Xsan Programming Guide

Storage > File Management

2006-05-23

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Figures and Tables

Introduction

Xsan is a high-performance storage area network (SAN) file system for Mac OS X and Mac OS X Server. It enables users to share centralized disk storage with multiple computers over Fibre Channel. With the Xsan file system installed, up to 64 Xserve or Power Mac systems can read and write to the same storage volume at the same time. Xsan features include the following:

- High-performance, 64-bit file system for SAN file sharing over Fibre Channel
- File-level locking for protected concurrent read/write access to shared volumes for up to 64 computers
- Metadata controller failover and Fibre Channel multipathing for high-availability, redundant configurations
- Flexible volume management for data consolidation, dynamic scaling of storage pools, and volume mapping
- "Affinities" for automatically directing files to specified storage pools—for example, source files are automatically written to the RAID 0 pool and processed files are automatically written to the RAID 5 pool
- Bandwidth reservation settings to reserve requisite SAN bandwidth for highest-priority applications
- User quotas for storage resource allocation
- Integration with any LDAP directory for centralized management of user and group access privileges
- Easy remote administration using the Xsan Admin utility for setting up and monitoring the SAN
- Compatibility with ADIC's StorNext file system for heterogeneous SAN environments

Programmers developing applications that take advantage of Xsan features should use this API. Xsan is particularly suited for applications used by professionals that collaboratively process audio and video data, IT departments that rely on SANs to consolidate and manage growing storage resources, and scientists that use computing clusters to work on computationally intensive projects that include large data sets.

The Xsan API uses `fcntl`(2) to send requests to and receive results from a Mac OS X kernel that includes Xsan. The API provides macros for use with `fcntl`(2) that allow you to allocate extent space for a file and get the list of extent space that has already been allocated for a file, get and set a file's affinity, enable and disable real-time I/O for a storage pool or file descriptor, get the real-time I/O parameters for a storage pool, get and set quota limitations for a user or group, and query the kernel for Xsan version information.

Organization of This Document

This book contains the following chapters:

- "Xsan Overview" (page 9) describes the terminology and concepts used in Xsan.
- "Xsan Tasks" (page 19) explains how to use the system calls to perform Xsan related tasks.

See Also

Refer to the following reference document for XSan:

- *Xsan Reference*

Xsan Overview

Xsan is a specialized file system technology for Mac OS X and Mac OS X Server that allows multiple computers to access centralized disk storage over Fibre Channel. Xsan delivers essential technology for building storage networks using Apple products, including Xserve RAID. This capability is critical for customers that manage large amounts of data or require simultaneous high-speed access to shared data.

Terminology

A **SAN** is a high-speed subnetwork of shared storage devices.

A **node** is a computer attached to a SAN.

The software that manages Xsan file system functions, such as file locking, space allocation, and data access authorization, is called the **metadata controller**.

Xsan **file system client** software runs on all nodes in the SAN and communicates with the metadata controller in order to provide Xsan services. The term **file system client** refers to a node that is running the Xsan file system client software.

The metadata controller uses **callbacks** to communicate with file system clients.

A redundant array of independent disks (**RAID**) device is a category of disk devices that combines two or more drives increased for fault tolerance and performance. There are several RAID levels:

- Level 0 provides **data striping**, where blocks of a file are spread across multiple disks, increasing performance; this level does not have any provisions that increase fault tolerance.
- Level 1 provides disk mirroring.
- Level 3 provides the data striping of Level 0 and also reserves a disk for storing error correction data, thereby increasing performance and fault tolerance.
- Level 5 provides data striping at the byte level and maintains stripe error correction information.

A **JBOD** (just a bunch of disks) is a disk that is not configured for RAID.

A **logical unit number** (LUN) is an aggregation of physical devices. Applications access LUNs through the special files in the system's `/dev/disk` directory. For RAID devices, a LUN is typically a RAID-5 with three or more physical drives making up the LUN. For JBOD devices, one JBOD is one LUN.

A **storage pool** is a grouping of LUNs that have the same characteristics. Another term for storage pool is **stripe group**. One or more storage pools form a mountable **volume**. The number of volumes hosted by a single Xsan metadata controller should not exceed eight.

Stripe depth is the number of disks that have been assigned to a storage pool.

The **stripe breadth** is the maximum amount of data that is read or written before switching to the next LUN in the storage pool. When the last LUN is reached, I/O operations go back to t he first LUN. This is how large logical I/O operations are broken down into stripes across multiple LUNs. For example, if the stripe breadth for a storage pool is set at 4 MB, each I/O operation on that storage pool is physically no more than 4 MB. A 16 MB I/O operation would be broken down into 4 physical I/O operations.

A **stripe line** is the stripe breadth multiplied by the number of LUNs in the storage pool. To maximize performance, make I/O requests that area stripe line in size.

For real-time I/O, **well-formed I/O** is I/O that is a stripe line in size. This size makes the best utilization of the disks in the storage pool and maximizes the transfer rate. For non-real-time I/O, well-formed I/O is I/O that is memory aligned (modulus 4 bytes), 512-byte sector aligned, and modulus sector sized.

A **block** is the smallest number of bytes that can be read or written.

Storage pools can be assigned one or more values, known as an **affinity identifiers**, and a file can be assigned one **affinity**. When a request is made to allocate space in a file for which the affinity has been set, the space is allocated from the storage pool that an affinity identifier that matches the file's affinity. For example, consider a SAN with some moderate performance JBOD LUNs and some high performance RAID-5 LUNs. By grouping the RAID-5 LUNs into the same storage pool and assigning them a specific affinity identifier, the developer can steer performance critical data to that storage pool. Files containing less critical data or files that do not have an affinity are assigned to the storage pool that consists of JBOD LUNs.

When a storage pool is in **real-time I/O mode**, file system clients that have processes that do non-real-time I/O must request a non-real-time I/O **token** Xsan throttles the speed of I/O of applications that are not in real-time mode so that their I/O does not interfere with real-time I/O. This document uses the term **gate** to describe processes or file descriptors that are not in real-time I/O mode and the term **ungated** to describe processes or file descriptors that are in real-time I/O mode.

An **extent** is a chunk of file data whose allocation is contiguous on a storage pool. A file's data may be stored in one or more extents. Information about an extent includes its file-relative starting byte offset, its file system starting byte offset, the file system ending byte offset, and the ordinal of the storage pool on which the extent resides. File system clients use **extent mapping tables** to load information about a file's extents. Loading extent information improves performance by eliminating a subsequent trip by the file system client to the metadata controller in order to retrieve extent information for the range mapped by an I/O request.

When there are two or more storage pools that have the same characteristics, an **allocation strategy** is needed. The strategy can be to round-robin files through the set of storage pools, balance the remaining space in the storage pools, or fill the first storage pool before going to the next storage pool.

A **disk file system**, such as a UFS or HFS+ file system, resides on the internal drives of a computer or on storage devices that are attached directly to the computer. A **network file system** allows data on internal drives or on directly attached drives to be shared with other computers on the network. Examples of network file systems include Apple Filing Protocol (AFP), Server Message Block (SMB), Common Internet File System (CIFS), or Network File System (NFS). A **distributed file system** is a blend of disk file system and network file system used to simplify data sharing through the creation of a single shared name space across a collection of servers. A **cluster file system** gives multiple computers simultaneous, very high-speed access to all shared data residing on an external, centralized storage pool. The storage pool typically consists of highly available RAID systems. Xsan is a cluster file system.

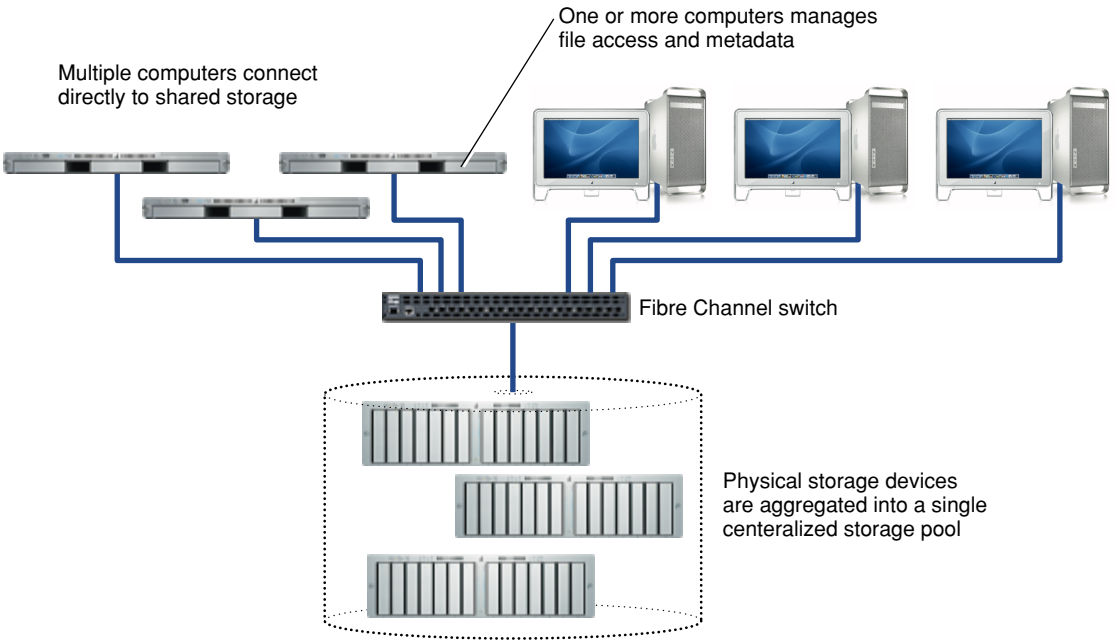
Storage Area Networks

The proliferation of digital content across organizations in virtually every industry is causing storage needs to grow exponentially. In some industries, such as video processing and biotechnology, storage needs double every year. Three basic types of architectures have evolved to meet these growing storage needs:

- **Direct-attached storage** (DAS). A DAS solution consists of disk drives or storage systems connected directly to a server or workstation. Typically, the storage system is connected to a PCI card in the host server or workstation through a Fibre Channel interface. The server formats the DAS using a compatible disk file system, such as UFS or HFS+. It shares the storage with other computers over Ethernet using a network file sharing protocol, such as AFP, SMB, CIFS, or NFS.
- **Network-attached storage** (NAS). A NAS is a specialized storage device that connects to a local area network and shares files with clients using network file sharing protocols, such as AFP, SMB, CIFS, or NFS. True NAS devices use a specialized server operating system having limited functionality.
- **Storage Area Network** (SAN). A SAN is a network whose primary purpose is to connect (typically over Fibre Channel) multiple storage devices with multiple computers. Unlike a NAS, which uses network file sharing protocols to deliver stored data to network clients, a SAN connects multiple clients directly to the storage network through Fibre Channel switch. Data traverses the SAN using a high-speed SCSI data transfer protocol. A SAN pools storage resources by creating a network of physical storage devices that can be managed as a single virtual storage entity. A SAN centralizes storage management, which simplifies administration and allows storage resources to be used more efficiently.

All three storage architectures benefit from continual improvements in the size and performance of disk subsystems. However, only SANs can scale to meet the capacity and performance demands of users in the design and print, video, and biotechnology industries. SAN solutions streamline access to shared storage, simplifying workgroup collaboration and improving workflow productivity. SAN technology also increases the flexibility and scalability of storage deployments, making it easier for administrators to respond to growing capacity and performance requirements.

Figure 1-1 SAN architecture



SAN Components

Like a local area network (LAN), SANs use software and hardware components to create a fabric of interconnected storage devices and computers. SAN components consist of the following:

- Storage hardware. Disk-based storage (such as Xserve RAID) is the primary storage media in any SAN solution. Many SANs also use a tape-based system for backup. Appropriate storage hardware should offer high-availability features to prevent data loss or downtime in the event of a disk failure, power outage, or other issue.
- Host bus adapter (HBA). An HBA is an adapter (usually a PCI card, such as Apple's 2GB Fibre Channel PCI card) that's installed in each node computer in a storage network an is used to connect the node to a Fibre Channel switch.
- Fibre Channel switch. This multiport device provides the interconnections required to connect multiple Fibre Channel storage devices to computer nodes. Today's SAN solutions use 1Gb or 2Gb (100MB/s or 200MB/s) Fibre Channel, an open standard technology for reliable, high-speed storage interconnectivity
- Fibre Channel cables. Fibre Channel cables physically connect all of the nodes in a storage network. Copper cables are most economical and used for short distances; optical cables can transmit stored data over many kilometers.
- SAN file system. SAN file system software is the key software component required for building a SAN. A SAN file system (such as Xsan) virtualizes multiple physical storage devices into a single volume and makes it visible to all computer nodes in the SAN. A SAN file system uses a controller to manage metadata, or data about the data, to track the physical location of files and provide file locking to prevent multiple users from accidentally writing to the same file at the same time.

SAN Deployment

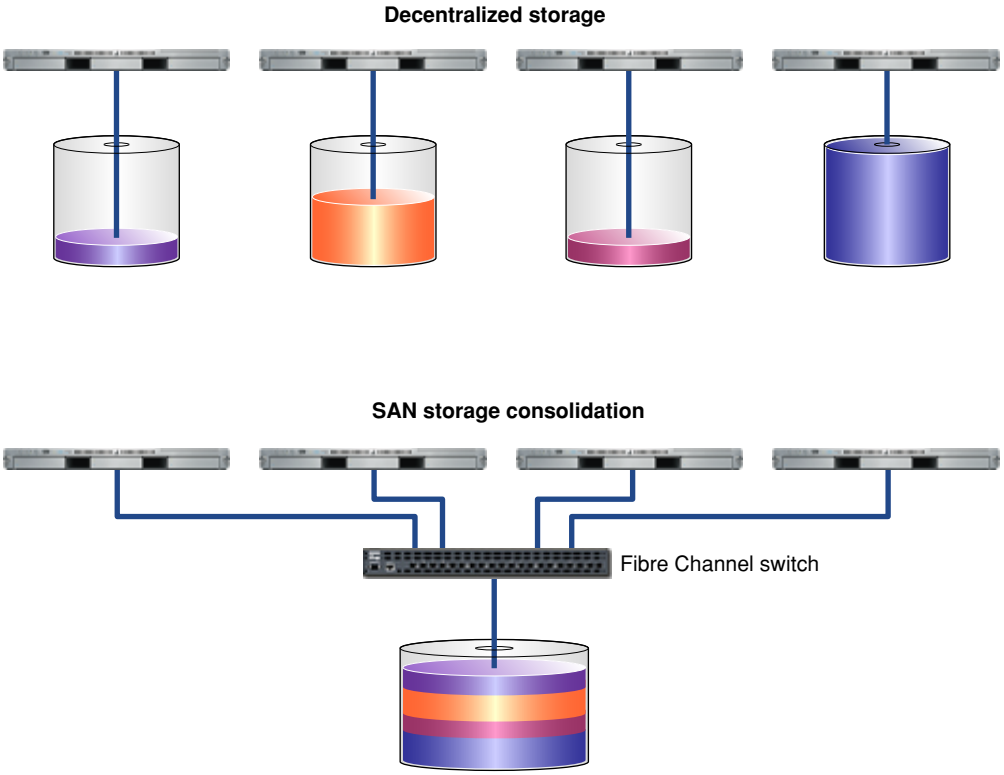
Xsan is designed to meet the needs of professional, education, and business users.

Storage Consolidation and NAS Replacement

Businesses in every vertical industry and government agencies are facing constant growth in data and more stringent government regulations for maintaining compliant records. This growth is placing more demand than ever on their storage infrastructures. Previously, most businesses and government agencies relied on a decentralized storage model with many disparate network-attached storage devices deployed throughout the organization—often one for every department or for each network application. This method offers little flexibility, however, and can result in inefficiencies as storage needs grow disproportionately. IT departments must purchase more storage to meet the growing needs of one application, even if existing storage devices may be being under utilized by other applications. This method also results in management inefficiencies, as multiple storage resources are maintained and allocated independently.

Figure 1-2 (page 13) compares traditional decentralized storage with the data consolidation that SAN technology provides. With traditional centralized storage, each server is connected to a separate storage device. Some storage resources may be under utilized while others are at or near capacity. With SAN technology, data is consolidated into a single storage pool that all computers in the SAN can directly access (through a Fibre Channel switch). This saves money by increasing the efficiency, flexibility, and scalability of an organization's existing storage resources.

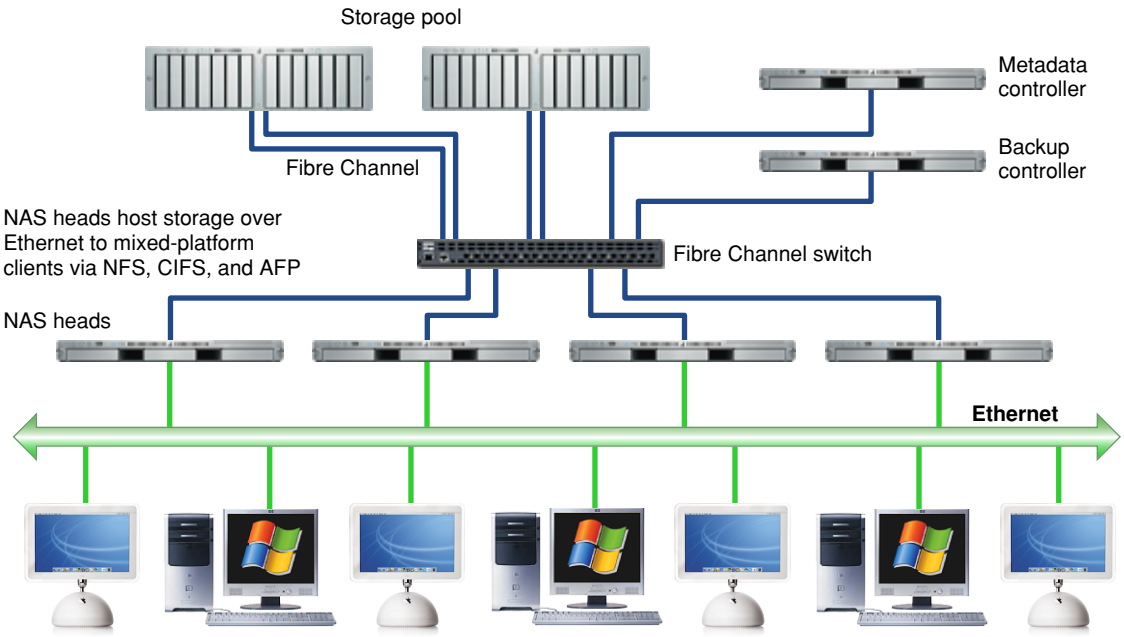
Figure 1-2 Decentralized storage compared with SAN storage consolidation



Today, IT departments look at data consolidation as a way to simplify administration and maximize utilization of existing storage resources. Using SAN technology, they can create a consolidated storage pool that can be centrally managed and allocated to any application, anywhere in the organization. By consolidating storage, reducing data duplication, and increasing the flexibility of existing storage hardware, investing in a SAN solution can reduce the storage requirements of organizations and lower IT costs.

- Traditional IT services. The ability to connect many servers to many storage devices, as shown in Figure 1-3, enables IT departments to consolidate data for more flexible and efficient utilization of storage resources. Deploying a dedicated storage network can make network services faster and more reliable, because one server isn't the single point of access for stored data. Xsan also makes it easy to scale out storage—adding more Xserve RAID systems can increase SAN capacity and performance.

Figure 1-3 Scalable NAS deployment



Educational institutions are experiencing the same growth in storage needs that are facing businesses today. Affordable storage and tools for efficiently consolidating and managing storage resources are critical in this area. Most educational institutions have moved beyond student information systems and now maintain all student records on disk, including student-generated data, such as multimedia projects. Many schools use server-based home directories to make students' work available to them from any computer on the network. These advances place enormous demand on storage resources and file sharing services. Xsan will be used in educational institutions primarily for data consolidation and creating scalable, affordable, and easy to maintain alternatives to decentralized NAS storage devices.

Collaborative Workflow Requiring High Bandwidth

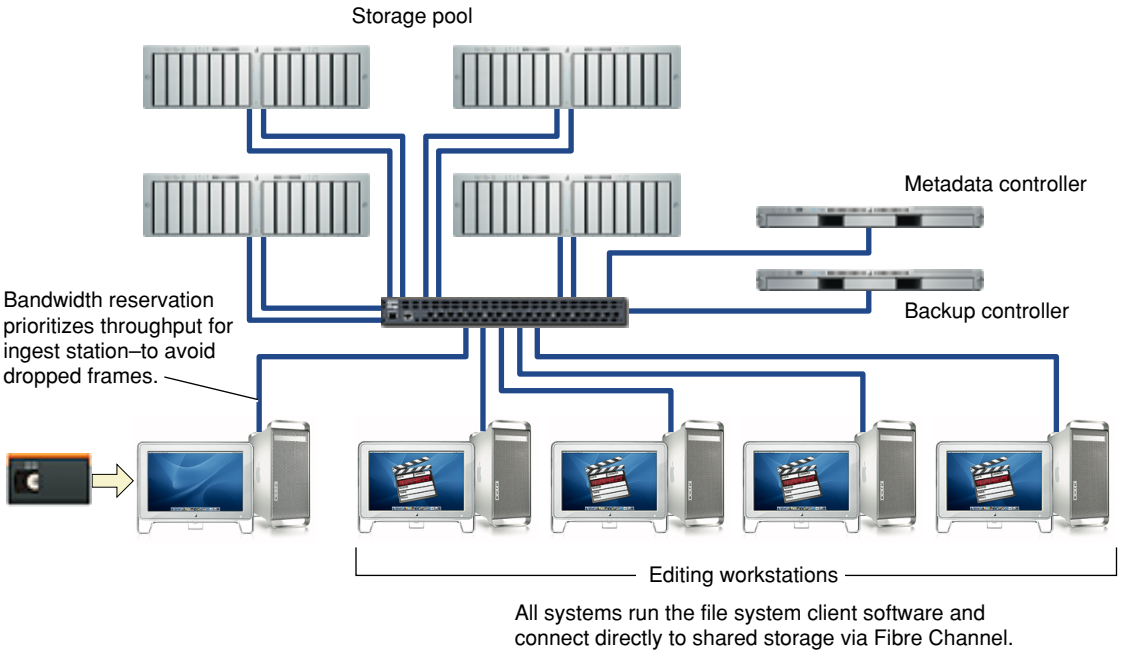
Professionals in the video and audio industries rely heavily on SANs for storing and sharing huge media files. The SAN—not the LAN—is the key to collaboration and efficient workflow for organizations that specialize in professional video and film editing, audio editing, and effects and motion graphics creation.

Capturing video or audio data from tape to digital storage system—the "ingest" process—demands consistent high-bandwidth performance. The SAN must be able to capture media from an ingest station (or multiple ingest stations) accurately and without dropping frames.

Currently, in post-production video and audio workflows, source media is rarely altered. Any alterations, such as color corrections, are noted in the project file and changed media is written as a separate render file. Each editing workstation usually stores these smaller-sized render files on the local drive. During the finishing process, these modifications are transferred on the LAN and merged to create a final master file.

Deployment of Xsan in professional audio and video processing environments eliminates data duplication between systems, which reduces storage requirements. Once data is captured to the SAN, it can be safely and securely stored, and made available to any authorized client workstations without ever being transferred between systems. Each client can work directly on the shared storage device, and multiple individuals can even work on the same file at the same time, streamlining collaboration because there's no waiting for files to traverse the network. Integration with LDAP directory services enables administrators to impose user-based access controls to secure digital assets. Figure 1-4 (page 15) shows how Xsan would be deployed in a video processing environment.

Figure 1-4 Xsan video production deployment



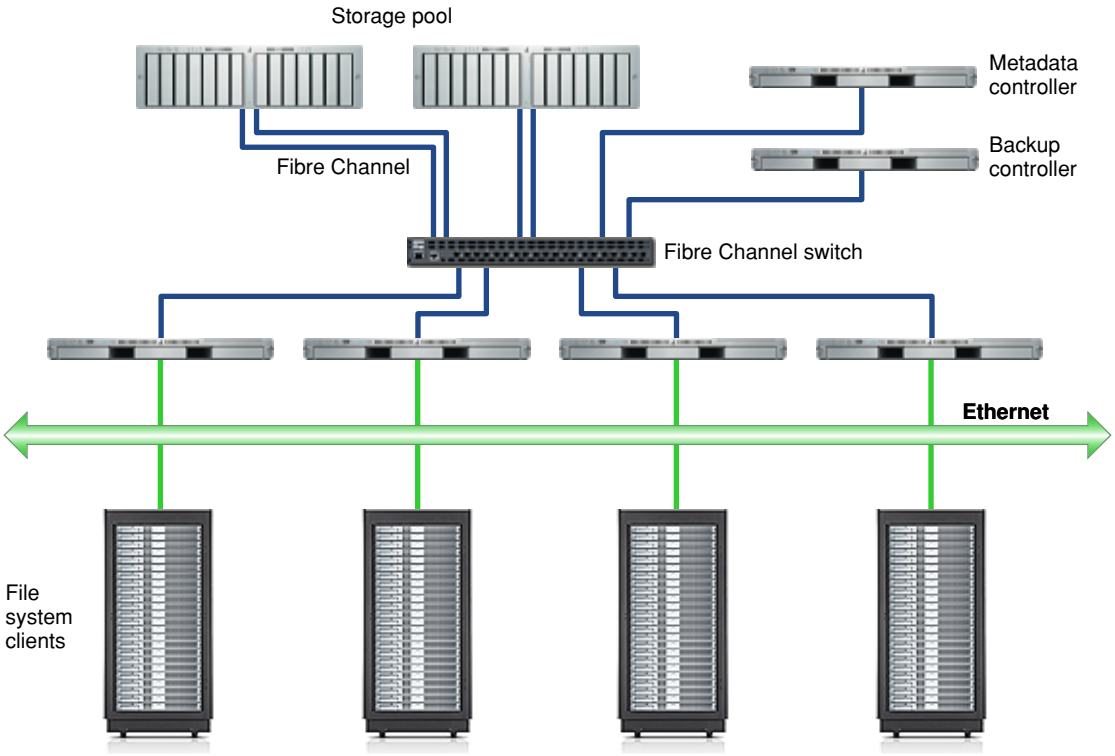
Computational Clusters

A computational cluster consists of multiple computers running an application against a single large data set. Computational clusters are increasingly being used in scientific computing, render farms, and other processor-intensive applications. The data set is typically hosted on a file server, and individual cluster nodes access the data using a network file system protocol, such as NFS, which limits scalability and performance. Xsan removes these limitations, providing faster, more scalable data sharing across small (64 nodes or less) and large clusters (more than 64 nodes).

In a small cluster, each node is connected directly to the SAN via Fibre Channel and all nodes can mount the same Xsan volume and read directly from the same files, thereby eliminating data replication.

Large clusters use a variation of NAS-SAN integration. Such clusters typically have one or more head nodes that each manage a portion of the cluster. For example, a large cluster may use one head node for every 25 to 50 cluster nodes. In this case, Xsan handles data management at the head node. Each of the head nodes in the cluster is connected directly to the SAN using a Fibre Channel HBA. These nodes access the data sets directly from the SAN and distribute the data to their nodes using network file system protocols, such as NFS. This method of accessing and distributing data is faster and uses less network bandwidth across the cluster than typical NAS solutions. At the end of the processing run, the processed data is returned to the head nodes, which in turn write directly to the shared SAN volume. Figure 1-5 (page 16) illustrates the layout of a large computational cluster.

Figure 1-5 Large computation cluster



Xsan Components

Xsan software must be installed on each computer, or node, in an Xsan SAN. Xsan software includes two components:

- **metadata controller**—this software manages SAN file system functions such as file locking, space allocation, and data access authorization. During setup, the administrator designates one computer as the metadata controller for the SAN. To maximize SAN availability, additional systems can act as failover metadata controllers that take over in the event that the primary metadata controller is incapacitated by a hardware or software failure.

- **file system clients**—this software runs on all other systems participating in the SAN, running side-by-side with other native file systems that the operating supports, such as HFS+ and UFS. The Xsan file system client software communicates with the metadata controller securely over Ethernet, enabling each client to directly access shared disk space on the SAN.

Xsan also includes an administration utility that can be installed on any Mac OS X system to enable secure remote setup, management and monitoring of the Xsan SAN, as well as a command line management utility.

Xsan Tasks

The Xsan API macros for use with the `fnctl`(2) system call to complete the following tasks:

- get and set a file's affinity
- allocate space in a file and get information about a file's allocated space
- enable and disable real-time I/O for a storage pool and individual files
- get and set quota limits for users and groups

Before You Start

The next few paragraphs describe some things you should know about this API before attempting any of the tasks.

This API uses the `fcntl`(2) system call to send requests to the Xsan system. With a few exceptions, each `fcntl`(2) call takes as parameters a file descriptor, a macro name that identifies the type of request, and a union of a request and a reply structure. Some requests don't require a request structure and some requests don't require a reply structure.

Except where noted, zero is returned on success. Failure is indicated when an `fcntl` (2) call returns –1 and `errno` is set to indicate the error.

All offsets and sizes are given in byte terms and are rounded up, if necessary, to the next Xsan file system block size.

For requests that return variable length buffers (such as extent lists), the request returns the total number of items available, as well as the number of items returned by this particular request. If more data is available than can be returned in the user's buffer, as much as possible will be copied into the buffer, and no error will be returned. For subsequent requests, the user can specify a different starting location (ordinal for storage pools, starting offset for extents). If the list changes while the list is being retrieved, inconsistent results may be returned. When there are no more entries available, `ENOENT` is returned.

Working With Real-Time I/O

Real-time I/O mode is a major feature of Xsan. When a storage pool is in real-time I/O mode, non-real-time I/O requests are gated so they do not hinder the speed of real-time I/O operations. The activity of processes doing real-time I/O is not monitored or scheduled in any special way to ensure that real-time I/O processes receive the bandwidth they requested. Real-time I/O processes are responsible for gating their own I/O requests to the desired rate, such as the frame rate for a specific video format.

The Xsan real-time I/O implementation is storage-pool-centric; all configuration and operations act on a specific storage pool. This allows storage pools with differing access characteristics to use different real-time settings. When a file system client successfully requests a certain amount of real-time I/O, the metadata controller sends a callback and a token to all connected clients to inform all connected file system clients of the amount of non-real-time I/O remaining and available for access. The token can also be thought of as a capability that allows the client to perform I/O without contacting the metadata controller.

In both the file system client and the metadata controller, real-time I/O is implemented as a state machine. Each state has a set of events and allowable actions. In this document, the term "state" refers to the internal state machine.

Applications that operate in real-time I/O mode do so by opening a file using the `open`(2) system call and obtaining a file descriptor.

Real-time I/O is configured in the metadata controller configuration file, and no configuration is required on file system clients. There is a real-time I/O tuning parameter that can be specified when a file system client mounts a file system.

For the purposes of determining bandwidth rates, well-formed real-time I/O is the stripe breadth multiplied by the number of LUNs in the storage pool in size. This size makes the best utilization of the disks in the storage pool and maximizes the transfer rate. Internally, non-real-time I/O is tracked by number of I/O operations per second. An I/O operation is a minimum of a file system block size, and a maximum of the file system block size multiplied by the stripe breadth, as shown in the following comparison:

```
FsBlockSize <= I/O <=(FsBlockSize * StripeBreadth)
```

Typically, it is easier to qualify an I/O subsystem in terms of sustainable megabytes per second. However, internally, the file system tracks everything on an I/O per second basis. Note that the file system only tracks and gates non-real-time I/O. An I/O is a minimum of the file system block size and is typically the point at which the file system hands the request off to the disk driver.

The file system counts the number of non-real-time I/O operations that occurred during a given second. If the count exceeds the amount of I/O that has been allotted for non-real-time I/O operations, the I/O request is pended until non-real-time I/O becomes available again (typically in the next second). Non-real-time I/O operations are honored in first in, first out (FIFO) fashion, and no priority is assigned to them.

To convert between I/O operations and megabytes per second, Xsan uses a formula that quantifies I/O as well-formed. The reason for this is due to the way in which many video applications make real-time I/O requests. To optimize the disk subsystem, real-time I/O operations are well-formed in order to saturate the disks. In Xsan terminology, this would be an I/O that covers all of the disks in a stripe. This can be expressed as follows:

```
ios_sec = mb_sec / (StripeBreadth * StripeDepth * FsBlockSize);
```

For example, with a file system block size of 4KB, a stripe breadth of 384, and a stripe depth of 4, the equivalent number of I/O operations per second for each well-formed I/O would be 216 MB per second / (384 * 4 * 4KB). This is equivalent to 221184 KB per second / 6144KB = 36 I/O operations per second.

All storage subsystems are different, so the user must qualify the I/O subsystem and determine the maximum amount of I/O bandwidth that is available. Xsan relies on the correct setting in the configuration file. If the storage system changes, for example, because a new disk array is installed, the user must re-qualify the I/O subsystem to determine the amount of bandwidth that is now available. This amount is specified in the metadata controller configuration file. The user can also specify the minimum amount of bandwidth to be provided to non-real-time I/O processes.

To maximize performance, most real-time I/O requests are made a stripe line at a time. Non-real-time I/O operations are a minimum of a file system block in size.

The calculations of available real-time and non-real-time I/O are discussed in the section "Calculating Available Real-Time I/O Operations" (page 24) and "Non-Real-Time I/O Tokens" (page 27), respectively.

Configuring Real-Time I/O

Real-time I/O is configured in the configuration file on the metadata controller located at `/Library/Filesystems/Xsan/config`. No configuration is required on file system clients, but a `mount` option is available on file system clients for performance tuning.

Configuring the Metadata Controller

The five keywords listed in Table 2-1 (page 21) configure real-time I/O in the metadata controller configuration file.

Table 2-1 Real-time I/O configuration keywords

Keyword	Meaning	Default
`Rtios`	Maximum number of real-time I/O operations of any size allowed on the storage pool during any one second period	Zero (no real-time I/O)
`Rtmb`	Maximum number of real-time megabytes per second allowed on the storage pool during any one second period.	Zero (no real-time I/O)
`RtiosReserve`	Amount to reserve in I/O operations per second from the maximum allowed for non-real-time I/O operations. This value must be greater than the equivalent of one megabyte per second or the amount that can be transferred to a single stripe line, where a stripe line is the stripe breadth multiplied by the number of LUNs in the storage pool.	Equivalent of one megabyte per second
`RtmbReserve`	Amount to reserve in megabytes per second from the maximum allowed for non-real-time I/O operations. This value must be greater than one.	One megabyte per second
`RtTokenTimeout`	Time, in seconds, to wait for file system clients to respond to a token callback.	2 seconds

All keywords in the metadata controller configuration file are case-insensitive.

Rtios and Rtmb Keywords

One of `Rtios` or `Rtmb`, or both may be specified in the metadata controller configuration file. Both keywords refer to the total amount of sustained bandwidth available on the disk subsystem. Any I/O operations, whether real-time or non-real-time, are ultimately deducted from this overall limit. If both `Rtios` and `Rtmb` are specified, the lower limit is used to throttle non-real-time I/O. For example, setting `Rtios` to 2048 and `Rtmb` to 10

specifies that the storage system can support a maximum of 2048 I/O operations per second, aggregate among all file system clients at any instant, or 10 megabytes per second, whichever is lower. If `Rtios` and `rtmb` are not specified, real-time I/O is not available on the storage pool.

Specifying `Rtmb` in the metadata controller configuration file is only recommended if all of the I/O operations are well-formed in size (that is, the stripe breadth multiplied by the number of LUNs in the storage pool). If not, using the well-formed I/O calculation to convert between megabytes per second and I/O operations per second can lead to unexpected results.

RtiosReserve and RtmbReserve Keywords

The `RtiosReserve` and the `RtmbReserve` keywords allow the administrator to increase the amount of bandwidth reserved for non-real-time I/O. This bandwidth is shared by all non-real-time applications on each client.

The default is one megabyte and cannot be decreased. If it were possible to configure zero I/O operations per second for non-real-time I/O, a system could block with many critical file system resources held waiting for I/O to become available.

If values for both the `RtiosReserve` and the `RtmbReserve` keywords are specified, the lower of the two amounts is used.

RtTokenTimeout Keyword

The `RtTokenTimeout` keyword configures the amount of time that the metadata controller waits for file system clients to respond to callbacks. For example, if the metadata controller fails, file system clients attempt to re-establish their connections to the metadata controller when the metadata controller becomes available again. In most SANs, the default setting of two seconds is sufficient. To cause the metadata controller to wait longer for callback responses and avoid unnecessary timeouts, this value is typically increased for SANs that have a mixture of client machine types (Mac OS X, Linux, Windows NT, and Irix) whose differing TCP/IP characteristics cause re-connection times to vary or for SANs that have more than 32 clients.

If a file system client times out on a token retraction, the file system client that make the original request receives an error from the metadata controller that includes the IP address of the offending client. This information is logged to `syslog` and to the desktop on Windows clients, which can help in diagnosing reconnect failures and in determining if the setting for the `RtTokenTimeout` keyword should be increased.

Sample Configuration

For this sample configuration, consider a video playback application that requires a constant rate of 186 megabytes per second in order to display images correctly and without dropping any frames. The application gates itself; that is, it requests I/O at a rate to satisfy the requirements for correctly displaying an image.

In this example, assume the I/O subsystem has been qualified at 216 megabytes per second. The file system block size is 4KB. The disk subsystem is actually a large RAID array that internally maps many drives to a single LUN. There are four LUNs in the storage pool; each LUN is optimized for a 1.5 megabyte transfer. This corresponds to the following in the metadata controller configuration file:

```
[StripeGroup MyStripeGroup]
StripeBreadth 384
Node CvfsDisk0 0
Node CvfsDisk1 1
Node CvfsDisk2 2
Node CvfsDisk3 3
```

```
Rtmb 216
```

This example assumes there is only one storage pool for user data in the file system. There may be other storage pools for metadata and journal that are not shown.

Configuring the File System Client

The only available file system client configuration option controls the length of time for which a non-real-time I/O token is valid. When a file system client obtains a non-real-time I/O token from the metadata controller, the token gives that client access to a specific amount of non-real-time I/O. If the file system client is inactive for a period of time, the token is relinquished and the non-real-time I/O is released back to the metadata controller for distribution to other file system clients.

The length of time for which a non-real-time I/O token is valid is controlled by the `nrtiotokenhold` option to the `mount` command. The default is 60 seconds, which means that after 60 seconds in which no non-real-time I/O occurs on the storage pool, the non-real-time token times out and the non-real-time I/O is released back to the metadata controller.

The timeout period should be specified in five second increments. If not, the timeout period is automatically rounded up to the next five second interval. If the `syslog` level is set to debug, the file system client writes its `mount` parameters in the `syslog` file so that the timeout period can be noted.

Enabling Real-Time I/O Mode

Initially, all storage pools in the file system are in non-real-time I/O mode, and all processes make their requests directly to the I/O subsystem without being gated.

The `F_SETRTIO` request enables real-time I/O mode. Depending on the type of file descriptor that is passed to the `F_SETRTIO` request, the `FSETRTIO` request places an entire storage pool or an individual file within a storage pool in real-time I/O mode.

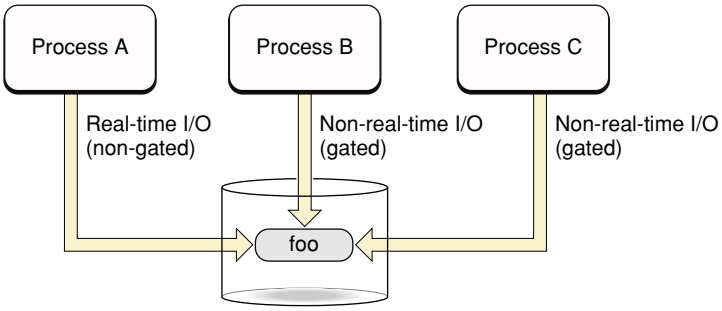
The `F_SETRTIO` request accepts two different types of file descriptor:

- A file descriptor for the root directory. When the file descriptor passed to an `F_SETRTIO` request is for the root directory and the `RT_SET` flag is set, the entire storage pool is put into real-time I/O mode, but no specific file descriptor is tagged as being ungated. The application is responsible for making a subsequent `F_ENABLERTIO` request on a specific file descriptor. The real-time I/O parameters that were specified when the `F_SETRTIO` request was made apply every file descriptor opened on a file on the storage pool for which an `F_ENABLERTIO` request is made.
- A file descriptor for a regular file. When the file descriptor passed to an `F_SETRTIO` request is for a regular file and the `RT_SET` flag is set, the result is equivalent to making an `F_SETRTIO` request on the root directory followed by an `F_ENABLERTIO` request on a file descriptor for a regular file in that storage pool.

Sharing Access to Files

Real-time I/O is enabled at the file descriptor level, not the file level, so different processes can share a file in real-time I/O and non-real-time I/O mode at the same time. As shown in Figure 2-1, sharing allows a real-time process to perform ingest of material (such as video data) at the same time that non-real-time processes perform other operations on a file.

Figure 2-1 Real-time I/O and non-real-time I/O share access to a file



In Figure 2-1, process A has ungated access to the file `foo`. Process B and process C are accessing the same file, but the file system client gates their I/O accesses.

Sharing Disk Resources

In this example, consider the case of a full-featured disk subsystem (Redundant Array of Very Expensive Disks, or RAVED) using a very high speed interconnect. Many clients can share these disk arrays, but it is sometimes desirable to limit a file system client's access to the array. Real-time I/O provides a mechanism for "political bandwidth management" so that no one file system client can consume all bandwidth resources.In such a scenario, the storage pool is always in real-time I/O mode. Each file system client has a token specifying the number of permissible non-real-time I/O operations per second. Unfortunately it is not possible to assign a different non-real-time I/O limit to different clients. The foundation of such an approach is a simple program that puts the storage pool into real-time I/O mode as soon as the metadata controller is up and servicing requests. When the storage pool enters real-time I/O mode, the bandwidth as specified in the metadata controller's configuration file is shared by all clients. In cases such as this, the real-time limit (`rtios` or `rtmb`) is calculated to be the total bandwidth desired to each file system client multiplied by the number of possible file system clients. As each file system client attempts to access the disk subsystem, it obtains a non-real-time I/O token. The metadata controller sends out callbacks adjusting down the amount of available bandwidth. No one file system client is allowed to exceed the threshold specified by the non-real-time I/O token, thereby assuring fairness to all file system clients.

Calculating Available Real-Time I/O Operations

When the metadata controller receives a request for real-time I/O, it considers the amount of real-time I/O that is being held in reserve. The metadata controller uses the reserve amount as a soft limit that it will not exceed. The calculation for available real-time I/O is as follows:

```
avail_rtio = rtio_limit - rtio_current;
avail_rtio -= rtio_reserve;
```

All internal calculations are made in terms of I/O operations per second.

Ungated Non-Real-Time I/O

Setting the `RT_NOGATE` flag when making an `F_SETRTIO` request causes I/O with the specified file descriptor request to be ungated without using any amount of real-time I/O or non-real-time I/O. This is useful for infrequently accessed files, such as index files, that should not be counted against available non-real-time I/O. System designers typically allow for some amount of overage in their I/O subsystem to account for ungated files.

Getting the Requested Amount of Real-Time I/O

Setting the `RT_MUST` flag when making an `F_SETRTIO` request causes the `F_SETRTIO` request to fail if the requested amount of real-time I/O cannot be provided. System designers that require a specific amount of real-time I/O should use the `RT_MUST` flag, and, if necessary, repeat the request until the requested amount of real-time I/O bandwidth becomes available.

Disabling Real-Time I/O Mode

Once enabled for a storage pool, real-time I/O mode remains in effect until a process explicitly makes an `F_SETRTIO` request with the `RT_CLEAR` flag set on the file descriptor for the root directory. Real-time I/O mode is implicitly disabled if the file system is unmounted or the system is rebooted.

Once enabled for an individual file descriptor, real-time I/O mode remains in effect until the process passes the file descriptor to `RT_SETRTIO` with the `RT_CLEAR` flag set, passes the file descriptor to `F_DISABLERTIO`, or closes the file. In any case, all real-time I/O that was allocated to the file descriptor is released back to the system. However, if multiple file descriptors are open on the file, only the last close of the file triggers the releasing action. This is because the file system is not informed of the close until the last close of a file.

The file system client automatically issues a call to the metadata controller with the specifying the amount of real-time I/O that is being released. The release of real-time I/O for a file causes the metadata controller to issue callbacks to all file system clients to adjust the amount of bandwidth available to them.

If the metadata controller cannot be reached for some reason, the request is enqueued on a daemon and the process that is closing the file is allowed to continue. In the background, the daemon attempts to inform the metadata controller that the real-time I/O has been released.

Callbacks

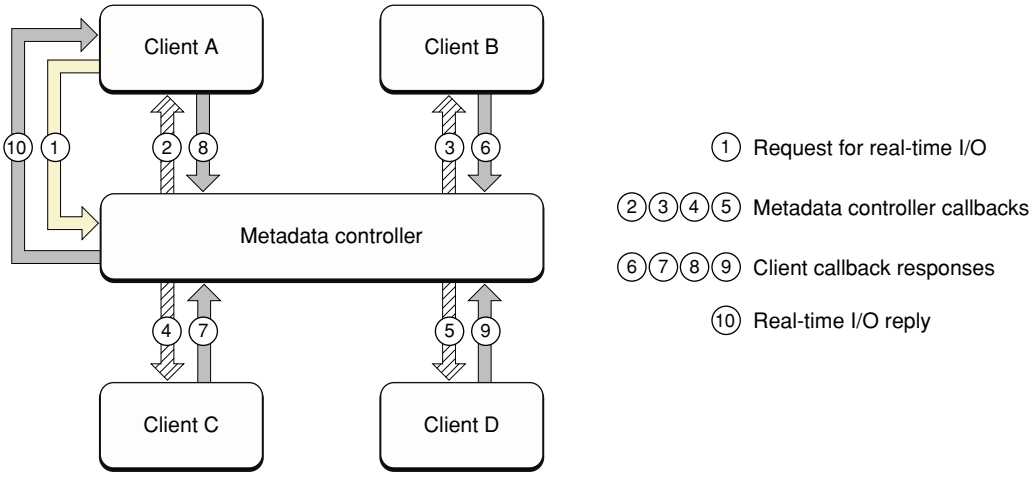
The metadata controller uses callbacks and tokens to communicate with file system clients. A **callback** is an unsolicited message from the metadata controller to a file system client requesting that the client adjust its non-real-time I/O parameters. The callback is accompanied by a **token** that specifies the amount of non-real-time I/O available on the storage pool.

Transition from Non-Real-Time I/O to Real-Time I/O Mode

Initially, all storage pools in a file system are in non-real-time I/O mode. When the metadata controller receives the initial request for real-time I/O on a storage pool, it first issues callbacks to all file system clients informing them that the storage pool is now in real-time I/O mode. The token that accompanies the message specifies that no non-real-time I/O is available. File system clients must now obtain a non-real-time I/O token before they can do any non-real-time I/O on the storage pool.

After sending out all callbacks, the metadata controller sets a timer based on the value of the `RtTokenTimeout` keyword. If each file system client responds to its callback within the value of `RtTokenTimeout`, the request for real-time I/O succeeds, and a reply indicating success is sent to the requesting client. Figure 2-2 summarizes this process.

Figure 2-2 Successful transition to real-time I/O mode



In step 1 of Figure 2-2, a process on client A requests some amount of real-time I/O. This is the first request, so the metadata controller issues callbacks to all connected file system clients (steps 2 through 5) informing them that the storage pool is now in real-time I/O mode. The file system clients respond to the metadata controller in steps 6 through 9. When all of the file system clients have responded, the metadata controller sends a reply to the file system client that made the original request (step 10) indicating success.

The next section, "Token Retraction" describes what happens when one or more of the file system clients do not respond to the callbacks sent in steps 2 through 5.

Token Retraction

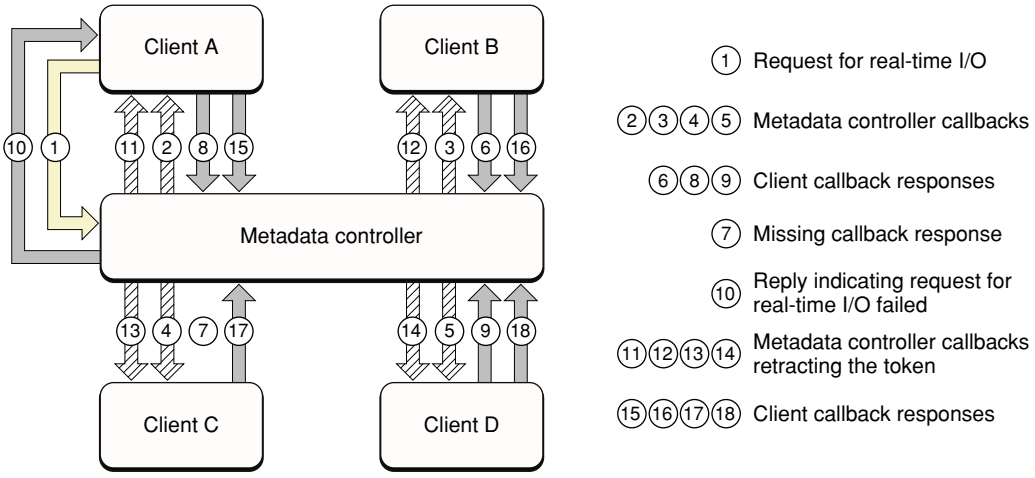
If the `RtTokenTimeout` timer expires before one or more clients has not responded, the metadata controller must assume that a serious problem has occurred. The metadata controller enters the token retraction state and does not honor any real-time I/O or token requests until it receives positive acknowledgement from all file system clients to which it originally sent callbacks.

In token retraction state, the metadata controller sends a response to the requesting file system client with the IP address of the first client that did not respond to the callback. This response allows the requesting client to log the error and the IP address so that system administrators can start to diagnose the failure. The metadata controller then sends callbacks to all file system clients that it previously sent callbacks to, retracting them to their original state, and it sets the storage pool back to its original state — in this case, non-real-time I/O mode.

If a file system client does not respond within the value of the `RtTokenTimeout` keyword, the token retraction callbacks are retracted and sent out again. This process is repeated until all file system clients respond. During this time, real-time I/O requests are not honored and are enqueued.

Figure 2-3 shows the flow of callbacks and responses in the token retraction state.

Figure 2-3 Token retraction state



In Figure 2-3, client A requests some amount of real-time I/O, as it did in Figure 2-2. However, in Figure 2-3, client C did not respond to the initial callback in time (step 7). The metadata controller returns a failure code as its reply to client A's initial real-time I/O request and sends callbacks to all file system clients indicating the storage pool is no longer in real-time I/O mode (steps 11-14). In the example, client C responds to the second callback (step 17), so the metadata controller stops sending out callbacks. The storage pool is returned to non-real-time I/O mode.

Note that token retraction sequence can have an unintended effect on soft-mounted file systems. If a request times out because other file system clients are not responding, failure is returned to the application, and then at some point the metadata controller is able to process the real-time I/O request successfully, the storage pool is put into real-time I/O mode after the original requester has received an error code. The metadata controller and file system clients log their actions extensively to `syslog`, so if this situation arises, it can be detected.

Non-Real-Time I/O Tokens

A token grants a client some amount of non-real-time I/O for storage pool and is encapsulated in a callback message from the metadata controller. Initially, no tokens are required to perform non-real-time I/O. Once a storage pool is put into real-time I/O mode, the metadata controller sends callbacks to all clients informing them that they now need a token to perform any non-real-time I/O, as described in "Transition from Non-Real-Time I/O to Real-Time I/O Mode" (page 25).

The file system client requests a non-real-time I/O token from the metadata controller when the client receives its next non-real-time I/O request. Once a file system client has a token, it can perform as much I/O per second as is allowed by that token until the token expires, and it does not need to contact the metadata controller on every I/O request.

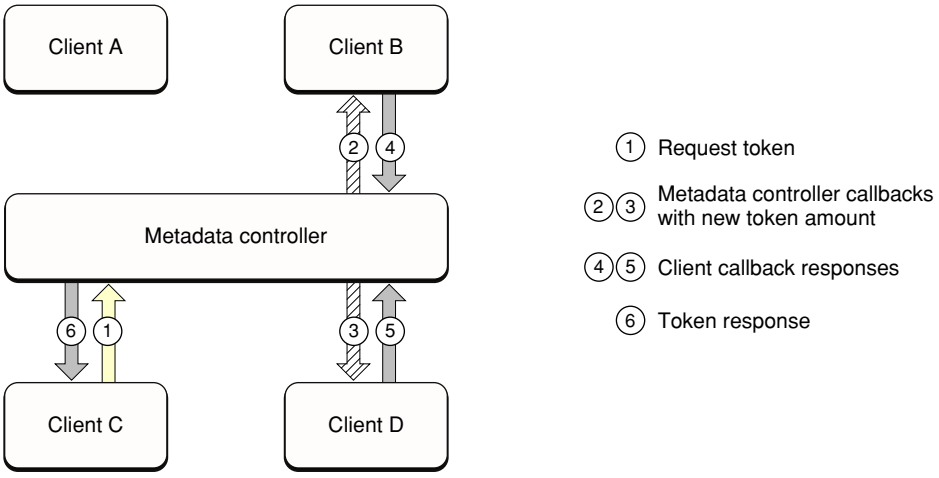
The metadata controller calculates the amount of available non-real-time I/O bandwidth using the following formula:

```
avail_nrtio = rtio_limit - rtio_current;
avail_nrtio /= current_num_nonrtio_clients + 1;
```

In the above calculation, the amount of the reserve parameter (`RtiosReserve` or `RtmbReserve`, whichever is lower) has already been subtracted from the amount of existing real-time I/O (`rtio_current`). As each file system client requests a non-real-time I/O token, the number of current non-real-time I/O file system clients (`current_num_nonrtio_clients`) increases and the amount of available non-real-time I/O decreases.

Each time the amount of available non-real-time I/O changes, the metadata controller sends a callback to each file system client that has an existing non-real-time I/O token, as shown in Figure 2-4.

Figure 2-4 Non-real-time I/O token adjustments



In Figure 2-4, assume the storage pool is already in real-time I/O mode as a result of a `F_SETRTIO` request from client A. Clients B and D are doing non-real-time I/O to the storage pool and each has a token specifying the amount of available non-real-time I/O. Then in step 1, client C requests a non-real-time I/O token. In steps 2 and 3, the metadata controller sends callbacks to clients B and D specifying the new token amount. The file system clients respond in steps 4 and 5, acknowledging the new token amount. In step 6, the metadata controller sends the new token to client C. Notice that in steps 2 and 3, the metadata controller only sends callbacks to file system clients that already have non-real-time I/O tokens.

Failure Scenarios

Two major failure scenarios can affect real-time I/O:

- metadata controller failures
- file system client failures

For metadata controller and file system client failures, Xsan attempts to return to the state it was in before the failure without manual intervention.

Metadata Controller Failures

If the metadata controller crashes, become unavailable due to a network outage, or is stopped, there is no immediate affect on real-time (ungated) I/O. As long as there is no need to contact the metadata controller, for example to update an attribute or make an extent request, I/O proceeds as if the metadata controller

were still available. If there is a need to make real-time I/O requests, the request hangs for 5 seconds. All other Xsan requests hang as controlled by the options that were specified to the `mount` command when the file system was mounted.

Non-real-time I/O is pended until the file system client reconnects to the metadata controller. The reason for making non-real-time I/O pended is that because the storage pool is in real-time I/O mode, there is no way to know when non-real-time I/O parameters change while the metadata controller is disconnected. When the file system client reconnects to the metadata controller, the file system client must re-request any real-time I/O that it requested previously. This is because the metadata controller does not log real-time I/O parameters across reboots. Therefore, it is up to file system clients to inform the metadata controller of the amount of required real-time I/O and to restore the metadata controller to its state as it was before the failure. In most cases, this strategy results in the amount of real-time and non-real-time I/O being exactly the same as it was before the failure. The exception is when the aggregate of processes request more real-time I/O than is configured, a condition known as **oversubscription**. Oversubscription comes about when, before the crash, processes requested more real-time I/O than was actually available so the metadata controller adjusted the requested amounts down. When oversubscription occurs, it is not deterministically possible to re-create the amount of real-time I/O exactly as it was before the failure. Therefore, if the amount of real-time I/O is required to be the same across reboots of the metadata controller or metadata failover, all processes should use the `RT_MUST` flag to ensure they get the amount of real-time I/O bandwidth that they need, both at the time and in case a recovery from a metadata controller crash is required.The process of each file system client re-requesting real-time I/O is exactly the same as the initial process of requesting real-time I/O. When each file system client re-establishes its real-time I/O parameters, non-real-time I/O processes are allowed to request non-real-time I/O tokens. Several seconds may elapse before the SAN settles back to its previous state. It may be necessary to adjust the value of the `RtTokenTimeout` parameter in the configuration file on the metadata controller to allow enough time for file system clients that are slow to reconnect to reconnect.

If a metadata failover is available, the clients reconnect to the metadata failover and reestablish their real-time I/O requirements as if the primary metadata controller had just come back online.

File System Client Failures

When a file system client disconnects from the metadata controller abruptly (a crash or a network outage) or in a controlled manner (an unmount), the metadata controller releases the file system client's resources back to the SAN. If the file system client had real-time I/O in progress on the storage group, that amount of real-time I/O is released back to the system. The metadata controller sends callbacks to those file system clients that have non-real-time I/O tokens informing them of the new amount of available non-real-time I/O, or, if the storage pool is transitioning from real-time to non-real-time I/O mode, the metadata controller sends callbacks to all file system clients.

If the disconnected file system client had a non-real-time I/O token, the token is released and the metadata controller recalculates the amount of available non-real-time I/O. The metadata controller sends callbacks to all file system clients that have non-real-time I/O tokens informing them of the new amount of available non-real-time I/O.

File System Client Token Releases

Although it is not a failure case, the handling of a file system client token release is exactly the same as the handling of file system client failures. If no I/O occurs within the fixed amount of time that non-real-time I/O tokens are valid, the client releases the token back to the metadata controller. The metadata controller re-calculates the amount of available non-real-time bandwidth and sends callbacks to other file system clients to let them know of the new amount.

To reduce system and SAN overhead in a situation in which an I/O operation occurs, for example, every 70 seconds, the `nrtiotokenhold` option to the `mount` command should be set to a value greater than or equal to 70 seconds. For details, see "Configuring the File System Client" (page 23).

Monitoring

The current real-time I/O statistics are available through the `cvadmin` utility. The utility's `show long` command provides information as to the current limit, the minimum amount reserved for non-real-time I/O, the number of active clients, the amount currently committed, and the amount of I/O a non-real-time application could expect to get when requesting I/O.Whenever the status of the storage pool changes (such as from non-real-time I/O to real-time I/O mode), an event is logged to `syslog`.

Document Revision History

This table describes the changes to *Xsan Programming Guide*.

Date	Notes
2006-05-23	Moved reference information to "Xsan Reference."
2005-04-29	Changed title from "Managing Networked Storage With Xsan." No technical changes.
2004-06-28	TBD

Document Revision History