# Guaranteed-Rate I/O Version 2 Guide

007–4244–004

# New Features in this Guide

This guide contains the following updates:

- The default for the `ggd2.options` file `-m` and `-s` options is 1 MB. See "`ggd2.options` File" on page 18.
- Minor editorial changes.

# Record of Revision

| Version | Description |
|---|---|
| 001 | February 2004<br>Original publication |
| 002 | March 2006<br>Revision |
| 003 | June 2006<br>Revision for IRIX 6.5.30 |
| 004 | August 2007<br>Revision for CXFS 4.2 |

# Contents

# About This Guide

This publication provides information about GRIO version 2, the second-generation guaranteed-rate I/O product from SGI. GRIO version 2 is supported with CXFS 3.4 or later.

## Related Publications

The following publications contain additional information that may be helpful:

- *CXFS Administration Guide for SGI InfiniteStorage*
- *CXFS MultiOS Client-Only Guide for SGI InfiniteStorage*
- *IRIX Admin: Disks and Filesystems*
- *XVM Volume Manager Administrator's Guide*
- Man pages:

  `ggd2`(1M)
  `grioadmin`(1M)
  `grioqos`(1M)
  `open`(2)
  `grio_avail`(3X)
  `grio_bind`(3X)
  `grio_modify`(3X)
  `grio_release`(3X)
  `grio_reserve`(3X)
  `grio_unbind`(3X)
  `griotab`(4)
  `grio2`(5)

## Obtaining Publications

You can obtain SGI documentation as follows:

- See the SGI Technical Publications Library at http://docs.sgi.com. Various formats are available. This library contains the most recent and most comprehensive set of online books, release notes, man pages, and other information.
- If it is installed on your SGI system, you can use InfoSearch, an online tool that provides a more limited set of online books, release notes, and man pages. With an IRIX system, enter `infosearch` at a command line or select **Help > InfoSearch** from the Toolchest.
- On IRIX systems, you can view release notes by entering either `grelnotes` or `relnotes` at a command line.
- On Linux systems, you can view release notes on your system by accessing the `README.txt` file for the product. This is usually located in the `/usr/share/doc/`*productname* directory, although file locations may vary.
- You can view man pages by typing `man` *title* at a command line.

## Conventions

The following conventions are used throughout this document:

| Convention | Meaning |
|---|---|
| `command` | This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures. |
| *variable* | Italic typeface denotes variable entries and words or concepts being defined. |
| **`user input`** | This bold, fixed-space font denotes literal items that the user enters in interactive sessions. (Output is shown in nonbold, fixed-space font.) |
| [ ] | Brackets enclose optional portions of a command or directive line. |

| | |
|---|---|
| ... | Ellipses indicate that a preceding element can be repeated. |

## Reader Comments

If you have comments about the technical accuracy, content, or organization of this publication, contact SGI. Be sure to include the title and document number of the publication with your comments. (Online, the document number is located in the front matter of the publication. In printed publications, the document number is located at the bottom of each page.)

You can contact SGI in any of the following ways:

- Send e-mail to the following address:

  techpubs@sgi.com

- Contact your customer service representative and ask that an incident be filed in the SGI incident tracking system.

- Send mail to the following address:

  SGI
  Technical Publications
  1140 East Arques Avenue
  Sunnyvale, CA 94085–4602

SGI values your comments and will respond to them promptly.

# Introduction

GRIO version 2 (GRIOv2) is the second-generation guaranteed-rate I/O product from SGI.

**Note:** When it is necessary to distinguish between the previous version (version 1) and the current version (version 2), this guide uses the terms *GRIOv1* and *GRIOv2*. Where the term *GRIO* is used without qualification, it refers to version 2.

## What Does GRIO Do?

GRIO does the following:

- Enables a proportion of a filesystem's I/O resources to be reserved for the exclusive use of a process or compute node
- Ensures that the rate at which a process or node issues I/O does not exceed its reservation and throttles I/O if necessary
- Ensures that the aggregate utilization of a filesystem never exceeds a configurable maximum level, referred to as the *qualified bandwidth*

With a carefully selected and verified qualified bandwidth, you can use GRIO to meet the quality-of-service requirements of demanding I/O workloads where data must be processed without interruption.

GRIO includes the following features:

- Support for CXFS filesystems shared among nodes in a cluster as well as locally attached XFS filesystems
- A simple filesystem-level performance qualification model (rather than the often complex device-qualification model used in GRIOv1)
- A range of tools for monitoring and measuring delivered bandwidth and I/O service time

## Terminology

GRIO uses the following terminology:

- *Quality of service (QoS)* refers to the performance properties of a system service (such as worst-case bandwidth or I/O service time).
- *Qualified bandwidth* is the maximum bandwidth that can be delivered by a filesystem (and the XVM volume on which it resides) in a given configuration under a realistic application workload such that all applications are delivered an adequate QoS.
- *Reservation* is the set of QoS parameters requested by a user application. Reservation requests are forwarded to the `ggd2`(1M) bandwidth management daemon. GRIO supports two kinds of reservations:
  - Explicit *application-level* reservations, which are made by individual applications using the `libgrio2` APIs
  - *Node-level* bandwidth allocations, which are used within GRIO to control all of the I/O flowing from a compute node to a given filesystem.
- *Guarantee* is the assurance made by the system to a user process that it will deliver data from a storage device at the reserved rate.
- *Stream* is the object within the kernel that encodes the reservation's QoS parameters and maintains the necessary scheduling and monitoring state required to fulfill the guarantee.

## GRIOv1 and GRIOv2 Differences

Table 1-1 summarizes the primary differences between GRIOv1 and GRIOv2.

**Table 1-1** Differences Between GRIOv1 and GRIOv2

| | GRIOv1 | GRIOv2 |
|---|---|---|
| Reservation-granting daemon: | `ggd` | `ggd2` |
| Userspace library: | `libgrio` | `libgrio2` |
| Logical volumes: | XLV | XVM |
| Filesystems supported: | Local XFS filesystems only | Local IRIX XFS filesystems and shared CXFS filesystems |
| Multiple-node support: | No | Yes |
| Qualification model: | *Device-level:* the maximum sustainable bandwidth for each **hardware component** in the I/O path is qualified individually. (This includes the storage devices, SCSI and Fibre Channel busses, system interconnects, and so on.) | *Filesystem-level:* the maximum sustainable bandwidth is measured across the entire **filesystem** under a realistic application workload. The qualified bandwidth is stored as follows:<br>• Local IRIX XFS filesystems: in the `/etc/griotab` file<br>• Shared CXFS filesystem: in the cluster database, using the `cxfs_admin`(1M) or `cmgr`(1M) command |
| Monitoring service | Limited administration tools | Comprehensive tools for measuring and monitoring delivered QoS levels, including collection of per-stream performance metrics. |
| Control of non-GRIO-managed I/O | No control | Cluster-wide encapsulation and control. |

When GRIOv2 begins managing a filesystem, every node with access to that filesystem is notified. From that point on, all user and system I/O that does not have an explicit reservation is encapsulated. This means that I/O that is not explicitly managed by a GRIO reservation is automatically associated with a system-managed kernel stream. The `ggd2` daemon allocates otherwise unused bandwidth to these streams, which allows I/O that is not explicitly managed by a GRIO reservation to be processed even when there are active reservations in the system. `ggd2` dynamically adjusts the amount of bandwidth allocated for this purpose based on monitoring of filesystem demand and utilization.

You can use the information provided by the QoS infrastructure to choose the tradeoff between resource utilization and delivered I/O performance that is most appropriate for a given application mix, workload, and production environment.

For more information, see the following:

- Chapter 4, "Administering GRIO" on page 25 and the `grioadmin`(1M) man page
- Chapter 5, "Monitoring Quality of Service" on page 33 and the `grioqos`(5) and `grioqos`(1M) man pages
- The following man pages:

  `ggd2`(1M)
  `griotab`(4)

Although you can have both the GRIOv1 and GRIOv2 subsystems installed on the same machine, only one of them can be active. For more information, see "Starting GRIO on IRIX Servers" on page 22.

## Overview

This guide provides the information you need to administer GRIO. It discusses the following:

- Chapter 2, "How GRIO Works" on page 5
- Chapter 3, "Setting Up GRIO" on page 11
- Chapter 4, "Administering GRIO" on page 25
- Chapter 5, "Monitoring Quality of Service" on page 33
- Chapter 6, "GRIO API Overview" on page 49

# How GRIO Works

This chapter discusses the following:



## Traffic Control

GRIO is a component on the XFS and CXFS I/O path that runs in every node with access to a GRIO-managed filesystem. When active, all I/O on a machine and in the cluster is controlled by the GRIO scheduler.

I/O falls into the following categories:

- *GRIO I/O*: direct (non-buffered) I/O for applications that have made an explicit GRIO reservation
- *Non-GRIO I/O*: all other buffered and system I/O

GRIO ensures that applications with reserved bandwidth receive data at the requested rate, regardless of other I/O activity on the node and elsewhere within the cluster. GRIO will throttle an application if it attempts to use more bandwidth than it has reserved.

## Stream Use and Real-Time File Setup

In order to use an application-level GRIO reservation, a file must be read or written using direct I/O requests. The `open`(2) man page describes the use and buffer alignment restrictions of the direct I/O interface. A GRIO reservation can be made for any regular file within an XFS or CXFS filesystem created on an XVM volume.

Both XFS and CXFS provide a dedicated real-time subvolume that allows filesystem metadata to be separate from user data. To allocate a file on the real-time subvolume of an XFS or CXFS filesystem, you must use the `fcntl`(2) `F_FSSETXATTR` command to set the `XFS_XFLAG_REALTIME` flag. You can only issue this command on a newly created file. It is not possible to mark a file as real-time after non-real-time data blocks have been allocated to it.

## Software Components

GRIO functionality is distributed between the following main components:

- `ggd2` daemon (see "`ggd2` Daemon" on page 6)
- `libgrio2` library, which implements the GRIO userspace API (see Chapter 6, "GRIO API Overview" on page 49)
- The operating system kernel, including the following:
  - Stream management
  - I/O scheduler
  - Cluster integration
  - Messaging

## ggd2 Daemon

The `ggd2`(1M) daemon is a user-level process started at system boot time that manages the I/O bandwidth for a collection of XVM volumes. It does the following:

- Activates and deactivates the GRIO kernel scheduler
- Processes client requests to reserve and release bandwidth

- Tracks bandwidth utilization
- Manages unallocated bandwidth
- Prevents oversubscription
- Enforces GRIO software licenses
- Broadcasts to the relevant kernels the amount of bandwidth per filesystem that may be used for non-GRIO I/O

## Qualified Bandwidth

The qualified bandwidth for a filesystem is the maximum I/O load that it can sustain while still satisfying requirements for delivered bandwidth and I/O service time.

You must determine a specific qualified bandwidth for each GRIO-managed filesystem. The qualified bandwidth is specified in the `griotab` file for local IRIX XFS filesystems or by using the `cxfs_admin`(1M) or `cmgr`(1M) command for shared CXFS filesystems.

The `ggd2` daemon is responsible for managing the allocation of available bandwidth between different applications and nodes.

You can adjust the qualified bandwidth as needed to make the best use of your system, taking into account the tradeoff between resource utilization and delivered I/O performance. For more information, see "Choosing a Qualified Bandwidth" on page 12.

## Managing Bandwidth: Encapsulation and Distributed Bandwidth Allocator

The `ggd2` daemon tracks the total qualified bandwidth and ensures that the total workload never exceeds the qualified bandwidth.

When `ggd2` begins managing a filesystem, every node with access to that filesystem is notified. Each node in turn creates a dedicated system stream for that filesystem, which is called the *non-GRIO stream*. From that point on, all user and system I/O that does not have an explicit GRIO reservation is encapsulated by this stream and then managed by the GRIO scheduler. For a local IRIX XFS filesystem, there is a single non-GRIO stream. For a shared CXFS filesystem, there is a non-GRIO stream on each node with access to the filesystem.

**Note:** The scheduling for all non-GRIO I/O within a GRIO-managed filesystem received from different applications and system services is on a first-come-first-served basis.

GRIO supports application-level reservations (created by GRIO-enabled applications using the `libgrio2` interfaces) and node-level bandwidth allocations (configured using the GRIO administration interfaces). The `libgrio2` interfaces permit an application to do the following:

- Reserve bandwidth
- Dynamically bind and unbind the resulting GRIO stream to any number of open file descriptors
- Modify its reservation
- Release its bandwidth back to the system

GRIO ensures that the requested guarantee is met for the aggregate I/O performed across the bound file descriptors.

However, applications are often not GRIO-enabled. For these applications, GRIO allows an administrator to configure a node-level bandwidth allocation. From CXFS 4.0 onwards, GRIO supports two types of node-level allocations:

- *Floor allocations* (`-F`), for which GRIO ensures that the node receives **at least** the configured bandwidth. While there is any unallocated bandwidth, and the node is issuing I/O at a rate greater than its initial allocation, `ggd2` will attempt to increase its allocation temporarily to help service the additional demand.
- *Ceiling allocations* (`-C`), for which the node receives **at most** the configured bandwidth. That is, the configured bandwidth acts as a cap on the amount of I/O that the node will be permitted to issue. This is the default.

To keep the total throughput of the filesystem high even when there are active GRIO streams or node-level allocations, `ggd2` attempts to allocate any unreserved portion of the qualified bandwidth for use by non-GRIO applications. This bandwidth is effectively lent for short periods of time until `ggd2` receives a new request for guaranteed-rate bandwidth, at which point it is reclaimed. GRIO applications have priority over non-GRIO applications.

The `ggd2` daemon periodically adjusts the amount of bandwidth allocated to the individual non-GRIO streams for its managed filesystems. This functionality is

referred to as the *distributed bandwidth allocator* (DBA). The DBA is responsible for determining how unreserved bandwidth is distributed between the nodes with access to the filesystem. By default, the DBA runs every two seconds , constantly allocating free bandwidth to nodes based on a range of dynamically monitored demand and utilization metrics. (To change the DBA default, edit the `ggd2.options` file on each of the GRIO server-capable nodes. For more information, see "`ggd2.options` File" on page 18.)

---

**Note:** The rate at which the DBA runs affects the delay that an application or node that does not have a GRIO reservation might experience when it starts doing I/O. The longer the interval, the longer a node may have to wait (with its I/O paused) before `ggd2` will increase its allocation.

---

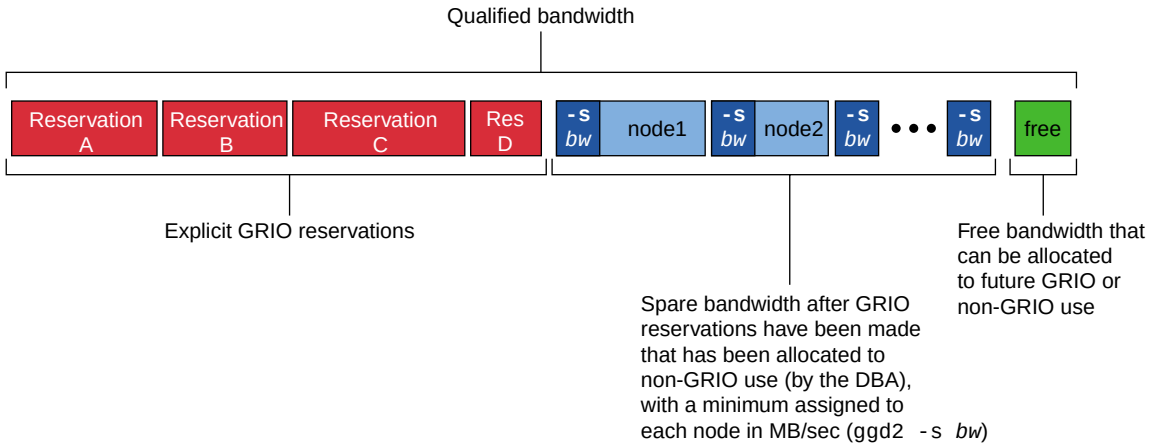


**Figure 2-1** Qualified Bandwidth

A user process can request a reservation using the `grio_reserve()` and `grio_reserve_fd()` library calls. Requests are forwarded to the `ggd2` that is actively managing the target volume domain. Requests to filesystems in the local domain are sent to the local instance of `ggd2` directly. Requests to filesystems in the cluster domain are forwarded to `ggd2` on the GRIO server, which may be running on a different node in the cluster.

For more information, see the `grio_reserve`(3X) man page.

## GRIO Server Relocation and Recovery

Each instance of `ggd2` maintains reservation and bandwidth state that must be kept consistent with one or more kernels.

If `ggd2` fails, a new `ggd2` instance will receive from the local kernel all of the information necessary to reestablish the following:

- Local volume reservations
- Cluster volume reservations (if the `ggd2` that failed was the GRIO server for the cluster)
- All of the DBA state for non-GRIO I/O

If a GRIO server node fails, a new GRIO server is automatically elected and all of the cluster volume reservations and DBA state is reestablished by that instance of `ggd2`.

You can also choose to manually migrate the GRIO server to another CXFS server-capable administration node in the cluster. For more information, see Chapter 4, "Administering GRIO" on page 25 and the `grioadmin`(1M) man page.

To determine the active GRIO server, use the `grioadmin -sv` command.

# Setting Up GRIO

This chapter discusses the following:



## Installation Requirements

In a cluster deployment, **every** node must be GRIO-enabled. As of CXFS 3.4, every supported platform is GRIO-enabled. For more information, see the *CXFS Administration Guide for SGI InfiniteStorage* and the *CXFS MultiOS Client-Only Guide for SGI InfiniteStorage*.

### IRIX Installation Requirements

To operate in local volume domain on an IRIX node, you must install the `eoe.sw.grio2` product.

To enable clustered GRIO support on IRIX, you must install both the `eoe.sw.grio2` and the `cxfs.sw.grio2_cell` products.

### SGI ProPack Installation Requirements

Install the following RPM on all SGI ProPack nodes in the cluster:

`grio2-cmds-`*version*`.ia64.rpm`

Install the following additional RPM on SGI ProPack server-capable nodes:

`grio2-server-`*version*`.ia64.rpm`

## Deployment Considerations for Cluster Volumes

You must observe the following constraints when setting up GRIO filesystems in a cluster:

- If any of the logical units (LUNs) on a particular device will be managed as GRIO filesystems, then all of the LUNs should be managed as GRIO filesystems. Typically, there will be hardware contention between separate LUNs, both in the storage area network (SAN) and within the storage device. If only a subset of the LUNs are managed, I/O to the unmanaged LUNs could still cause oversubscription of the device and could in turn violate guarantees on the managed filesystems.
- A storage device containing GRIO–managed filesystems should not be shared between clusters. The GRIO daemons running within different clusters are not coordinated, and unmanaged I/O from one cluster can cause guarantees in the other cluster to be violated.

## Data Layout

To set up a filesystem on a RAID device such that you achieve correct filesystem device alignment and maximize I/O performance, remember to do the following:

- Ensure that each data partition is correctly aligned with the internal disk layout of its LUN
- Set XVM stripe parameters correctly
- Pass correct volume geometry (stripe unit and width) to `mkfs_xfs`(1)

For more information, see the `grio2`(5) man page.

## Choosing a Qualified Bandwidth

You can adjust the qualified bandwidth to reflect the specific trade-off between delivered QoS and utilization of the storage infrastructure for your situation.

The following affect the qualified bandwidth you will choose:

- The hardware configuration
- The application work flow and I/O load
- The specific QoS requirements of applications and users

Typically, the first concern is that the required bandwidth can be delivered by the storage system. The second concern is the service time observed for individual I/Os.

Determining qualified bandwidth is an iterative process. There are several strategies you can use to determine and fine-tune the qualified bandwidth for a filesystem. For example:

- Establish a given bandwidth and then adjust so that the QoS requirements are met. Do the following:

  1. Make an initial estimate of the qualified bandwidth. You can use the fixed storage architecture parameters (RAID performance, number of HBAs, and so on) to estimate the anticipated peak bandwidth that can be delivered. The qualified bandwidth is then determined as an appropriate fraction of this peak.
  2. Configure `ggd2` appropriately using the `griotab` file or the `cxfs_admin`(1M) or `cmgr`(1M) command.
  3. Run a test workload.
  4. Monitor the delivered performance.
  5. Refine the estimate as needed.

- Establish that QoS requirements are satisfied and then adjust to maximize throughput. To do this, increase the load until the storage system can no longer meet the application QoS requirements; the qualified bandwidth must be lower than this value.
- Explore the space of possible workloads and test whether a given workload satisfies both bandwidth and application QoS requirements.

Although the hardware configuration provides a basis for calculating an estimate, remember that the qualified bandwidth is also affected by the particular work-flow issues and the QoS requirements of individual applications. For example, an application that has large tunable buffers (such as a flipbook application that does aggressive RAM caching) can tolerate a greater variation in I/O service time than can a media broadcast system that must cue and play a sequence of very short clips. In

the first example, the qualified bandwidth would be configured as a larger proportion of the sustained maximum. In the second example, the qualified bandwidth might be reduced to minimize the system utilization levels and improve I/O service times.

A high qualified bandwidth will generally achieve the greatest overall throughput but with the consequence that individual applications may intermittently experience longer service times for some I/Os. This variation in individual service times is referred to as *jitter*; as the storage system approaches saturation, service-time jitter will typically increase. A lower qualified bandwidth means that total throughput will be reduced, but because the I/O infrastructure is under less stress, individual requests will typically be processed with less variation in individual I/O service times. Figure 3-1 illustrates these basic ideas. The precise relationship between load on the storage system and variation in I/O service time is highly dependent on your storage hardware.

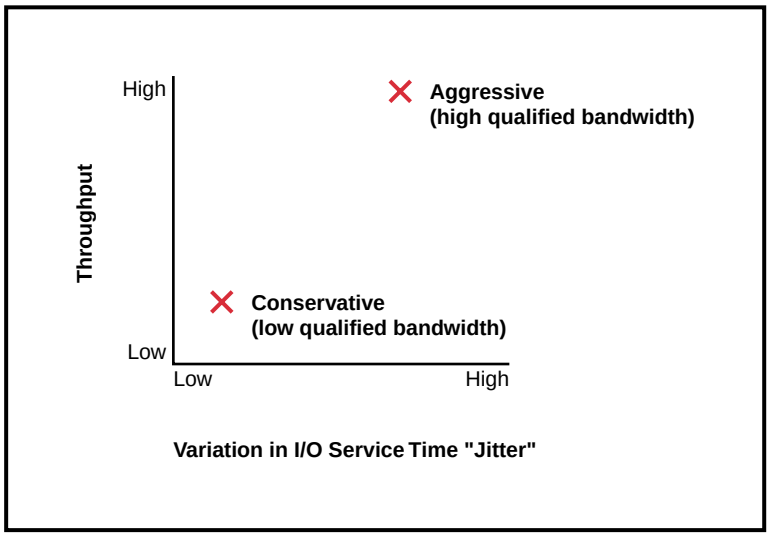


**Figure 3-1** Tradeoff Between Throughput and Variation in I/O Service Time (Jitter)

Some storage devices (particularly those with real-time schedulers) can provide a fixed bound on I/O service time even at utilization levels close to their maximum. In this case, the qualified bandwidth can be set higher even where applications have tight QoS requirements. The user-adjustable qualified bandwidth provides the

flexibility required for GRIO to work with both dedicated real-time devices as well as more common off-the-shelf storage systems.

---

**Note:** In all cases, you must verify the chosen qualified bandwidth by testing the storage system under a realistic workload.

---

You can use the `grioqos`(1M) tool to measure the delivered QoS performance. This tool extracts QoS performance for an active stream without disturbing the application or the kernel scheduler. GRIO maintains very detailed performance metrics for each active stream. Using the `grioqos` command while running a workload test lets you answer questions such the following for every active stream in the system:

- What has been the worst observed bandwidth over a 1-second period?
- What is the worst observed average I/O service time for a sequence of 10 I/Os?

For more information about GRIO tools and the mechanisms for accessing QoS data within the kernel, see Chapter 5, "Monitoring Quality of Service" on page 33, and the `grioqos`(1M) man page.

## Local Volumes and Cluster Volumes

A managed volume can be one of the following:

- A *local volume* is attached to the node in question. This volume is in the *local volume domain*.

  Local volumes are always managed by the instance of the `ggd2` daemon running on the node to which they are attached.

- A *cluster volume* is used with CXFS filesystems and is shared among nodes in a cluster. This volume is in the *cluster volume domain*.

  All cluster volumes are managed by a single instance of the `ggd2` daemon running on one of the CXFS administration nodes in the cluster; this node is referred to as the *GRIO server*. There is one GRIO server per cluster.

  The GRIO server is elected automatically. You can relocate it by using the `grioadmin`(1M) command. The GRIO server must be a CXFS administration node. Client-only nodes will never be elected as GRIO servers. For more information, see Chapter 4, "Administering GRIO" on page 25, and the `grioadmin`(1M) man page.

If a given CXFS administration node has locally attached volumes and has also been selected as the GRIO server, then the `ggd2` running on that node will serve dual-duty and will manage both its own local volume domain and the cluster volume domain.

For more information about CXFS, see "Cluster Volume Domain Configuration" on page 16, *CXFS Administration Guide for SGI InfiniteStorage*, and *CXFS MultiOS Client-Only Guide for SGI InfiniteStorage*.

## Local Volume Domain Configuration

To configure GRIO for local volume domains, you must provide information in the `/etc/griotab` file.

The `/etc/griotab` file lists the volumes that should be managed by GRIO and the maximum qualified bandwidth they can deliver. This file is read at startup and whenever `ggd2` receives a `SIGHUP` signal (such as when you issue a `killall -HUP ggd2` command). See the `ggd2`(1M) and `griotab`(4) man pages for more information.

## Cluster Volume Domain Configuration

You must use the `cxfs_admin`(1M) or `cmgr`(1M) command to configure cluster volumes for GRIO.

A prompting mode is also available for `cxfs_admin`(1M) or `cmgr`. For more information, see the *CXFS Administration Guide for SGI InfiniteStorage*.

If you have installed the `cxfs.sw.grio2_cell` subsystem and turned on GRIO, the `ggd2` daemon will automatically query the cluster configuration database for GRIO volume configuration information. `ggd2` dynamically tracks updates to the cluster database.

### `cxfs_admin` Configuration Examples

There are two GRIO attributes associated with filesystems:

- `grio_managed`, which specifies whether a filesystem is managed by GRIOv2 (`true`) or not (`false`). The default is `false`. Setting `grio_managed` to `false` disables GRIO management for the specified filesystem, but it does not reset the `grio_qual_bandwidth` value. In this case, `grio_qual_bandwidth` is left unmodified in the cluster database and ignored.

- `grio_qual_bandwidth`, which specifies a filesystem's qualified bandwidth in bytes (`B` suffix), kilobytes (`KB`), megabytes (`MB`), or gigabytes (`GB`), where the units are multiples of 1024. The default is `MB` for 4000 or less, `B` for 4001 or greater. If the filesystem is GRIO-managed, you must specify a qualified bandwidth with this attribute. You can modify the qualified bandwidth for a mounted filesystem without taking it offline.

  **Note:** These are advanced-mode attributes. When configuring GRIO with `cxfs_admin`, you should use `set mode=advanced`.

  For example, any one of the following commands sets a filesystem's qualified bandwidth to 1.2 GB/s:

```
cxfs_admin:cluster> modify filesystem grio_qual_bandwidth=1288500000
cxfs_admin:cluster> modify filesystem grio_qual_bandwidth=1258300KB
cxfs_admin:cluster> modify filesystem grio_qual_bandwidth=1288.8MB
cxfs_admin:cluster> modify filesystem grio_qual_bandwidth=1288.8MB
```

### `cmgr` Configuration Examples

To mark a filesystem as GRIO-managed and set its qualified bandwidth, use the following commands:

```
# /usr/cluster/bin/cmgr
Welcome to SGI Cluster Manager Command-Line Interface

cmgr> modify cxfs_filesystem filesystem in cluster cluster
cmgr> set grio_managed to true
cmgr> set grio_qualified_bandwidth to qualified_bandwidth
cmgr> done
```

The value for *qualified_bandwidth* is specified in bytes per second. For example, the following sets the qualified bandwidth to 200 MB/s (200*1024*1024):

```
cmgr> set grio_qualified_bandwidth to 209715200
```

To show the current status of a shared filesystem:

```
cmgr> show cxfs_filesystem filesystem in cluster cluster
...
              GRIO Managed Filesystem: true
              GRIO Managed Bandwidth: qualified_bandwidth
...
```

**Note:** In `cmgr`, you must unmount a filesystem before you can modify it.

## Licensing

The GRIO licensing schema supports one license for local volumes and one license for cluster volumes.

The `ggd2` daemon checks the license at startup, whenever it detects a configuration change, or when it receives a `SIGHUP` signal. The `ggd2` daemon validates the configuration at startup and whenever the configuration is changed.

If the configuration of a volume domain is altered and becomes unlicensed, `ggd2` enters a passive mode in which all further requests pertaining to that domain, with the exception of release requests, are denied. A message is sent to the system log and that volume domain will remain deactivated until the configuration returns to a licensed state, at which time another message will be logged indicating the domain is again active.

For more information, see the `license.dat`(5) man page.

## `ggd2.options` File

The `ggd2.options` file contains the command line options for `ggd2` when launched at startup.

The location of the file differs by operating system:

- IRIX: `/etc/config/ggd2.options`
- SGI ProPack: `/etc/cluster/config/ggd2.options`

You can uncomment and edit lines as required. The arguments are as follows:

`-d` *level* Sets the maximum debug level and logs the specified level of messages to both the system log and to an additional log file called `/var/tmp/ggd2log/`*PID*.

*level* is an integer in the range `0` through `4`(the higher the *level* number, the more debug information that is output). The levels are as follows:

- `0` logs critical system resource errors
- `1` logs the above plus `ggd2`–specific error and warning messages
- `2` logs the above plus important events or state changes
- `3` logs the above plus infrequent, less-important events
- `4` logs the above plus debug-level messages

By default, `ggd2` logs just level `0` critical system resource errors to the system log only.

`-f` Runs `ggd2` in the foreground. By default, `ggd2` is started as a daemon.

`-m` *bw* Specifies the minimum amount of *bw* bandwidth in KB/s (default) that `ggd2` will allocate for non-guaranteed user and system I/O **per GRIO-managed volume**. All nodes issuing non-GRIO I/O will receive a fair share of this minimum bandwidth. A node will be allocated the bigger value specified by `-m` or `-s`. The default is 1 MB/s.

For example, `-m2048` causes `ggd2` to allocate a minimum of 2048 KB/s to each GRIO-managed volume. This bandwidth becomes permanently allocated to non-GRIO I/O and cannot be reserved for GRIO I/O. Use the suffix `K` or `M` to explicitly specify bandwidth in KB/s or MB/s. For example, `-m3M` causes `ggd2` to allocate a minimum of 3 MB/s to each GRIO-managed volume.

`-r` *percent* Reserves a percentage of each volume's available qualified bandwidth for GRIO I/O. Reservation requests are then serviced directly from this pool of cached free bandwidth without blocking. *percent* is the percentage of each volume's qualified bandwidth that `ggd2` attempts to keep unallocated, expressed as an integer in the range `0` through `100`. You should choose this value based on the following:

- Expected I/O utilization levels
- Importance of minimizing the stream creation latency
- Expected rate at which reservation requests will be made

By default, `ggd2` allows unreserved bandwidth to be allocated for servicing non-GRIO I/O. This maximizes the total throughput of the system. However, as `ggd2` only makes adjustments to these allocations periodically, a new reservation may block until `ggd2` can reclaim the requested bandwidth.

**Note:** Using the `-r` option causes a proportion of the unreserved I/O capacity to remain unused and reduces the total throughput and efficiency of the system for non-GRIO I/O. You should only use this option if minimizing reservation latency is a priority.

For example, given a volume with a qualified bandwidth of 200 MB/s, `-r 20` will instruct `ggd2` to try to keep up to 20% (40 MB/s) of any remaining unreserved bandwidth cached and available for servicing reservation requests directly. `ggd2` adjusts this cache of free bandwidth every time the distributed bandwidth allocator (DBA) runs, which defaults to once every 2 seconds (see `-u`). With these settings, `ggd2` will be able to grant an additional 40 MB/s every 2 seconds without blocking any reservation requests.

`-s` *bw*
: Specifies the minimum amount of *bw* bandwidth in KB/s (default) that `ggd2` will allocate for non-GRIO user and system I/O **per node**. A node will be allocated the bigger value specified by `-s` or `-m`. The default is 1 MB/s.

  For example, `-s2048` causes `ggd2` to allocate a minimum of 2048 KB/s to each node accessing a GRIO-managed volume. This bandwidth becomes permanently allocated to non-GRIO I/O and cannot be reserved for GRIO I/O. Use the suffix `K` or `M` to explicitly specify bandwidth in KB/s or MB/s. For example, `-s3M` causes `ggd2` to allocate a minimum of 3 MB/s to each node accessing a GRIO-managed volume.

`-u` *ms*
: Specifies the distributed bandwidth allocator (DBA) update interval in milliseconds (ms), where *ms* is a value in the range `250` through `100000`. The default is `2000` (2 seconds). For more information about DBA, see "Managing Bandwidth: Encapsulation and Distributed Bandwidth Allocator" on page 7.

**Note:** The rate at which the DBA runs affects the delay that an application or node that does not have a GRIO reservation might experience when it starts doing I/O. The longer the interval, the longer a node may have to wait (with its I/O paused) before ggd2 will increase its allocation.

For example:

```
# command line options for ggd2 when launched from /etc/init.d/grio2
# uncomment/edit lines as required
#

# Minimum non-GRIO bandwidth per node. Units are Mbytes/sec
# -s 1M

# debug level, in the range 0 to 4
# -d 1

# minimum reserve bandwidth to accommodate short-latency reservation
# demands, expressed as a percentage of the total qualified bandwidth
# -r 30

# Distributed Bandwidth Allocator (DBA) update interval
# value in milliseconds
# -u 2000
```

For changes to this file to take effect, do one of the following on the GRIO server, which will cause `ggd2` to reread its options file:

- Stop and restart the `grio2` service:

  ```
  # /etc/init.d/ grio2 stop
  # /etc/init.d/grio2 start
  ```

- Run the following:

  ```
  # run killall -HUP ggd2
  ```

To determine the active GRIO server, use the `grioadmin -sv` command.

**Note:** In the event of a GRIO server relocation or recovery, you must perform the above steps on each GRIO server-capable node in the cluster.

## Starting GRIO on IRIX Servers

On IRIX, it is possible to have both the GRIOv1 and GRIOv2 subsystems installed. However, only one of these subsystems can be active. The subsystem that is turned `on` in `chkconfig` is started by default at boot time and remains in effect until the `chkconfig` setting is changed and the machine is rebooted.

### Starting GRIOv2 when GRIOv1 is Active

Suppose you were running GRIOv1 and wanted to switch to GRIOv2. After performing the configuration tasks discussed in this guide, you would do the following:

1. Turn off GRIOv1 (`grio`) and turn on GRIOv2 (`grio2`):

   ```
   # chkconfig grio off
   # chkconfig grio2 on
   ```

2. Reboot the system to allow the kernel to be reinitialized with the GRIOv2 scheduler.

You do not need to manually start GRIOv2 because the daemon is automatically started upon reboot when the `chkconfig` setting is `on`.

**Note:** If GRIOv1 is still enabled when you perform a GRIOv2 library call, the return will be `ENOSYS`. If you do not have either the GRIOv1 or GRIOv2 kernel initialized, the return will be `EAGAIN`, indicating that the subsystem has not yet initialized and the application should retry the request.

## Starting GRIOv2 when GRIOV1 is Not Active

If you have not run GRIOv1 during the current boot session, you can start GRIOv2 by doing the following:

1. Turn on GRIOv2:

   ```
   # chkconfig grio2 on
   ```

2. Start GRIOv2:

   ```
   # /etc/init.d/grio2 start
   ```

You must perform the manual start only once. When the machine is rebooted, GRIOv2 will be restarted automatically as long as its `chkconfig` setting remains `on`.

# Administering GRIO

You can use the `cxfs_admin`(1M) or `cmgr`(1M) command to configure a qualified bandwidth and activate a filesystem as GRIO-managed. You can use the `grioadmin` tool to perform node-level administration tasks for local XFS and shared CXFS filesystems, such as the following:

- List active GRIO reservations on the current node
- Create, modify, and release node-level bandwidth allocations
- Query available bandwidth

## `grioadmin` Command Line

`grioadmin [`*options*`] [`*fs*`|`*streamID*`]`

The arguments are as follows:

`-a` Prints the bandwidth available for reservation for the specified filesystem and the total bandwidth currently reserved for use by the local node, either allocated to the node or reserved by applications. This total includes bandwidth temporarily allocated to the node by the distributed bandwidth allocator (DBA) running within `ggd2`.

`-C` Indicates that the node-level allocation requested with the `-g` or `-m` option should be **at most** *bw* MB/s (a ceiling). This is the default behavior for `-g`.

`-F` Indicates that the node-level allocation requested with the `-g` or `-m` option should be **at least** *bw* MB/s (a floor).

`-g` *bw* Allocates *bw* MB/s for access by the local node to the specified filesystem, if bandwidth is available. This node-level allocation is shared by all applications running on the node that are not GRIO aware. The allocation type can be either a ceiling (`-C`) or a floor (`-F`). The default allocation type is a ceiling .

`-h` Print a usage message.

`-l` Lists active streams in an easily parsed form (one per line).

| | |
|---|---|
| `-L` *cell* | Relocates the GRIO server to the node with the cell ID *cell*. To determine the cell ID, use any of the standard CXFS reporting utilities, such as `cxfs_admin`, `cxfs_info`, or `clconf_info`. |
| `-m` *bw* | Modifies the current node-level allocation on the specified filesystem to a bandwidth of *bw* MB/s. You can modify the type of an active allocation by also specifying either `-C` or `-F`. |
| `-r` | Releases the current node-level allocation on the specified filesystem. |
| `-s` | Prints a more human-readable summary of active streams than `-l`. The results are grouped per filesystem. |
| `-v` | Displays verbose output (used with `-s`). |
| *streamID* | Specifies the ID of an active GRIO stream. |
| *fs* | Specifies a path that identifies a mounted GRIO-managed filesystem (in which case the non-GRIO stream for the filesystem is used). |

If you specify `grioadmin` without any arguments, it prints a usage message by default.

## `grioadmin` Examples

The following example commands create, inspect, modify, and remove a node-level allocation of the GRIO-managed filesystem `/stripe` with a qualified bandwidth of 350 MB/s.

**Note:** `grioadmin` classifies active streams as one of the followings:

- `App`, which is a GRIO stream that is created explicitly by an application using the interfaces in the GRIO library. Only I/O to file descriptors that have been bound to such a stream using `grio_bind`(3X) will receive the requested QoS guarantee.
- `Dynamic`, which is the component of the node-level allocation that is controlled by the DBA running within `ggd2`. It is updated periodically based on the amount of free bandwidth in the filesystem and the I/O demand from different nodes.
- `Static`, which is the component of the node-level allocation that has been configured by an administrator using `grioadmin` with the `-g` option. The `Static` component is not affected by the operation of the DBA or other `App` reservations, and persists until you remove it manually by using the `-r` option. You can configure a node-level allocation as either a floor or a ceiling. The `grioadmin -s` option output indicates the type of active node-level allocations by printing a `FLOOR` or `CEIL` designator with the `Static` component of the currently allocated bandwidth.

1. Query the available bandwidth on the `/stripe` filesystem and show the active streams:

```
$ grioadmin -a /stripe
349.94 MB/s available on /stripe
1.00 MB/s allocated to this node
$ grioadmin -s
/stripe:
  Dynamic         1.00 MB/s
```

   The output from these commands shows the following:

   - The minimum dynamic allocation that `ggd2` is configured to make to an idle node is 1.00 MB/s
   - There are just under 350 MB/s available to be reserved

2. Create a 200–MB/s node-level allocation for this node in the `/stripe` filesystem:

```
$ grioadmin -g200 /stripe
Static allocation configured for /stripe.
Stream ID is d919c6e5-8405-1029-8d74-08006913a7f7
```

   A stream will be created for the allocation and is attached to the non-GRIO stream in the kernel.

3. Verify that the available bandwidth was reduced accordingly:

```
$ grioadmin -a /stripe
149.94 MB/s available on /stripe
200.06 MB/s allocated to this node
```

4. Inspect the kernel streams:

```
$ grioadmin -s
/stripe:
  Dynamic         1.00 MB/s
  Static        200.00 MB/s  CEIL
```

This shows that the `/stripe` filesystem has its initial minimal `Dynamic` allocation and a new node-level (`Static`) allocation.

To be more verbose and print the stream IDs, add the `-v` option:

```
$ grioadmin -sv
/stripe:
  Dynamic         1.00 MB/s  b77c9351-7b63-1029-8f56-08006913a7f7
  Static        200.00 MB/s  d919c6e5-8405-1029-8d74-08006913a7f7  CEIL
```

5. Increase the node-level allocation to 300 MB/s without disturbing in-progress I/O on the node:

```
$ grioadmin -m300 /stripe
Static bandwidth allocation for filesystem /stripe has
been modified:
$ grioadmin -s
/stripe:
  Dynamic         1.00 MB/s
  Static        300.00 MB/s  CEIL
```

6. Start a GRIO-aware application that requests a 50–MB/s reservation and print the stream IDs:

```
$ grioadmin -s /stripe
/stripe:
  Dynamic         1.00 MB/s
  Static        200.00 MB/s  CEIL
  App (31932)    50.00 MB/s
```

This output shows that the new `App` stream has a process ID of `31932`.

7. Terminate the application and remove the node-level allocation:

```
$ grioadmin -r /stripe
Static bandwidth allocation for filesystem /stripe has
been released.
$ grioadmin -s
/stripe:
  Dynamic         1.00 MB/s
```

The following examples show using `grioadmin` on a Windows client to create a floor reservation (`-F`) , modify it to a ceiling allocation (`-C`) , and then release it:

```
C:\>grioadmin -sv
GRIO cluster server is cxfsaltix1 (cell 0)
x:\mnt\tp9500_1: cluster:
  Dynamic         1.00 MB/s  6016cd36-8900-2a10-b682-9c12fb0816e8

C:\>grioadmin -F -g 10 x:\mnt\tp9500_1
Static allocation configured for x:\mnt\tp9500_1.
Stream ID is a267e371-caca-324f-3b89-f8c41eaca4b0.

C:\>grioadmin -sv
GRIO cluster server is cxfsaltix1 (cell 0)
x:\mnt\tp9500_1: cluster:
  Dynamic         1.00 MB/s  6016cd36-8900-2a10-b682-9c12fb0816e8
  Static         10.00 MB/s  a267e371-caca-324f-3b89-f8c41eaca4b0 FLOOR

C:\>grioadmin -C -m 10 x:\mnt\tp9500_1
Static bandwidth allocation for filesystem x:\mnt\tp9500_1 has been modified.

C:\>grioadmin -sv
GRIO cluster server is cxfsaltix1 (cell 0)
x:\mnt\tp9500_1: cluster:
  Dynamic         1.00 MB/s  6016cd36-8900-2a10-b682-9c12fb0816e8
  Static         10.00 MB/s  a267e371-caca-324f-3b89-f8c41eaca4b0 CEIL

C:\>grioadmin -r x:\mnt\tp9500_1
Static bandwidth allocation for filesystem x:\mnt\tp9500_1 has been released.

C:\>grioadmin -sv
GRIO cluster server is cxfsaltix1 (cell 0)
x:\mnt\tp9500_1: cluster:
```

```
Dynamic          1.00 MB/s  6016cd36-8900-2a10-b682-9c12fb0816e8
```

Figure 4-1 through Figure 4-3 show `cxfs_info` examples on a Windows node.

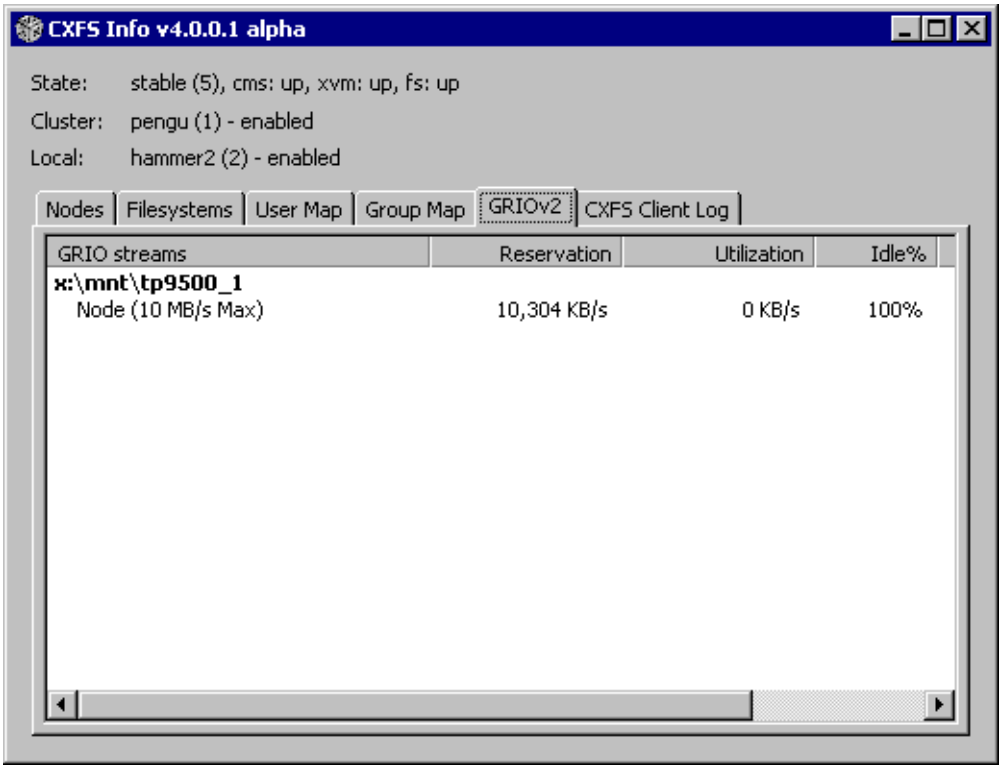


**Figure 4-1** Ceiling (**Max**) Node-Level Allocation

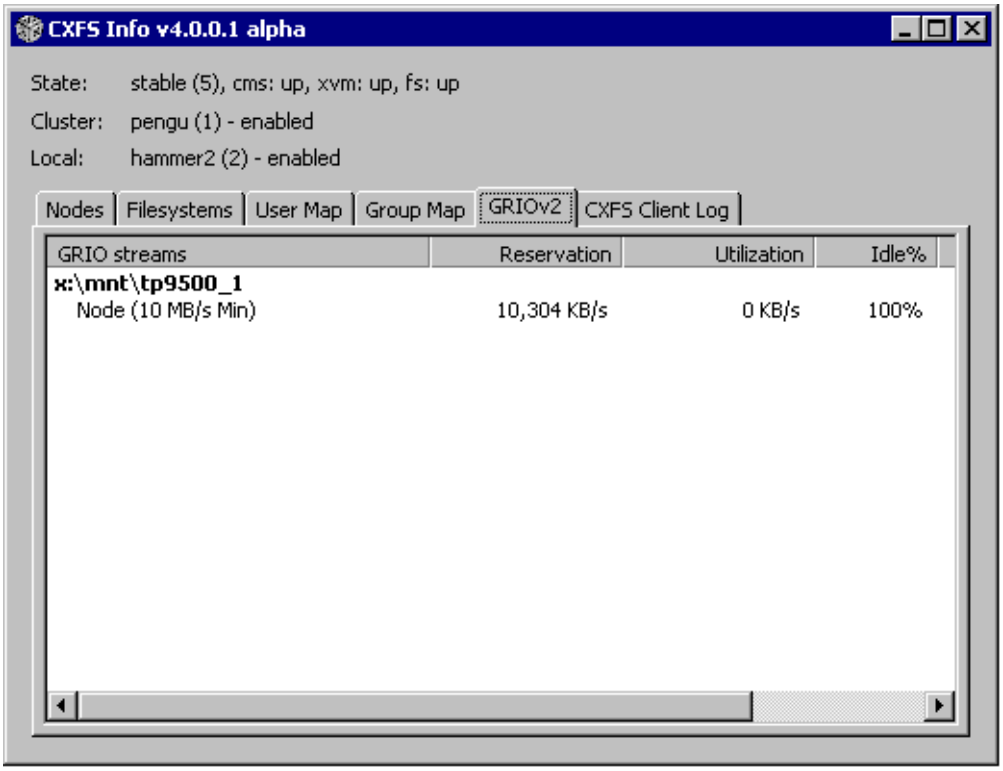


**Figure 4-2** Floor (**Min**) Node-Level Allocation

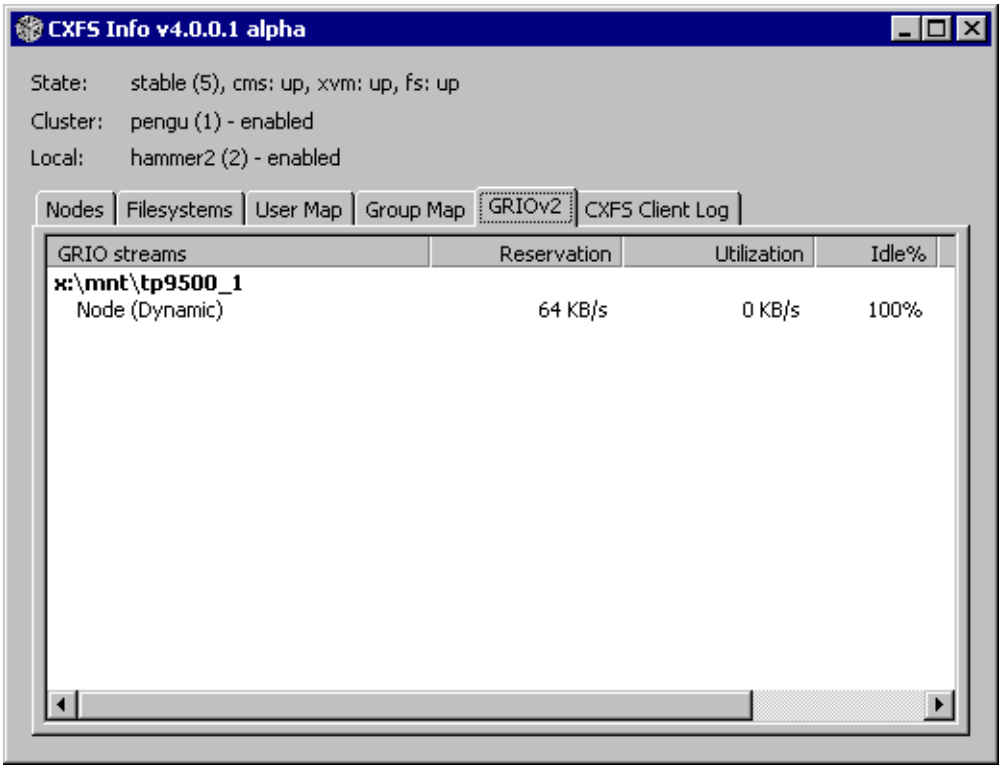


**Figure 4-3** No Node-Level Allocation

# Monitoring Quality of Service

You can use the `grioqos`(1M) command to extract and report the QoS metrics that GRIO maintains for each active stream. `grioqos` loops, repeatedly fetching new QoS statistics from the kernel for the specified application stream or node-level allocation. `grioqos` displays a header containing the following information:

- Operating system and release
- Date and time
- Filesystem reported on
- Current reservation and stream ID

This section discusses the following:



## `grioqos` Command Line

`grioqos [`*options*`] [`*streamID*`|`*fs*`] [`*delay*` [`*count*`]]`

| | |
|---|---|
| `-c` | Clears the screen before printing each new set of statistics. |
| `-h` | Prints a usage message. |
| `-i` | Reports the following low-level QoS metrics for all currently configured sampling intervals:<br>`minbw` |

```
maxbw
lastbw
minio
maxio
lastio
```

For details about these metrics, see "Quality-of-Service Metrics" on page 41.

`-I` *intervals*

Reports the same low-level QoS metrics as `-i`, but for a specified range of sampling intervals.

*intervals* is a comma-separated list of sampling intervals expressed as either a number of I/Os or a time interval in msecs. For example, the following would report results averaged over the last 5 and 10 I/Os, and over the last 1 and 2 seconds, respectively:

```
-I 5,10,1000ms,2000ms
```

`-l`

Lists active streams in an easily parsed form (one per line with the following fields:

- Filesystem mount point (or the string `<unmounted>` if the filesystem is not mounted)
- Type of the stream
- Stream ID
- Reserved bandwidth reported in bytes and msecs
- Process ID for application-created streams

`-m`

Enables monitoring mode, which reports the following high-level stream and I/O metrics:

```
bytes
msecs
bckt
bckt (max)
total
rate
bklg
issd
```

```
idle
thrt
wait
```

See "Monitoring Stream and I/O Metrics" on page 38 for more information.

`-n` Prints a more human-readable version of the performance information reported with the `-i` option for all currently configured sampling intervals:

```
lastbw
minbw
maxio
```

For more information, see "Quality-of-Service Metrics" on page 41

The minimum bandwidth and maximum average service time are the metrics of most concern when attempting to deliver guaranteed data rates.

`-N` *intervals* Reports the same metrics as `-n`, but for a specified range of sampling intervals.

*intervals* is a comma-separated list of sampling intervals expressed as either a number of I/Os or a time interval in msecs. For example, the following would report results averaged over the last 5 and 10 I/Os, and over the last 1 and 2 seconds, respectively:

```
-N 5,10,1000ms,2000ms
```

`-o` *file* Logs output to the specified *file*.

`-r` Resets the specified statistics when used with one of the following options:

- High-level stream statistics: `-m`
- Low-level QoS statistics: `-i`, `-I`, `-n`, `-N`, `-t`, or `-T`

The `-r` option is ignored if none of these other options is specified.

GRIO will continue to update some kernel statistics even when no I/O is being performed (such as when the rate metric reported in the `-m` mode is updated even on an idle stream). In order to get results that accurately correspond with those seen by a user application, you should start `grioqos` with the `-r` option at the same time that the application test begins.

`-R` *intervals*

Reconfigures the kernel QoS monitoring intervals and resets the statistics. This allows you to change the set of sampling intervals used in the kernel to compute recent bandwidth and average service time.

*intervals* is a comma-separated list of sampling intervals expressed as either a number of I/Os or a time interval in msecs. For example, the following would report results averaged over the last 5 and 10 I/Os, and over the last 1 and 2 seconds, respectively:

```
-R 5,10,1000ms,2000ms
```

By default, GRIO is configured to compute statistics for a wide range of sampling intervals. However, it can be useful to change these intervals using the `-R` option when a monitored application has a buffering behavior that is not well-matched by the default intervals.

---

**Note:** GRIO always configures two additional intervals automatically:

- The single sample interval, which tracks the best and worst case service times for individual I/Os
- The maximum interval, which is as large as the kernel data structures can accommodate

---

`-s`

Prints a more human-readable summary of active streams than `-l`. Results are grouped per filesystem and include the following:

- Stream type
- Process ID (for application streams)

| | |
|---|---|
| | • Bandwidth reservation in MB/s<br>• Stream IDs (when `-v` is also specified)<br>For more information about the output format, see the `grioqos`(1M) man page. |
| `-t` | Displays a per-stream I/O service time histogram for all buckets. |
| `-T` *buckets* | Displays a per-stream I/O service time histogram for the specified *buckets*. You can display a ranges of values. For example, the following would cause the values of 11 histogram buckets to be displayed:<br>`-T 0,1,2,3,20-25,52` |
| `-v` | Display verbose output (used with `-s`). |
| *streamID* | Specifies the ID of an active GRIO stream. |
| *fs* | Specifies a path that identifies a mounted GRIO-managed filesystem. |
| *delay* | Specifies the length of time in seconds that `grioqos` should sleep before retrieving each new set of statistics. |
| *count* | Specifies the total number of samples to be retrieved. |

If you specify `grioqos` without any arguments, it prints a usage message by default.

To terminate `grioqos`, press `Ctrl-C` on Windows or send a `SIGINT` on other platforms.

## GRIO Scheduler

Interpreting the statistics collected by `grioqos` requires a basic understanding of the GRIO scheduler.

GRIO uses the token bucket abstraction to limit the average rate and burstiness of I/O flowing to or from the filesystem. Conceptually, each stream has a bucket of tokens. Each token confers the right to issue one unit of I/O. Tokens are added to the token bucket at a rate corresponding to the GRIO reservation and accumulate up to the maximum size of the bucket, at which point any further tokens are discarded. When a new I/O request arrives, it is issued if there are sufficient tokens available to

the token bucket; if there are insufficient tokens, it is added to the throttle queue for the stream, where it is held for a short period before the token bucket is checked again. The rate at which tokens accrue to the token bucket controls the average rate of the stream. The maximum size of the token bucket controls the size of the largest burst of I/O that can be issued.

The ability to issue a temporary burst of I/O above the reserved data rate is important. It is the mechanism within GRIO by which an application or device that temporarily falls below the required data rate can catch up, thus preserving the required average data rate.

GRIO implements a variation of the weighted round-robin scheduling discipline. At each scheduler activation, it visits each stream in the system and issues as much I/O as it can, up to the limit of the token bucket. The order in which the streams are visited is always the same. To increase the determinism of the resulting I/O flow, GRIO will (on platforms where it is possible) attempt to disable further I/O reordering operations in lower-level devices.

## Monitoring Stream and I/O Metrics

In monitoring mode (enabled with `-m`), `grioqos` reports the following metrics:

`bytes, msecs`
: Reports the current GRIO reservation. If the monitored stream is a non-GRIO stream, this includes both the static and dynamic components (and may change as the DBA periodically adjusts the dynamic allocation or if an administrator modifies the static allocation using `grioadmin`). An application reservation may change if the application uses the `grio_modify`(3X) call to modify its reservation at runtime.

`bckt, bckt (max)`
: Describes the current state of the token bucket:

- `bckt` measures the current contents of the token bucket in MB. The contents of `bckt` change continuously as I/O is issued.
- `bckt = (max)` is the size of the token bucket in MB and the maximum burst of I/O that GRIO will issue to the filesystem. The value of `bckt (max)` is related to the size of the current reservation and only changes when the reservation is changed.

| | |
|---|---|
| `total, rate` | Describes the amount of data transferred:<br>• `total` is the total amount of data in MB transferred across the stream since it was created or the statistics were reset<br>• `rate` is the overall data rate in MB/s that was achieved<br>When a stream is first initialized, the token bucket is full, which means that `bckt` is equal to `bckt (max)`. An unthrottled application can issue a large initial burst of I/O before it drains its token bucket and the GRIO throttle forcibly slows it down. Depending on the size of individual I/Os, the action of the throttle can cause the instantaneous bandwidth to oscillate slightly above and below the guaranteed rate. In these cases, however, the overall data rate including the initial burst is greater than the requested data rate and can be verified with the rate metric (for example, by using `grioqos -rm`). |
| `bklg, issd` | Tracks I/Os being actively processed by the stream:<br>• `bklg` is the backlog of I/O that has been placed on the throttle queue<br>• `issd` is active I/O that has been issued to the volume but has not yet completed |
| `idle, thrt, wait` | Accounts for the utilization of the stream. These are instantaneous metrics that are computed for the period since the last sample:<br>• `idle` is the percentage of the time during which the stream was not processing I/O, that is, there was no active I/O and no I/O on the throttle queue (`bklg` and `issd` are both equal to 0)<br>• `thrt` is the percentage of the time during which the stream had I/O on the throttle queue (`bklg` is non-zero) |

- `wait` is the percentage of the time during which there was active I/O (`issd` is non-zero)

The stream utilization metrics (`idle`, `thrt`, and `wait`) can be useful when trying to understand the interaction between an application, the GRIO scheduler, and the storage device. Table 5-1 describes commonly observed behaviors and their corresponding metrics.

**Table 5-1** Relationship of Stream Utilization Metrics to Application State

| idle | thrt | wait | Application State |
|---|---|---|---|
| Low | Low | Low | Expected behavior for a self-throttled application:<br>• The application is issuing I/O to the filesystem efficiently, so the stream is rarely idle<br>• The application is not issuing I/O at a rate faster than its reservation, as there is little I/O on the throttle queue of the stream<br>• I/O is being serviced quickly suggesting that the filesystem is not currently oversubscribed |
| Low | High | Low | Expected behavior for an application being throttled by GRIO. |
| Any | Any | High | The application is spending a lot of time waiting for I/O. This may or may not be a problem, but if the application is seeing poor QoS as reported by the `-i`, `-I`, `-n`, `-N`, `-t`, or `-T` options, you should review the qualified bandwidth for this filesystem. An indication of poor QoS would be low worst-case bandwidth and high average service times over relatively long sampling intervals. |
| High | Any | Any | The stream is spending a lot of time idle. The application may not be issuing I/O to the filesystem efficiently. You should investigate whether it is using multithreaded or asynchronous I/O. If the desired data rate in userspace is not being achieved, the behavior of the application should be reviewed. |

## Quality of Service

Depending on the amount of I/O buffering an application performs, it may be more or less sensitive to variation in I/O service time, also known as *jitter*. This can vary from tens of seconds for applications that have large buffers and use threaded or asynchronous I/O, to tens of milliseconds for single-threaded applications with little buffering that require a low upper bound on I/O service time.

Approaches to measuring I/O performance often tend to focus at the ends of this spectrum, measuring one of the following (which can be limiting as a result):

- Average bandwidth and ignoring the effects of service interruptions over shorter time intervals
- Worst-case service time that (for applications that are able to tolerate more jitter) can be a stronger criteria that is useful

The GRIO QoS infrastructure provides a configurable mechanism for monitoring performance over the entire range of time scales from the service times of individual I/Os to the sustained bandwidth over long sampling intervals. It can do so for an individual application or over a period of time without instrumenting or otherwise disrupting the performance of the application.

## Quality-of-Service Metrics

Within the kernel, GRIO records the I/O completion times for all recent I/Os to or from a stream. From this high-resolution data, it computes a number of derived metrics that can be efficiently exported to userspace. You can change the monitoring intervals over which these metrics are computed by using `grioqos`. Sampling intervals can be expressed as either a time *t* (such as `1000ms`) or as a number of individual samples *n*. For instance, `grioqos` can display average I/O service time and bandwidth for the last four I/Os, the last 200ms, the last second, and so forth.

GRIO computes the following metrics for each configured sampling interval:

| | |
|---|---|
| `lastbw` | Describes the recent average bandwidth, which is the bandwidth observed over the last *t* ms or *n* samples. It is an instantaneous metric describing recent stream activity. |
| `minbw, maxbw` | Describes the minimum and maximum values of `lastbw`. These metrics track the worst- and best-case bandwidth delivered over any continuous interval of |

| | |
|---|---|
| | the specified length since the creation of the stream or the last time the statistics were reset. |
| `lastio` | Describes the average I/O service time for I/Os over `t` ms or *n* samples. When *n* is 1, this metric records the actual service times of individual I/Os. When *n* is greater than 1, this metric is the average of the observed service times. It is an instantaneous metric describing recent stream activity. |
| `minio, maxio` | Describes the minimum and maximum values of `lastio`. Like `minbw` and `maxbw`, these metrics track the worst- and best-case average service times delivered over any continuous interval of the specified length since the statistics were initialized or last reset. |

## `grioqos` Caveats

There is a size restriction on the kernel structures used to hold recent I/O statistics. If a requested metric cannot be computed because there is insufficient data, a single hyphen (`-`) is printed. This can also happen when the QoS metrics have been recently reset using the `-r` or `-R` options. For example, requesting a sampling interval of `10000ms` may display only a hyphen (-) because the GRIO kernel structures cannot hold enough individual samples to compute an average over ten seconds. However, for most I/O rates and sampling intervals, the kernel structures should be adequate.

Use care when interpreting the low-level QoS statistics. A number of the bandwidth and service time measures only make sense if they have been recorded during a period of continuous, consistent application I/O (for example, for a video playout).

The `lastbw` and `maxbw` metrics are meaningful regardless of the behavior of the application. However, `minbw` tracks all interruptions to the flow of I/O. This includes interruptions due to the normal operation of the application as opposed to an actual service interruption in the filesystem or device. Thus, if the application stops and starts I/O during the sampling period, this will be recorded in the `minbw`, which will in turn be of little use in detecting a real service interruption and is unlikely to provide any useful insight into the performance of the application and system.

Similarly, the `lastio` metric is most useful if the application uses a consistent request size when issuing I/O to the filesystem. If the application issues I/O of widely varying size, then the service time is permuted both by filesystem and device issues and the behavior of the application. For such applications, this makes it very difficult

to determine the origin of a performance issue. This is particularly true for non-GRIO streams., which manage all of the I/O on a node that does not otherwise have an explicit GRIO reservation. This includes the following:

- Direct I/O from applications that do not have a GRIO reservation
- Buffered I/O from all sources via the buffer cache (or whatever the native filesystem caching mechanism is for the platform)
- All other system I/O to the managed filesystem

The result is that the non-GRIO stream may see a large variation in I/O sizes and the average service time of those I/Os is unlikely to provide useful insight into the performance of the system.

## `grioqos` Examples

This section shows `grioqos` used to monitor a GRIO-aware application. High-level stream and low-level quality-of-service metrics are collected. The application is temporarily suspended to show the effect on the stream utilization and average data rate. The example filesystem `/mirror` has a qualified bandwidth of 30 MB/s.

1. Confirm the available bandwidth on `/mirror`:

```
$ grioadmin -a /mirror
29.94 MB/s available on /mirror
0.06 MB/s allocated to this node
```

   There are just under 30 MB/s available, and a minimal dynamic allocation. Now we start the test application, which makes a 20–MB/s reservation and starts performing reads as fast as it can. The I/O size is just under 8 MBs. The application is multithreaded and configured to have up to four I/Os active.

2. List the active streams and get the stream ID of the application's GRIO stream:

```
$ grioqos -sv
/mirror:
  Dynamic          0.06 MB/s  b77c9351-7b63-1029-8f56-08006913a7f7
  App (6754151)   20.00 MB/s  03041498-871c-1029-87e2-08006913a7f7
```

3. Monitor the application stream:

```
$ grioqos -m 03041499-871c-1029-87e2-08006913a7f7 1

IRIX64 octane 6.5 01062343 IP30 07/21/05

Filesystem: /mirror

App (6754151) 20.00 MB/s 03041498-871c-1029-87e2-08006913a7f7

-           bytes msecs  bckt (max)  total rate  bklg issd idle thrt wait
-           bytes    ms    MB    MB     MB MB/s    MB    MB   %    %    %
21:00:38 20971520  1000 26.97 40.00   0.00 0.00  0.00 15.82   -    -    -
21:00:39 20971520  1000  7.63 40.00  31.64 25.5  0.00 23.73   0    5  100
21:00:40 20971520  1000  4.12 40.00  63.28 28.1 15.82 15.82   0   88  100
21:00:41 20971520  1000  0.61 40.00  94.92 29.1 23.73  7.91   0  100   85
21:00:42 20971520  1000  5.01 40.00 110.74 25.9 23.73  7.91   0  100   72
21:00:43 20971520  1000  1.49 40.00 134.47 25.5 23.73  7.91   0  100   55
21:00:44 20971520  1000  5.89 40.00 158.20 25.2 31.64  0.00   0  100   71
21:00:45 20971520  1000  2.41 40.00 174.02 23.8 23.73  7.91   0  100   60
21:00:46 20971520  1000  6.80 40.00 197.75 23.8 31.64  0.00   0  100   65
21:00:47 20971520  1000  3.29 40.00 213.57 22.9 23.73  7.91   0  100   66
21:00:48 20971520  1000  7.69 40.00 237.30 23.0 31.64  0.00   0  100   61
21:00:49 20971520  1000  4.18 40.00 253.12 22.3 23.73  7.91   0  100   70
21:00:50 20971520  1000  0.67 40.00 276.86 22.4 23.73  7.91   0  100   55
21:00:51 20971520  1000  5.13 40.00 292.68 21.9 23.73  7.91   0  100   70
...
```

The first few samples show that the token bucket `bckt` is initially full, which allows the overall data rate rate to jump above the reserved 20 MB/s briefly (see "Monitoring Stream and I/O Metrics" on page 38).

The stream utilization metrics `idle`, `thrt`, and `wait` show that while the application is draining its token bucket, the application spends all of its time waiting for I/O to the device. Very quickly, the token bucket empties completely and GRIO begins to throttle the application. `thrt` jumps to 100%. `wait` drops to around 60-70%, which is consistent with the qualified bandwidth.

The maximum this filesystem can deliver is 30MB/s, therefore a reservation of 20MB/s should keep the filesystem active approximately two-thirds of the time, which is what we see. The application is clearly very efficient about issuing I/O

to the filesystem (multithreaded with four active I/Os), because there is never any point when the stream is idle and the filesystem does not have I/O to process.

4. To simulate an interruption, temporarily suspend the application in userspace (sending it a `SIGSTOP`). The `grioqos -m` output would change as follows:

```
21:01:04 20971520  1000  7.42 40.00 561.62 21.2 15.82  0.00   0  100   61
21:01:05 20971520  1000 11.82 40.00 577.44 21.0  0.00  0.00  31   44   49
21:01:06 20971520  1000 32.04 40.00 577.44 20.2  0.00  0.00 100    0    0
21:01:07 20971520  1000 40.00 40.00 577.44 19.5  0.00  0.00 100    0    0
21:01:08 20971520  1000 40.00 40.00 577.44 18.9  0.00  0.00 100    0    0
...
```

The application stops issuing I/O completely and immediately the utilization metrics change:

- The token bucket fills
- Any remaining I/O on the throttle queue drains out (`thrt` goes to 0)
- The stream becomes completely idle

---

**Note:** The `rate` metric, which computes the overall data rate, is updated even while the stream is idle and gradually decreases during this period of inactivity.

---

5. Restart the application. The `grioqos -m` output changes accordingly:

```
21:01:12 20971520  1000 16.10 40.00 593.26 17.1  0.00 23.73  23    0   77
21:01:13 20971520  1000  4.80 40.00 632.81 17.8 15.82 15.82   0   70   99
21:01:14 20971520  1000  1.53 40.00 664.45 18.1 23.73  7.91   0  100  100
21:01:15 20971520  1000  5.93 40.00 688.18 18.3 31.64  0.00   0  100   66
21:01:16 20971520  1000  2.42 40.00 704.00 18.2 23.73  7.91   0  100   60
21:01:17 20971520  1000  6.82 40.00 727.73 18.3 31.64  0.00   0  100   64
21:01:18 20971520  1000  3.31 40.00 743.55 18.3 23.73  7.91   0  100   63
21:01:19 20971520  1000  7.71 40.00 767.29 18.4 31.64  0.00   0  100   63
```

There is a small initial burst as the token bucket is drained and GRIO throttles the application to 20 MB/s.

6. During the same run, we collect low-level QoS statistics. At the start of the run, use `-i` to display all of the intervals that are being monitored in the kernel:

```
$ grioqos -i 03041498-871c-1029-87e2-08006913a7f 1

IRIX64 octane 6.5 01062343 IP30 07/21/05

Filesystem: /mirror

App (6754151) 20.00 MB/s 03041498-871c-1029-87e2-08006913a7f7

-          interval   minbw   maxbw  lastbw  minio  maxio lastio
-                 -    MB/s    MB/s    MB/s     ms     ms     ms
21:00:38        1io       -       -       -  296.8 1004.3 1004.3
+               2io   29.32   32.89   32.89  402.2  967.8  967.8
+               3io   30.68   31.94   31.00  505.0  882.0  882.0
+               4io   31.01   31.38   31.38  611.6  788.4  788.4
+               5io   31.46   31.46   31.46  690.1  690.1  690.1
+               6io       -       -       -      -      -      -
+              10io       -       -       -      -      -      -
+             100ms   29.32   32.89   32.89  402.2  967.8  967.8
+             200ms   29.32   32.89   32.89  402.2  967.8  967.8
+             500ms   30.68   31.00   31.00  716.5  882.0  882.0
+            1000ms   31.46   31.46   31.46  690.1  690.1  690.1
+            2000ms       -       -       -      -      -      -
+            5000ms       -       -       -      -      -      -
+            1500io       -       -       -      -      -      -
```

There are 14 intervals being monitored for this stream. This sample was collected just after the application was started and only a small number of I/Os had been issued. There is insufficient data to compute some of these metrics and a number of samples are displayed as "-" .

7. Select two intervals (`500ms` and `2000ms`) and monitor them during the course of the run:

```
$ grioqos -I "500ms,2000ms" 03041498-871c-1029-87e2-08006913a7f 2

IRIX64 octane 6.5 01062343 IP30 07/21/05
Filesystem: /mirror

App (6754151) 20.00 MB/s 03041498-871c-1029-87e2-08006913a7f7
```

```
-          interval   minbw   maxbw  lastbw  minio  maxio lastio
-                 -    MB/s    MB/s    MB/s     ms     ms     ms
21:00:38      500ms       -       -       -      -      -      -
+            2000ms       -       -       -      -      -      -
21:00:40      500ms   30.37   31.93   31.48  479.0 1009.0  959.2
+            2000ms   31.46   31.46   31.46  789.4  789.4  789.4
21:00:42      500ms   18.91   32.31   20.14  479.0 1224.1 1224.1
+            2000ms   25.37   31.72   25.37  789.4 1057.5 1057.5
21:00:44      500ms   18.91   32.31   20.75  479.0 1588.4 1583.9
+            2000ms   19.74   31.72   20.38  789.4 1527.7 1527.7
...
```

As seen in the high-level metrics, there is an initial burst of I/O before the application is throttled by GRIO. The current bandwidth `lastbw` quickly stabilizes at around 20 MB/s. After the application is suspended in userspace, the low-level QoS statistics clearly record the interruption:

```
21:01:13      500ms    1.15   32.31   27.27  479.0 1609.3  554.7
+            2000ms    1.15   31.72    3.19  789.4 1594.4  816.3
21:01:15      500ms    1.15   34.95   31.96  479.0 1609.3  988.3
+            2000ms    1.15   33.08   33.08  788.7 1594.4  861.1
...
```

# GRIO API Overview

User processes communicate with the `ggd2` daemon using the following core library calls:



The process that initially reserves bandwidth by calling `grio_reserve()` or `grio_reserve_fd()` is referred to as the *owning process*. Any streams not already released when their owning process exits will be automatically released. Processes can share streams. The ownership of a GRIO stream is nontransferable.

## `grio_avail()`

Synopsis:

```
#include <grio2.h>

int grio_avail(
         const char *fs,
         grio_off_t *bytes, grio_msecs_t *msecs)

cc ... -lgrio2
```

The `grio_avail()` call returns the currently available guaranteed-rate bandwidth for a specified filesystem. The returned bandwidth is the qualified bandwidth of the filesystem minus bandwidth reserved for active GRIO streams and any bandwidth statically allocated for non-GRIO I/O. While `ggd2` temporarily allows unreserved

bandwidth to be used for servicing nonguaranteed-rate I/O, this bandwidth is reclaimed when a GRIO reservation is received and is therefore considered available.

For more information, see the `grio_avail`(3X) man page.

## grio_bind()

Synopsis:

```
#include <grio2.h>

int grio_bind(grio_descriptor_t fd, grio_stream_id_t *stream_id);

          cc ... -lgrio2
```

The `grio_bind()` call binds one or more open file descriptors to a GRIO stream. Once bound, all I/O to or from the file descriptors will receive the QoS guarantees of the stream.

Binding a file descriptor increments the reference count of a stream by 1. The file descriptor remains bound to the stream until it is either closed or explicitly unbound with `grio_unbind()`.

The file descriptor must be capable of GRIO I/O. That is, it must refer to an open file on an XFS or CXFS filesystem and be configured for direct I/O.

For more information, see the `grio_bind`(3X) man page.

## grio_get_stream()

Synopsis:

```
#include <grio2.h>

int grio_get_stream(
          grio_descriptor_t fd,
          grio_stream_id_t *stream_id);

cc ... -lgrio2
```

The `grio_get_stream()` call returns the ID of the stream to which it is bound.

For more information, see the `grio_get_stream`(3X) man page.

## grio_modify()

Synopsis:

```
#include <grio2.h>

int grio_modify(
          grio_stream_id_t *stream_id,
          grio_off_t *bytes, grio_msecs_t *msecs,
          int flags)

cc ... -lgrio2
```

The `grio_modify()` call changes the properties of an existing GRIO stream. You can increase or decrease reserved bandwidth by using this call.

---

**Note:** `grio_modify()` is a synchronous call and, when increasing a reservation, may block while bandwidth is reallocated. This delay can be in the order of 1 or 2 seconds and applications should be designed to accommodate this delay if necessary. While a call to `grio_modify()` is being processed, I/O to the stream continues uninterrupted at its existing rate.

---

For more information, see the `grio_modify`(3X) man page.

## grio_release()

Synopsis:

```
#include <grio2.h>

int grio_release(grio_stream_id_t *stream_id)

cc ... -lgrio2
```

The `grio_release()` call removes a GRIO stream ID from the system and releases the primary reference taken when it was created. When all remaining references to

the stream are removed, the stream will be destroyed and its associated bandwidth will be returned to the system.

The `grio_release()` call hides the stream. Attempts to bind new file descriptors using `grio_bind()` will fail with a return value of `ENOENT`. However, the QoS guarantees of the stream will remain in effect until all remaining bound file descriptors are either unbound or closed, and any in-flight I/O to the stream completes.

This behavior gives an application some flexibility in controlling the extent of a GRIO guarantee. For instance, by binding a file descriptor to a stream and immediately releasing the stream, an application can create a temporary reservation that persists for as long as the file descriptor remains open. Alternatively, if a process does not explicitly release a stream, the guarantee persists until that process exits.

For more information, see the `grio_release`(3X) man page.

## `grio_reserve()` and `grio_reserve_fd()`

Synopsis:

```
#include <grio2.h>

int grio_reserve(
          const char *path,
          grio_off_t *bytes, grio_msecs_t *msecs,
          int flags,
          grio_stream_id_t *stream_id)

int grio_reserve_fd(
          grio_descriptor_t fd,
          grio_off_t *bytes, grio_msecs_t *msecs,
          int flags,
         grio_stream_id_t *stream_id)

cc ... -lgrio2
```

The `grio_reserve()` and `grio_reserve_fd()` calls reserve guaranteed rate bandwidth to or from a GRIO-managed filesystem. If successful, they set up a GRIO stream in the kernel with the requested properties and return its stream ID:

- `grio_reserve()` makes a filesystem-level reservation. The target filesystem is identified with a path that must be either the filesystem mount point or the device special file on which it is located. Before guaranteed rate I/O can be performed, an open file descriptor must be bound to the new stream using `grio_bind()`.
- `grio_reserve_fd()` makes a file-level reservation. It takes an open file descriptor on the target filesystem. In addition to reserving bandwidth, the file descriptor is bound to the newly created stream. The file must therefore be suitable for guaranteed rate I/O and satisfy the normal requirements of `grio_bind`.

The requested bandwidth is specified as the number of bytes delivered every *msecs* milliseconds. *msecs* is referred to as the *reservation interval*. This value provides additional information to the GRIO scheduler about an application's ability to tolerate variation in I/O service time. (For example, an application request of 1MB every tenth of a second suggests a tighter requirement than 100 MB delivered every second, even though both requests describe the same average data rate.) GRIO uses this information as a hint only, and honors the expressed bandwidth over an implementation-defined scheduling interval.

`grio_release()` should be called when the stream is no longer required.

The process that creates a stream with these calls is said to be the *owning process*. Any streams not already released when their owning process exits will be automatically released. The ownership of a GRIO stream is not transferable.

GRIO streams are reference-counted. When created, a new stream has a reference count of 1. This primary reference remains until the reservation is released using `grio_release()` or the owning process exits.

A stream persists until its reference count drops to 0. Binding a file descriptor using `grio_bind()` adds a reference. Unbinding using `grio_unbind()` or closing a file descriptor removes a reference. In-flight I/O will also add references to a stream for short periods of time.

It is possible, and frequently useful, for a stream to persist after it has been released. For more information, see the `grio_release()` man page.

**Note:** Both `grio_reserve()` and `grio_reserve_fd()` are synchronous calls and may block while bandwidth is reallocated. This delay is referred to as the *stream creation latency*.

In the worst case, this delay can be in the order of 1 or 2 seconds, although it may be significantly less depending on the configuration of a particular GRIO deployment.

The following are strategies to minimize the impact of this behavior:

- Reserve bandwidth well ahead of time
- Perform reservations in a dedicated thread
- Reuse a reservation wherever possible
- Configure `ggd2` to keep a proportion of the available free bandwidth uncommitted using the `-r` option

For more information, see the `grio_reserve`(3X) and `ggd2`(1M) man pages.

## `grio_unbind()`

Synopsis:

```
#include <grio2.h>

int grio_unbind(grio_descriptor_t fd);

cc ... -lgrio2
```

The `grio_unbind()` call unbinds a file descriptor from its GRIO stream. Unbinding a file descriptor decrements the reference count of its stream by 1.

Once unbound, I/O to or from the file descriptor may continue, but will be scheduled as regular, non-guaranteed rate I/O.

For more information, see the `grio_unbind`(3X) man page.

# Glossary

**burstiness**

The extent to which the I/O data rate for an application or node tends to vary suddenly, over short intervals of time. Contrast with an application that issues its I/O **smoothly** to the filesystem, with little variation in the rate at which it submits I/O. For example, an application that buffered a data flow in its internal buffers then periodically flushed that data to disk through multiple I/O worker threads would produce a highly bursty data flow.

**ceiling allocation**

A node-level allocation for which GRIO ensures that the node receives **at most** the configured bandwidth.

**cluster volume**

An XVM volume configured for shared access via a CXFS filesystem

**cluster volume domain**

An XVM concept that refers to the entire set of configured cluster volumes

**device-level qualification model**

The maximum sustainable bandwidth is measured for each hardware component in the I/ O path, a user reservation for I/O resources is then processed for each hardware component, and is refused if any one component would become oversubscribed.

See also *filesystem-level qualification model.*

**distributed bandwidth allocator (DBA)**

The functionality of the `ggd2` daemon that periodically adjusts the amount of bandwidth allocated to the individual non-GRIO streams for its managed filesystems. The DBA is responsible for determining how unreserved bandwidth is distributed among the nodes with access to the filesystem.

**filesystem-level qualification model**

A qualified bandwidth for the entire filesystem is determined empirically by verifying that the required QoS is delivered to all of the applications running a range of realistic workloads.

The maximum sustainable bandwidth is measured across the entire filesystem under a realistic application workload.

See also *device-level qualification model.*

**floor allocation**

A node-level allocation for which GRIO ensures that the node receives **at least** the configured bandwidth. While there is any unallocated bandwidth, and the node is issuing I/O, `ggd2` will allocate it additional bandwidth.

**GRIO**

Guaranteed-rate I/O. *GRIOv1* is GRIO version 1, *GRIOv2* is GRIO version 2. GRIOv1 is a legacy product available on IRIX only.

**GRIO I/O**

I/O for applications that have made an explicit GRIO reservation.

See also *non-GRIO I/O.*

**GRIO server**

The CXFS server-capable administration node on which the active `ggd2` daemon is running. All cluster volumes are managed by this single instance of `ggd2`. There is one GRIO server per cluster.

**guarantee**

The assurance made by the system to a user process that it will deliver data from a storage device at the reserved rate regardless of any other I/O activity on the system or on other nodes within its cluster.

**jitter**

The variation in individual service times. As the storage system approaches saturation, service-time jitter will typically increase.

**local volume**

An XVM volume that is dedicated for the sole use of one node.

**local volume domain**

An XVM concept that refers to the entire set of configured local volumes

**non-GRIO I/O**

All buffered and system I/O other than I/O for applications that have made an explicit GRIO reservation.

See also *GRIO I/O*

**non-GRIO stream**

A dedicated system stream created for a filesystem by a node when `ggd2` begins managing that filesystem. All non-GRIO I/O issued by a node is automatically attached to and managed by this stream.

**owning process**

The user process that initially reserves bandwidth by calling `grio_reserve()` or `grio_reserve_fd()`.

**qualified bandwidth**

The maximum bandwidth that can be delivered by a filesystem (and the XVM volume on which it resides) in a given configuration under a realistic application workload such that all applications are delivered an adequate QoS.

**quality of service (QoS)**

The performance properties of a system service (such as worst-case bandwidth or I/O service time).

**reservation**

The set of QoS parameters requested by a user application. Reservation requests are forwarded to the `ggd2`(1M) bandwidth management daemon.

**reservation interval**

For GRIO functions where the reservation is expressed as a number of bytes delivered every number of milliseconds, that time in milliseconds is referred to as the *reservation interval*. It gives GRIO an indication as to the request's sensitivity to I/O jitter.

**stream**

The object within the kernel that encodes the reservation's QoS parameters and maintains the necessary scheduling and monitoring state required to fulfill the guarantee.

**stream creation latency**

The delay resulting when the `grio_reserve()` and `grio_reserve_fd()` calls block while bandwidth is reallocated.

# Index

**X**