running *ggd* as a real-time process dedicates one or more CPUs to performing GRIO requests exclusively. Follow this procedure on a multiprocessor system to run *ggd* as a real-time process:

1. Create or modify the file */etc/config/ggd.options* and add **-c** *cpunum* to the file. *cpunum* is the number of a processor to be dedicated to GRIO. This causes the CPU to be marked isolated, restricted to running selected processes, and nonpreemptive. Processes using GRIO should mark their processes as real-time and runable only on CPU *cpunum*. The sysmp(2) reference page explains how to do this.
2. Restart the *ggd* daemon. See the section "Restarting the ggd Daemon" in this chapter for directions.
3. After *ggd* has been restarted, you can confirm that the CPU has been marked by giving this command (*cpunum* is 3 in this example):

   ```
   # mpadmin -s
   processors: 0 1 2 3 4 5 6 7
   unrestricted: 0 1 2 5 6 7
   isolated: 3
   restricted: 3
   preemptive: 0 1 2 4 5 6 7
   clock: 0
   fast clock: 0
   ```
4. To mark an additional CPU for real-time processes after *ggd* has been restarted, give these commands:

   ```
   # mpadmin -rcpunum2
   # mpadmin -Icpunum2
   # mpadmin -Ccpunum2
   ```

GRIO File Formats

The following subsections contain reference information about the contents of the three GRIO configuration files, */etc/grio_config*, */etc/grio_disks*, and */etc/config/ggd.options*.

/etc/grio_config File Format

The */etc/grio_config* file describes the configuration of the system I/O devices. The *cfg* command generates */etc/grio_config*, based on an optimal I/O size specified on the command *cfg* line. *cfg* scans the hardware in the system, the XLV volumes, and the information in the */etc/grio_disks* to create */etc/grio_config*. You can also edit */etc/grio_config* to tune the performance characteristics to fit a given application. Changes to */etc/grio_config* do not take effect until the *ggd* daemon is restarted (see the section "Restarting the ggd Daemon" in this chapter).

The information in */etc/grio_config* is used by the *ggd* daemon to construct a tree that describes the relationships between the components of the I/O system and their bandwidths. In order to grant a rate guarantee on a disk device, the *ggd* daemon checks that each component in the I/O path from the system bus to the disk device has sufficient available bandwidth.

There are two basic types of records in */etc/grio_config*: component records and relationship records. Each record occupies a single line in the file. Component records describe the I/O attributes for a single component in the I/O subsystem. CPU and memory components are described in the file, as well, but do not currently affect the granting or refusal of a rate guarantee.

The format of component records is:

componentname= parameter=value parameter=value . . . (*descriptive text*)

componentname is a text string that identifies a single piece of hardware present in the system. Some *componentname*s are:

SYSTEM	The machine itself. There is always one SYSTEM component.
CPU*n*	A CPU board in slot *n*. It is attached to SYSTEM.
MEM*n*	A memory board in slot *n*. It is attached to SYSTEM.
IOB*n*	An I/O board with *n* as its internal location identifier. It is attached to SYSTEM.
IOA*nm*	An I/O adaptor. It is attached to IOB*n* at location *m*.
CTR*n*	SCSI controller number *n*. It is attached to an I/O adapter.
DSK*n*U*m*	Disk device *m* attached to SCSI controller *n*.

parameter can be one of the following:

OPTSZ	The optimal I/O size of the component
NUM	The number of OPTSZ I/O requests supported by the component each second
SLOT	The backplane slot number where the component is located, if applicable (not used on all systems)
VER	The CPU type of system (for example, IP22, IP19, and so on; not used on all systems)
NUMCPUS	The number of CPUs attached to the component (valid only for CPU components; not used on all systems)
MHZ	The MHz value of the CPU (valid only for CPU components; not used on all systems)
CTLRNUM	The SCSI controller number of the component
UNIT	The drive address of the component
RT	Set to 1 if the disk is in a real-time subvolume (remove this parameter for soft guarantees)
RPOS	Determines the disk’s position in the striped subvolume

The *value* is the integer or text string value assigned to the parameter. The string enclosed in parentheses at the end of the line describes the component.

Some examples of component records taken from */etc/grio_config* on an Indy system are shown below. Each record is a single line, even if it is shown on multiple lines here.

- `SYSTEM= OPTSZ=65536 NUM=5000 (IP22)`

  The *componentname* SYSTEM refers to the system bus. It supports five thousand 64 KB operations per second.

- `CPU= OPTSZ=65536 NUM=5000 SLOT= 0 VER=IP22 NUMCPUS=1 MHZ=100`

  This describes a 100 MHz CPU board in slot 0. It supports five thousand 64 KB operations per second.

- `CTR0= OPTSZ=65536 NUM=100 CTLRNUM=0 (WD33C93B,D)`

  This describes SCSI controller 0. It supports one hundred 64 KB operations per second.

- `DSK0U0= OPTSZ=65536 NUM=23 CTLRNUM=0 UNIT=1 (SGI SEAGATE ST31200N9278)`

  This describes a SCSI disk attached to SCSI controller 0 at drive address 1. It supports twenty-three 64 KB operations per second.

Relationship records describe the relationships between the components in the I/O system. The format of relationship records is:

component`:` *attached_component1* *attached_component2* `...`

These records indicate that if a guarantee is requested on *attached_component1*, the *ggd* daemon must determine if *component* also has the necessary bandwidth available. This is performed recursively until the SYSTEM component is reached.

Some examples of relationship records taken from */etc/grio_config* on an Indy system are:

- `SYSTEM: CPU`

  This describes the CPU board as being attached to the system bus.

- `CTR0: DSK0U1`

  This describes the SCSI disk at drive address 1 being attached to SCSI controller 0.

/etc/grio_disks File Format

The file */etc/grio_disks* contains information that describes I/O bandwidth parameters of the various types of disk drives that can be used on the system.

By default, */etc/grio_disks* contains the parameters for disks supported by Silicon Graphics for optimal I/O sizes of 64K, 128K, 256K, and 512K. Table 9-4 lists these disks. Table 9-5 shows the optimal I/O sizes and the number of optimal I/O size requests each of the disks listed in Table 9-4 can handle in one second.

Table 9-4 Disks in */etc/grio_disks* by Default

Disk ID String
`"SGI     IBM  DFHSS2E    1111"`
`"SGI     SEAGATE ST31200N8640"`
`"SGI     SEAGATE ST31200N9278"`
`"SGI     066N1D          4I4I"`
`"SGI     0064N1D         4I4I"`
`"SGI     0664N1D         4I4I"`
`"SGI     0664N1D         6S61"`
`"SGI     0664N1D         6s61"`
`"SGI     0664N1H         6s61"`
`"IBM OEM 0663E15         eSfS"`
`"IMPRIMIS94601-15        1250"`
`"SEAGATE ST4767          2590"`

Table 9-5 Optimal I/O Sizes and the Number of Requests per Second Supported

Optimal I/O Size	**Number of Requests per Second**
65536	23
131072	16

Table 9-5 (continued) Optimal I/O Sizes and the Number of Requests per Second Supported

Optimal I/O Size	Number of Requests per Second
262144	9
524288	5

To add other disks or to specify a different optimal I/O size, you must add information to the */etc/grio_disks* file. If you modify */etc/grio_disks*, you must rerun the *cfg* command to re-create */etc/grio_config* and then restart the *ggd* daemon for the changes to take effect (see the section "Restarting the ggd Daemon" in this chapter).

The records in */etc/grio_disks* are in these two forms:

`ADD "`*disk id string*`"` *optimal_iosize number_optio_per_second*

`SETSIZE` *device optal_iosize*

If the first field is the keyword ADD, the next field is a 28-character string that is the drive manufacturer's disk ID string. The next field is an integer denoting the optimal I/O size of the device in bytes. The last field is an integer denoting the number of optimal I/O size requests that the disk can satisfy in one second.

Some examples of these records are:

```
ADD     "SGI     SEAGATE ST31200N9278"  64K     23

ADD     "SGI             0064N1D 4I4I"  50K     25
```

If the first field is the keyword SETSIZE, the next field is the pathname of a disk device. The third field is an integer denoting the optimal I/O size to be used on the device.

Normally, the optimal I/O size of a disk device is determined by its stripe unit size. If the disk is not striped or you do not want to use the stripe unit size for the optimal I/O size, you can use the SETSIZE command to tell the *cfg* command how to construct the lines for the GRIO disk in the */etc/grio_config* file.

An example of a SETSIZE record is:

```
SETSIZE /dev/rdsk/dks136d1s0 50K
```

/etc/config/ggd.options File Format

/etc/config/ggd.options contains command-line options for the *ggd* daemon. Options you might include in this file are:

-d	Do not use the checksum at the end of */etc/grio_config*. This is option is required when */etc/grio_config* has been modified to tune performance for an application.
-c *cpunum*	Dedicate CPU *cpunum* to performing GRIO requests exclusively.

If you change this file, you must restart *ggd* to have your changes take effect. See the section "Restarting the ggd Daemon" in this chapter for more information.