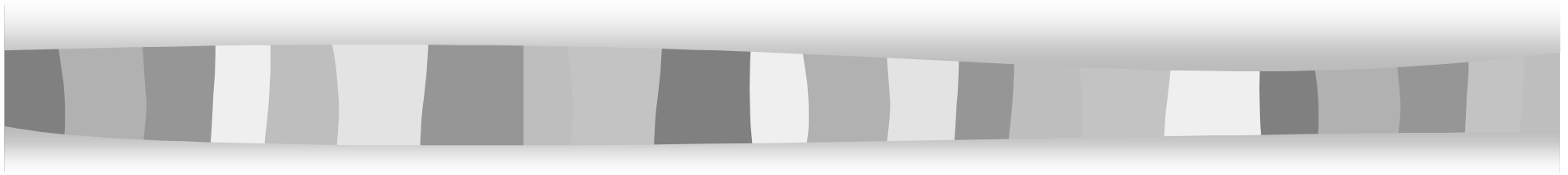
Shared (Disk) File Systems



Matthew T. O'Keefe
Associate Professor
Department of Electrical and Computer Engineering
University of Minnesota
Minneapolis, MN 55455
http://www.lcse.umn.edu/GFS

Outline of Talk

- The Problem of Shared Disk File Systems
- Distributed File Systems
- Shared File Systems: Definition and Key Properties
- Example Shared File System Designs
- Global File System
  - Architecture
  - Current Implementation
- Conclusions

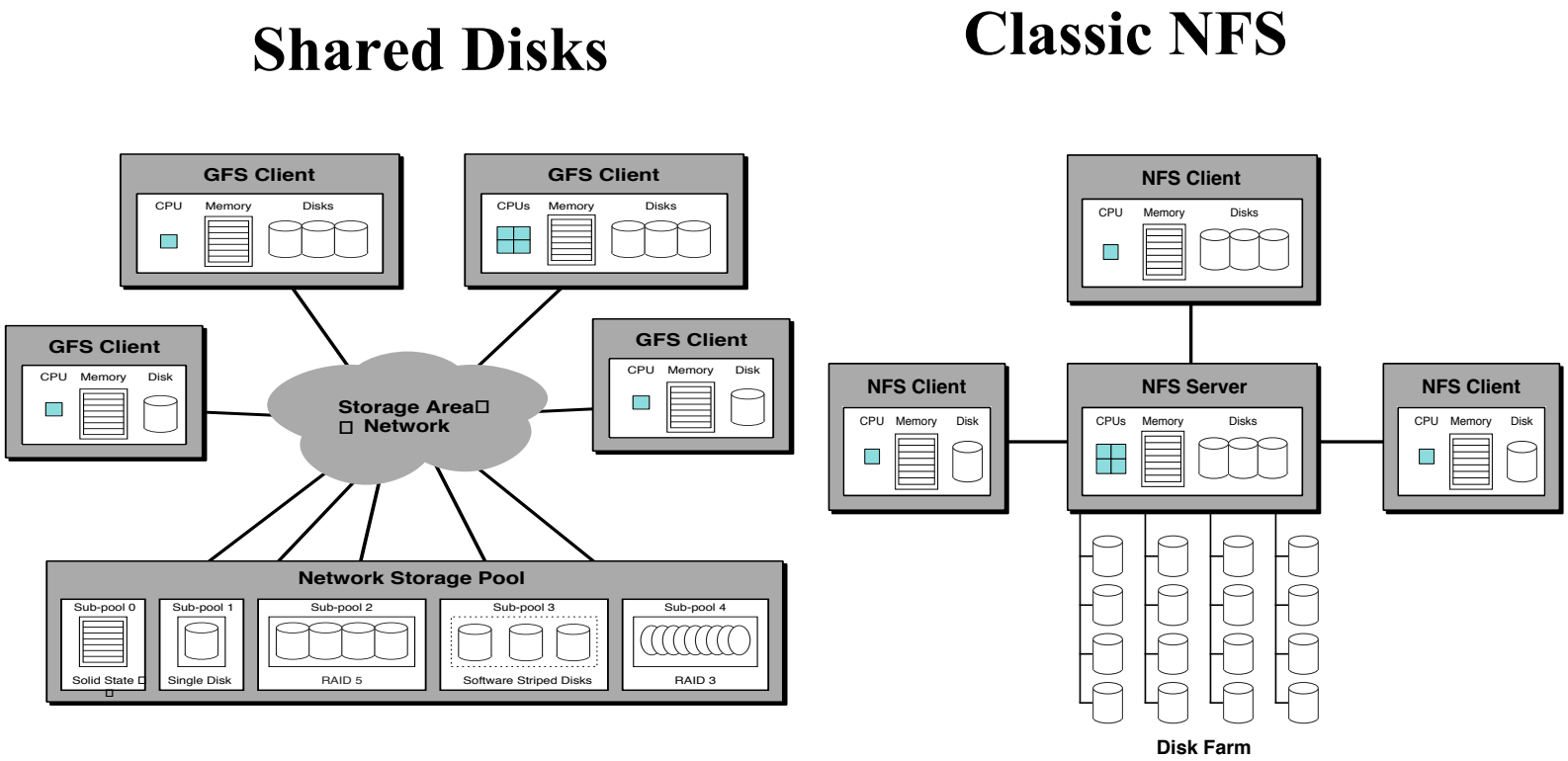
File Systems and Shared Disks: The Problem

- Shared disk network interconnects like Fibre Channel help solve important technical issues
  - Interconnect length, number of ports, speed
  - These are of course extremely important
- But there are significant opportunities to leverage these new capabilities to develop SCSI-based network file systems
- These file systems could be more scalable, cheaper, more reliable than current network-attached storage that uses NFS

File Systems and Shared Disks: The Problem

- The storage industry will evolve SCSI towards very smart disks that do file (instead of block operations)
  - That efficiently support caching and coherency on the drive
  - That build on the interface improvements to allow totally new storage architectures — SMP-like multiple-client architectures
  - The advanced development and research work in this area is happening now!
    - NSIC/NASD working group (Gibson)
    - Companies like Tricord out front technically

The New Opportunity: Exploiting Shared Disks

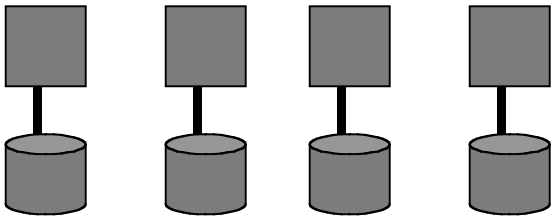


Explosive Data Growth

- Both documents and applications are becoming more media-rich, driving up file sizes
- Continued growth in capacity of memories and disks promotes further file growth
- Example environment: digital production houses
  - *Sneaker net* is preferred data transport media
  - *Academy* film format: about 3000x4000 pixels per frame
  - With 14-bits per RGB component: 42-bits per pixel
  - At 24 frames/second—2.0 GB for 1 second of film
  - 30 seconds with @66 Megabytes sec thru Fibre Channel

Potential New Applications

- Better Web server designs based on shared rather than replicated disks
  - Web server designs:



Potential New Applications

- More efficient parallel databases for data mining, other parallel-query-based applications
  - Significant advantages to shared disk in cluster environments
- Cluster-based applications which require high capacity and high bandwidth
  - Film and video
  - Feature film industry, advertising, local TV stations
- Ultimately, the right way to do distributed storage across the enterprise
  - Especially with smart disks

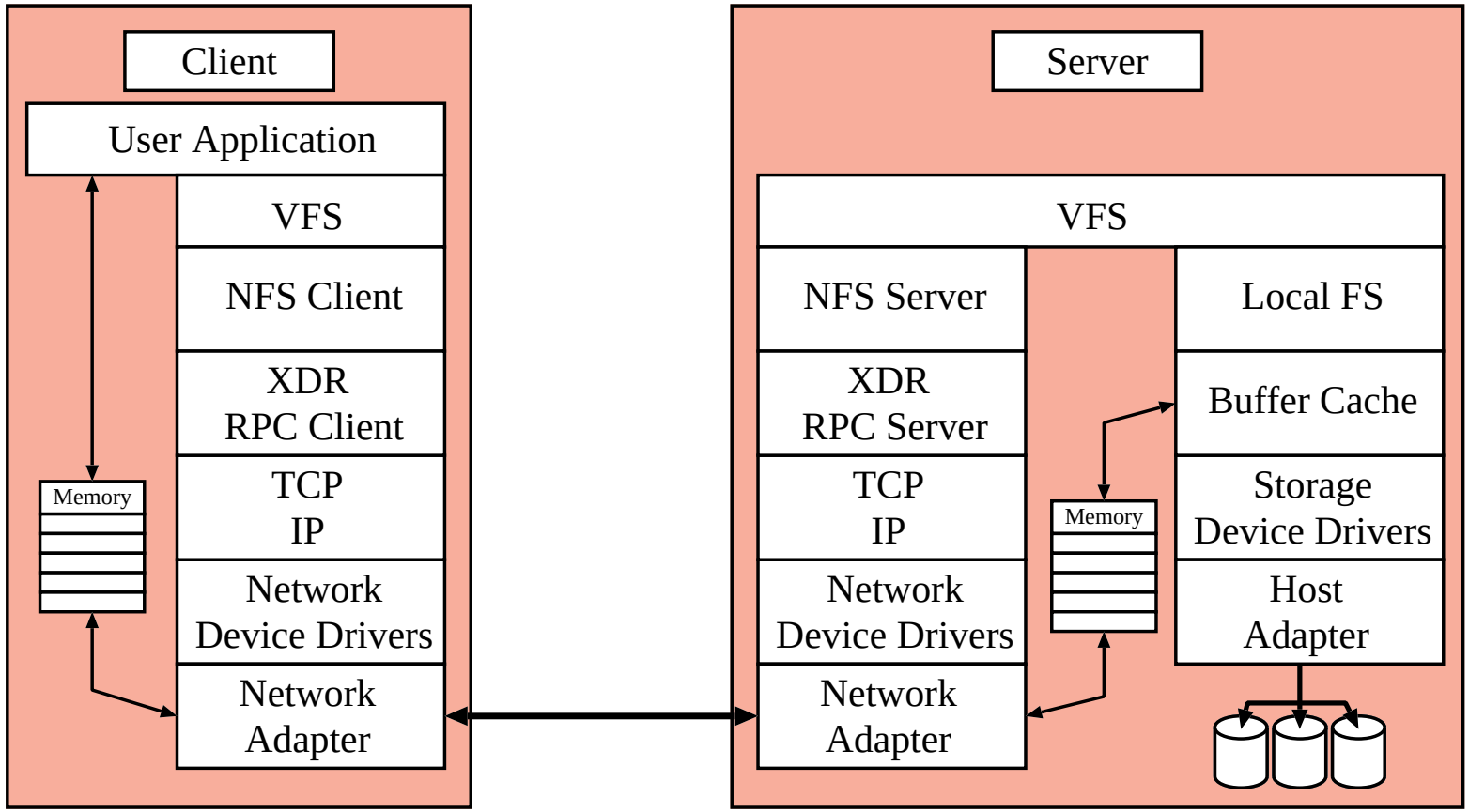
Classic Client-Server Distributed File Systems

- Sun's NFS (Network File System), Novell Netware, Microsoft's Lan Manager
- Traditional DFS'es use central server approach:
  - many clients share data through 1 central server
- Basic assumption: disks are not smart and cannot be attached to a network
- Generally complex and inefficient from standpoint of really large datasets
  - synchronous writes
  - client caching

DFS Evolution

- NFS got the ball rolling in the mid-1980's
- Popular but is well known to be inefficient
  - synchronous writes and write-through
  - statelessness means more retries
  - protocol stack overheads
- NFS is popular for several reasons
  - Its semantics are fuzzy so it can "work" in many different Oses
  - Brute-force hardware approaches can be used in some cases to increase performance

NFS Execution Path



Trends in Distributed File Systems

- AFS (DFS/DCE) and Coda use Distributed Servers distributed name spaces
  - redundancy
  - very aggressive caching, loose sharing semantics
  - better scaling, usable in Wide Area Network
  - complex, difficult to configure, closed system
- Still no concept of attaching devices directly to networks
  - Except for niches like mainframes and supercomputing

Merging Clients and Servers

- New approaches to DFS design allow machines to act as both clients and servers
- Merged client-server designs
  - Coda and Berkeley's xFS
- More functionality migrating to clients
- A natural effect given that computing systems today are driven by the desktop
- Clients are getting closer to servers in their capabilities

Future of Distributed File Systems

- NFS and other DFS'es rule file sharing today in LAN environments
  - these protocols have not driven the Web even though in some ways they are better protocols
  - Web technologies may displace current LAN DFS'es
  - DFS'es have generally not exploited LAN locality to improve performance: instead focus is portability

Future of Distributed File Systems

- Traditional distributed file systems solve a problem that no longer exists — machines could not talk to each other's storage devices
  - But they will continue to exist
  - Network-attached storage will allow NFS servers to be constructed from clusters of machines that share disks
- Fine-grained read/write sharing generally not supported
  - for example with NFS you can never be sure if the data you are reading is the latest copy if some other client has written to that file
  - AFS has session semantics

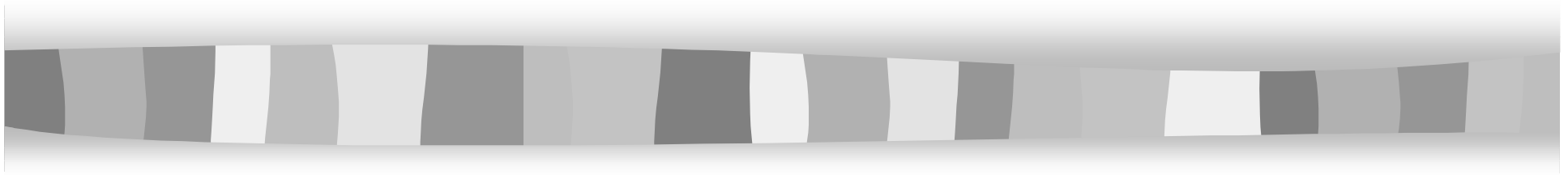
I/O Interfaces

- Channel interfaces (e.g., SCSI)
  - Connect computers to storage devices and other peripherals
  - High-performance, low connectivity, short connection distances
- Network interfaces (e.g., Ethernet)
  - Connects computers to other computers
  - Lower performance, high connectivity, long connection distances
- Merged interface — Fibre Channel
  - Both a channel and network interface
  - Supports storage attached to a network

Enabling Technologies

- Fibre Channel
  - High bandwidth, low latency network and channel interface
  - Highly scaleable, very flexible topologies
  - Becoming high-volume, hence lower-cost
  - Support from a wide-variety of adapter, computer, networking, and storage vendors
- Network-attached Storage (NAS)
  - Have your disks and share them too
  - Allows direct data transfer between disks and clients
- Together, Fibre Channel and NAS enable SCSI-based storage area networks (SANs).

Fibre Channel



Matthew T. O'Keefe
Department of Electrical and Computer Engineering
University of Minnesota
Minneapolis, MN 55455
http://www.lcse.umn.edu/GFS

Fibre Channel: Introduction

- Fibre Channel (FC) merges features from both networks and channel interfaces to create a storage interface that is
  - Fast
    - Fiber optic serial interface
  - Scalable
    - In the number of nodes, devices supported per node, and network bandwidth available per node
  - Efficient
    - Lots of low-level processing performed in silicon
  - Open, high-volume, industry standard
    - Many strong vendors and OEMs participating in Fibre Channel development and production
    - Industry groups: FCLC and FCA

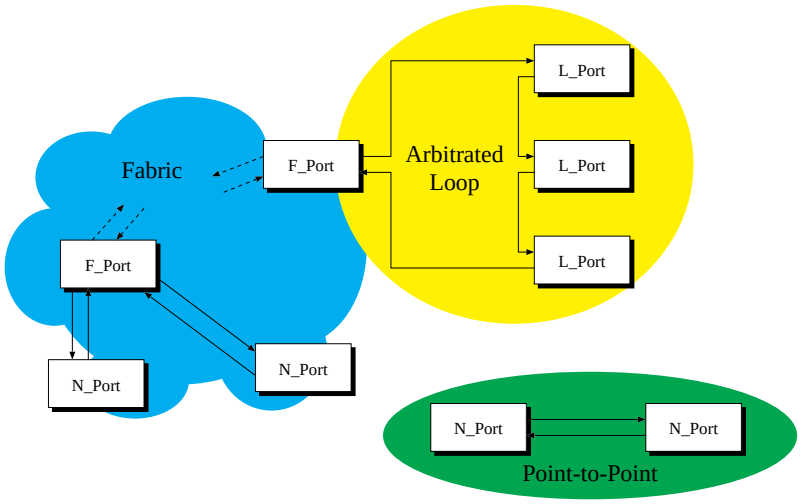
FC Layers

- FC functionality implemented across multiple layers
  - Physical media and transmission rates
  - Encoding scheme
  - Framing protocol and flow control
  - Common services
  - upper-level protocol interfaces

Physical and Signaling Layer

- **FC-0**
  - Covers physical characteristics of the interface and media including cables, connectors, drivers, transmitters and receivers
  - Single- and multi-mode fiber optics
    - 1 GHz shipping now, up to 4 GHz defined in standard
    - 1000s of meters
  - Copper coax for shorter distances (LANs)
- **FC-1**
  - dc-balanced 8B/10B code scheme (thank you IBM) used for clock recovery, byte synchronization, and encode/decode
  - Comma character insures proper byte and word alignment
  - Good error detection capabilities and simple logic implementation for the encoder and decoder

FC Topologies



- Dedicated Point-to-Point
  - Direct connection between single host and disk
  - N_port
- Shared Bandwidth Arbitrated Loop
  - Hubs used to connect nodes in shared loop topology
  - NL_ports
- Switched Fabric for Scalable Bandwidth
  - Multiple switches interconnect into a Fabric
  - N_ports to F_ports, NL_ports to FL_ports

Framing and Signaling Layer (FC-2)

- Defines transport mechanism independent of upper layer protocols
- Self-configuring and supports point-to-point, arbitrated loop and switched environments
- N_port is node (server, workstation or peripheral)
  - If port connected to a loop, it becomes an NL_port
  - Data communications occur between interconnected ports
  - Each node has an ASIC with an embedded FC Link Control Facility
  - Each port can act as an originator, responder, or both and has a unique identifier: *N_port* or *NL_port Identifier*

FC-2: Framing Layer

- Defines the framing structure and provides the following functions:
  - robust 32-bit cyclic redundancy check
  - Various classes of service to provide and manage
    - Circuit switching
    - Frame switching
    - Datagram services
    - Fractional bandwidth virtual circuits
  - A flow-control scheme that guarantees delivery
    - Buffer-to-buffer
    - Node-to-node
  - Built in protocol to aid in managing the link, control the FC config, perform error recovery, and recover link and port status information

FC Design Philosophy

- Designed to be hardware-intensive: use low-cost ICs to minimize software overheads
- Contents of frame determine destination
- At gigabit speeds decisions must be made in hardware
- FC-2 ordered sets, frames, sequences, exchanges

NASD Industry Standardization efforts

- Disks should be smart devices (they are already pretty clever, as are HBAs)
- File level operations performed at the disks
  - Rather than blocks have objects
  - The objects are directories, files, and objects related to recovery
- Joint industry effort led by Gibson et al.
  - Seagate, StorageTek, IBM, and others
- It's clear SCSI will be the transport protocol
  - This is where SCSI-4 is headed

Shared File Systems

- An architecture for distributed file systems based upon shared storage
- Fully exploits the special characteristics of Fibre Channel-based LANs
- Key feature is that clients transfer data directly from the device across the SAN (Storage Area Network)

Shared File Systems

- Key characteristics:
  - Hence more than one client may access the data from the same storage device
  - The device is shared between clients via some kind of interconnection network
  - Shared file systems must recognize the existence of other clients accessing the same storage devices and file system data and metadata
    - Directly through the metadata
    - Through a file manager
    - Precludes most local file systems: these consider storage devices as owned and accessed by a single host computer

Shared File Systems (Advantages)

- Advantages include
  (1) *Availability* is increased since when a single client fails another client can still access the associated data and continue the failed clients work
  (2) *Load-balancing* a mixed workload among multiple clients sharing disks is simplified by clients ability to quickly access any portion of the dataset on any of the disks
  (3) *Pooling* of storage devices into a shared disk memory equally accessible to all clients in the system is possible
  (4) *Scalability* in capacity, connectivity, and bandwidth can be achieved without limitations inherent in file systems designed with central servers

Example Shared File Systems

- DEC Vaxcluster (mid 1980s)
- IBM Sysplex (mainframes)
- Oracle Parallel Database Server (~1990)
- LLNL's HPSS
- Cray's Shared File System (1994)
- Veritas Cluster File System (1998)
- IBM's Parallel Journaled File System (1995)
- Gibson's NASD project at CMU (1995)
- U. Minnesota's Global File System (1996)

Classifying Shared File Systems (SFS)

- Symmetric or Asymmetric?
- A shared file system is *symmetric* if
  - any client can perform any file system operation without necessarily interacting with another client
- In *asymmetric* shared file systems a client must first make a request through a *file manager* executing on another client
  - File manager typically manages
    - File system metadata
    - Checks file access permissions
    - Provides client with info necessary access data directly on disk via the SAN

Asymmetric Shared File Systems

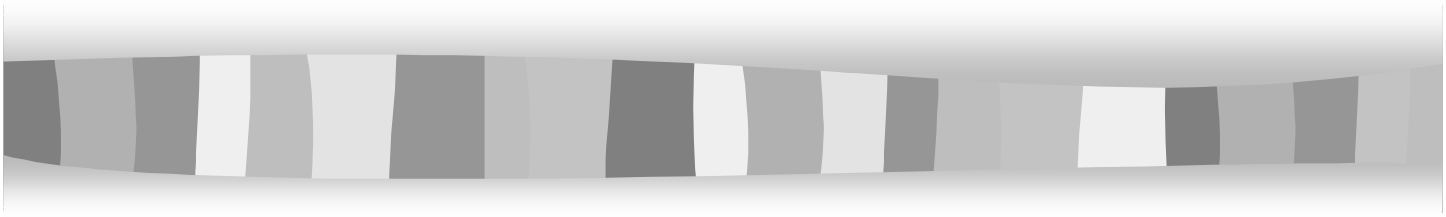
- Asymmetric shared file systems replace the server with the file manager
  - clients must access metadata through the file manager which effectively controls client access to files
  - sometime called 3rd-party transfer: the file manager is a 3rd-party which sets up xfer between client and device
  - once transfer approved, direct transfer is possible

Asymmetric SFS

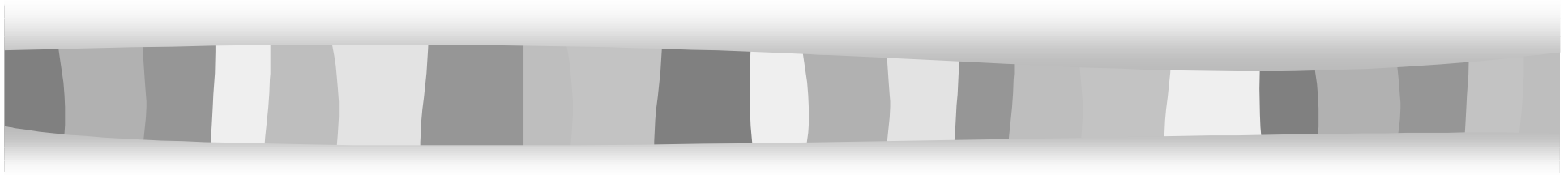
- Asymmetric shared file system advantages:
  - file manager design will generally re-use file server code
  - centralized design is simpler
- Asymmetric shared file system disadvantages:
  - same problem as client-server: server is a bottleneck to scalability (but does ease bandwidth and capacity issues)
  - must write both client and file manager code
  - centralized logging and locking introduce bottlenecks
  - file manager machine is a single point of failure

SFS Locking

- Where is locking performed?
  - in the clients or file manager:
    - centralized file manager (CMU NASD, CFS, HPSS)
    - distributed lock manager (Vaxcluster, Oracle PDS)
      - DLM's are notoriously difficult to design in the context of client failures
  - in the devices or network:
    - devices (GFS, Cray SFS)
      - simpler protocol to design
      - requires pool of fine-grain locks in the device or network
      - these locks must be fast



IBM Sysplex



Matthew T. O'Keefe
Department of Electrical and Computer Engineering
University of Minnesota
Minneapolis, MN
http://www.lcse.umn.edu/GFS

IBM Sysplex

- IBM has been clustering mainframes for a long time
- JES eases task of distributing work among a cluster of mainframes
- In 1994, additional hardware and software provided to allow more fine-grain data sharing and parallelism
- Cluster of up to 32 MVS/ESA systems running System/390 mainframes (figure)
  - Multiple machines share data through ESCON directors
  - Provides simultaneous access to disk

IBM Sysplex

- ESCON director works like Digital Star coupler and Fibre Channel switch or hub in GFS
  - Allows multiple simultaneous transfers
- ESCON is 12-Mbyte/sec connection to S/390s
- *Sysplex timer* provides common real-time reference for all systems
  - Consistent data and time stamping of transactions
  - File modifications
  - Other parallel operations

Sysplex Coupling Facility

- Sysplex coupling facility is a hardware assist for performance enhancement: it provides
  - Cluster-wide locks
  - Stable electronic storage for caching Sysplex-wide info such as
    - Global work queues
    - Common read/write data
- Sysplex Timer and Coupling Facility are the “extra” hardware introduced to improve Sysplex performance in 1994

Exploiting Cluster Parallelism in Sysplex

- Parallel applications executing across the cluster will exploit Coupling Facility and Timer
- Pfister: "list of applications is Acronym City"
  - IMS DB (Information Management System Database Manager) is a hierarchical database product
  - DB2 (Database 2) — relational database product
  - VSAM (Virtual Storage Access Method)
  - RACF (Remote Access Control Facility) — security
  - JES2 for job entry

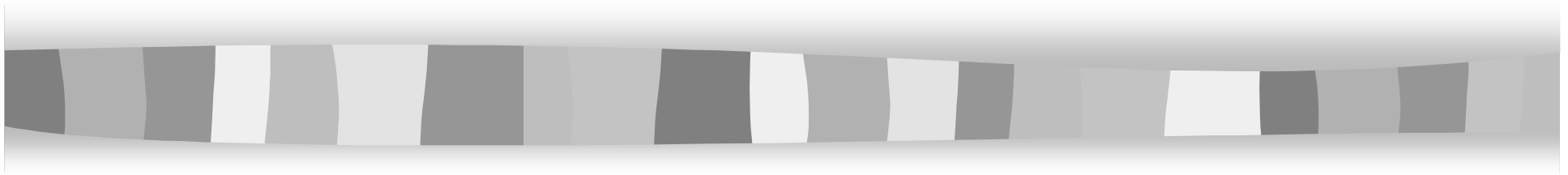
Sys Admin Tools in Sysplex

- System administration utilities allow creation of a single point of control
  - Single console from which target area can be managed and administered for entire cluster
  - ESCON Manager monitors and manages I/O configuration and status
  - others

Sysplex Summary

- *Symmetric or asymmetric?* Not clear to me from my sources
- *Locking?* Performed on device — Sysplex coupling facility or ESCON Director
  - SCSI's RESERVE/RELEASE is a descendant of this IBM facility for locking devices when multiple machines may access them
- *Proprietary or open storage interconnect?* ESCON is basically proprietary
- Existing file systems modified to work with cluster

DEC Vaxclusters



Matthew T. O'Keefe
Department of Electrical and Computer Engineering
University of Minnesota
Minneapolis, MN
http://www.lcse.umn.edu/GFS

Vaxcluster System

- Developed by DEC to provide a highly available system that provides users with a single system image across a cluster of Vax workstations
- Original implementations used the CI (Cluster Interconnect) — custom 70 Megabits (Mbps) network that interconnects both computers and disk controllers
  - Star Coupler
- A symmetric shared file system for Vaxcluster nodes
  - Locking handled through the DLM: Distributed Lock Manager

Distributed Lock Manager (DLM)

- DLM: a client-based lock manager that provides a generalized lock service for all resources in the Vax cluster
  - Devices
  - Print services
  - Files
  - Other abstractions defined by the OS

DLM

- Lock manager allows clients to request and release a lock
  - Each request specifies a locking mode: varying levels of exclusive control on associated resource
    - Exclusive: no other host may read or write to the resource
    - Concurrent read access: other clients may read or write to the shared resource

DLM Design Issues

- DLM handles cluster-wide synchronization
- Several goals influenced the design:
  - Programs using DLM should run in both single node and cluster configurations
  - Lock services must be efficient to support system-level software that makes frequent requests
    - Lock manager must minimize the number of messages needed to manage the locks
    - In single-node environment the lock manager should recognize the simpler environment and bypass cluster-specific overheads
  - Lock manager must recover from failures of nodes holding locks so that surviving nodes can continue to access shared resources in a consistent manner

Distributed Lock Manager (DLM)

- Lock requests may be queued
  - Once resource becomes available, the requesting client is so informed via a callback
  - Implies significant state associated with each lock regarding the status of outstanding requests for the lock
- DLM is distributed across the clients in Vaxcluster
  - Provides load balancing across the parallel resource: the locks
    - This aids scalability
  - Locks are cached on requesting client if possible
  - Must work in the presence of client failures

VMS File System and Vaxclusters

- Original VMS file system was modified to use the DLM to support shared, simultaneous access to
  - Files
  - Associated metadata
    - Directories, files, volumes
- Extensive caching used in original local VMS file system
  - This approach preserved in Vaxcluster implementation
  - Version numbers were associated with lock operations
  - Stale cache data detected as a disparity between version numbers caused by earlier file update operations performed by other clients

Version Numbers and Caching

- Example: clients A and B both accessing file `foo`
  - Assume both A and B have the file cached and both are allowed to read and write the file
  - A makes first access (read) to `foo`: lock version number (vn) is 1
    - Version number “1” is associated with cached data on A
  - B makes 2nd access (read) to foo : lock vn still 1
    - vn “1” associated with cached data on B
  - A makes 3rd access (write) to foo: lock vn incremented to 2
    - vn “2” now associated with cached data on A
  - B makes 4th access (read) to foo: since lock vn is “2” but cached vn is “1”, B knows it must not use its cached copy
    - Reads data directly from disk instead

Caching Data with Deferred Writeback

- *Software Interrupt Option:* when requesting an exclusive lock, process can specify it be notified by software interrupt
  - if another lock request on the resource is forced to block
  - Then a process owning the lock can cache and repeatedly modify the data
  - Data is written back to disk and lock released when the lock is notified that it is blocking another process

Lock Manager Features

- Lock manager provides a *lock conversion* service to change the locking mode on an existing lock
- Conversions are faster than new lock requests because table entry representing the lock already exists
- Applications will frequently hold a *null* lock on a resource and then convert it to a more restricted access mode later

Lock Manager Implementation

- The lock manager implementation intended to
  - distribute overhead of lock management throughout the cluster
  - while still minimizing the inter-node traffic needed to perform lock services
- The lock table is therefore divided into two parts (both of which are distributed):
  - Resource lock descriptions
  - Resource lock directory system
- Each resource has a *master node* responsible for granting locks on the resource

Master Node for a Lock

- Master maintains
  - a list of granted locks
  - queue of waiting requests for that resource
- Note that the master for all operations for a single tree is the node on which the lock request for the root was made
- The master maintains the lock data for its resource tree
  - But any node holding a lock on a resource mastered by another node keeps its own copy of the resource and lock descriptions
- The resource directory system maps a resource name into the name of the master node for that resource
  - The directory database is distributed among nodes willing to share the overhead associated with processing directory requests

Locking a Resource

- Given a resource name, a node can trivially compute the responsible directory as a function of
  - the name string and
  - the number of directory nodes
- To lock a resource, the lock manager sends a lock request message to the directory for that resource:
- The directory responds in one of 3 ways:
  1) if directory is on master node for that resource, lock request performed and confirmation returned
  2) if directory is not on master node but it finds the resource defined, it returns the identity of the master node
  3) if directory finds resource undefined, it returns a message telling requesting node to master the resource itself

Locking a Resource

- In best cases (1 & 3) above, two messages required to request a lock: case 2 takes four messages
- Unlock executed with one message
- If lock request is for sub-resource in a resource tree then requesting process will either
  - Be located on the master node (local request) or
  - Will know who the master for its parent is — allows bypass of directory lookup
- In any case number of messages independent of number of machines in the cluster

Vaxcluster File System Summary

- *Symmetric or asymmetric?* Symmetric
- *Locking?* Performed on clients using DLM
- *Proprietary or open storage interconnect?* CI and Star Coupler are proprietary
- Existing file systems modified to work with cluster

Cray Shared File System (SFS)

- Cray had a customer who required that multiple (at least 4) C90s be able to access the same data directly and be constantly available
  - this evolved later into a more general product (though to my knowledge few are installed)
- This required a shared disk solution
- Cray modified their own Unicos (Cray's version of UNIX) file system

SFS Goals

- Common storage device to share file system data and metadata
- High bandwidth
- All machines in the “complex” are peers — no server
- Re-use existing UNICOS file system code
- Works with all the standard UNICOS utilities
- High availability

“Special” Version of SFS

- 4 C90s shared a common SSD (Solid State Disk) which held the file system and metadata
- Separate arbitration device to control access to shared data and metadata:
  - semaphore device known as the SMP: serializes operations using atomic test-and-set
  - SMP can turn around request from C90 in 6 microseconds
  - special-purpose hardware: redundant — 2048 locks
  - confusing point: what are usually called locks the paper calls “semaphores”

Phase 2 SFS

- Generalized SFS to work with more standard, less expensive peripherals
- Generally available software in UNICOS 9.0
- In phase 2 the shared media is a HiPPI disk array
- Arbitration service provided by a SPARC workstation attached to the HiPPI switch (HSMP)
  - dedicated to SMP operation — 1 millisecond turnaround
  - locks implemented as IPI-3 disk commands

SFS-2 Operation

- UNICOS employs typical UNIX file system design principles
- Dinodes on disk reside in special “dinode regions”
  - dinode is 256 bytes long
  - with 4096-byte disk block, 16 inodes per inode region block
  - on RAID-5 device with 64K block size there are 256 inodes per inode block

SFS-2 Operation

- None of the standard UNIX buffering was used in SFS
- Dinode locking must be done externally
  - different from standard UNIX where in-memory lock sufficient to serialize access among many processes
- In SFS, locking accomplished both with the external semaphore "device" (HSMP) and by scribbling locking information into the dinode

SFS-2 Read

- Many contiguous SFS dinodes map to single semaphore (lock)
- Semaphore used like a test-and-set bit when accessing an "inode sector": a contiguous set of dinodes
  - semaphore acts like test-and-set bit: if busy try again later
  - if semaphore is acquired, enter name of C90 that grabbed it in the semaphore
- Once semaphore acquired, then the particular dinode to be acquired must also be locked by writing into dinode fields on disk

SFS-2 Read

- If dinode in sector acquired, we set lock and write that dinode sector back out to disk and release semaphore
- Read of disk data then performed
- Following read, repeat previous sequence to unlock the dinode
  - acquire semphore
  - acquire dinode lock
  - unlock and write back dinode into “inode sector”

SFS-2 Read

- Lots of traffic: at least 4 HiPPI packets sent per read(!) assuming no lock conflicts
- Dinode sector must be read and re-written twice
- No buffering as in non-shared case
- Significant performance degradation, attack by
  - file locking
  - large user buffers
  - reduced directory locking
  - faster metadata devices

SFS-3 Operation

- SFS-3 implements series of improvements
- An optimization proposed was to lock files in “read-data” or “write-data” mode (i.e. like traditional UNIX file locks)
  - known as “data locks” in SFS jargon
  - use fields in the dinode itself (like the “dinode sector” lock)
  - lock state and identity of the C90 owning the lock

SFS-3 Operation

- Multiple read locks are allowed
  - in-memory inode serializes reads from single C90 with multiple processes accessing same file
- Only one process on one machine may own a write-lock, and then only if there are no read locks in place

SFS-3 Operation

- File locking uses *fcntl* command (F_SETLK and F_GETLK)
  - but in SFS these are not advisory locks
  - always work across the whole file
- Lock state kept on disk and on-processor
- UNICOS can be set to always do Direct I/O — as in original SFS, when request not aligned UNICOS can break up the request into aligned and un-aligned requests

SFS-3 Operation

- Pre-allocation for writes proposed to reduce metadata access and related semaphore ops
- Recovery in SFS depends on central shared table in HSMP device
  - one semaphore for recovery: only one machine can do recovery at any one time
  - HSMP maintains heartbeat status of machines in cluster
  - during recovery semaphores held by failed clients must be released and disk metadata “fixed”

More SFS Performance Improvements

- No locks on directories while being searched
- Realized that locking the dinode number in the semaphore device would be useful
  - “inode sector” still locked for writes
  - dinode and device numbers combined into 32-bit number locked in semaphore — symbolic semaphores
  - keep track of machine holding the semaphore — like a version number but more limited in usefulness

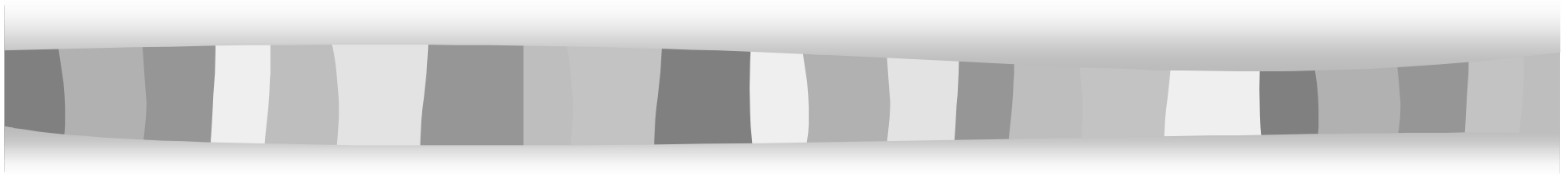
SFS Differences with GFS

- GFS decouples sharing from locking
- also decouples sharing from clustering
- GFS has no restrictions on read- and write- sharing
- Dinodes not concentrated in single sector on disk
  - why is this good/bad?

Cray SFS Summary

- *Symmetric or asymmetric?* symmetric
- *Locking?* Performed in device or network switch
- *Proprietary or open storage interconnect?* HiPPI, though not proprietary, is low volume
- Existing file systems modified to work with for shared file system problem

NASD: Network Attached Secure Disks



Matthew T. O'Keefe
Department of Electrical and Computer Engineering
University of Minnesota
Minneapolis, MN
`http://www.lcse.umn.edu/GFS`

NASD Overview

- Gibson has proposed *Network Attached Shared Disks (NASD)* as a standard for shared storage devices
- NASD goes beyond previous shared disk storage systems in two key areas
  - Security
  - Objects
- NASD-based file systems as currently proposed use file manager for directory and certain cryptographic operations
  - Mechanisms provided to keep these overheads low

NASD Overview

- NASD could potentially allow symmetric shared file system designs as well
- Secure communications between disks and clients achieved using capabilities that have been cryptographically sealed
  - Support for cryptographic operations placed on the devices

NASD Overview

- NASD systems dramatically raise the semantic level of disk drive operations
  - From fixed-size blocks to variable-sized objects
  - Objects can be files or directories; support for partitions (containers for separate groups of files) is provided
- Higher semantic level for objects on devices means that fewer disk commands need be sent over the network per file operation
  - Reduces network overheads and improves scalability

Properties of NASD (Gibson)

- *Direct Transfer*: data accessed by a filesystem client transferred between NASD drive and the client without store-and-forward through file server
- *Asynchronous Filesystem Oversight:* file manager retains some file server functionality as it relates to filesystem policies
  - Goal is to reduce overheads of additional checks required in shared storage environment (i.e., can this client access data on this drive?)
  - Offloads this work to the drive to keep load small on file manager as more clients and devices added to system

Properties of NASD (Gibson)

- *Cryptographic support for request integrity:* NASD drives must be capable of computing keyed digests on command and status fields with little or no performance penalty
  - Required since no synchronous filesystem oversight and no assumption network between clients and disk is secure
  - Cannot depend on trusted clients
  - With hardware support for cryptography in a drive, high-performance data integrity, data privacy and command privacy are feasible

Properties of NASD (Gibson)

- *Storage Self-Management:* NASD drives know a lot more about the nature of the data that they hold due to the higher level of abstraction (compared to SCSI's linear array of randomly-accessible blocks)
  - Hence blocks associated with a single file in an object can be kept contiguous
  - Access patterns to file objects as a whole discerned more easily due to concept of an object
  - Extends SCSI already extensive error recovery and self-checking and reporting of drive and media state to file objects
  - Reduce overall cost of storage management

NASD Objects and Operations

- Current SCSI disks provide virtual linear-block interface
  - Virtual because modern disks transparently re-map storage sectors to hide defective media and variations in track densities
  - Locality based optimizations supported since blocks small distance apart in the linear address space are physically close
  - More advanced SCSI devices also implement:
    - RAID
    - Data compression
    - Dynamic block remapping

NASD Objects and Operations

- NASD abandons idea that file managers understand and directly control layout
- NASD drives store variable-length logical byte-streams called *objects*

NASD Filesystems

- Filesystems request objects from the drive to hold the file's data
  - Read and write operations apply to a byte region within an object
  - Layout of the object on the drive determined by the NASD drive itself
  - Sequential addressed in file NASD objects allocated on media for fast sequential access
- Objects can be clustered: several obejects placed together on proximity list (multiple linear block lists)

NASD Objects and Operations

- Proximity on object list encourages proximity on physical media
- NASD supports secure communications using capabilities that have been cryptographically sealed by the right authority (filesytem manager or partition manager)
  - shared secret between NASD drive and the authority
  - Four-level key hierarchy includes
    - master key (system level, changed very infrequently, not stored online) and
    - drive key — long-term on-line key used to manipulate NASD partitions; changed when compromise feared

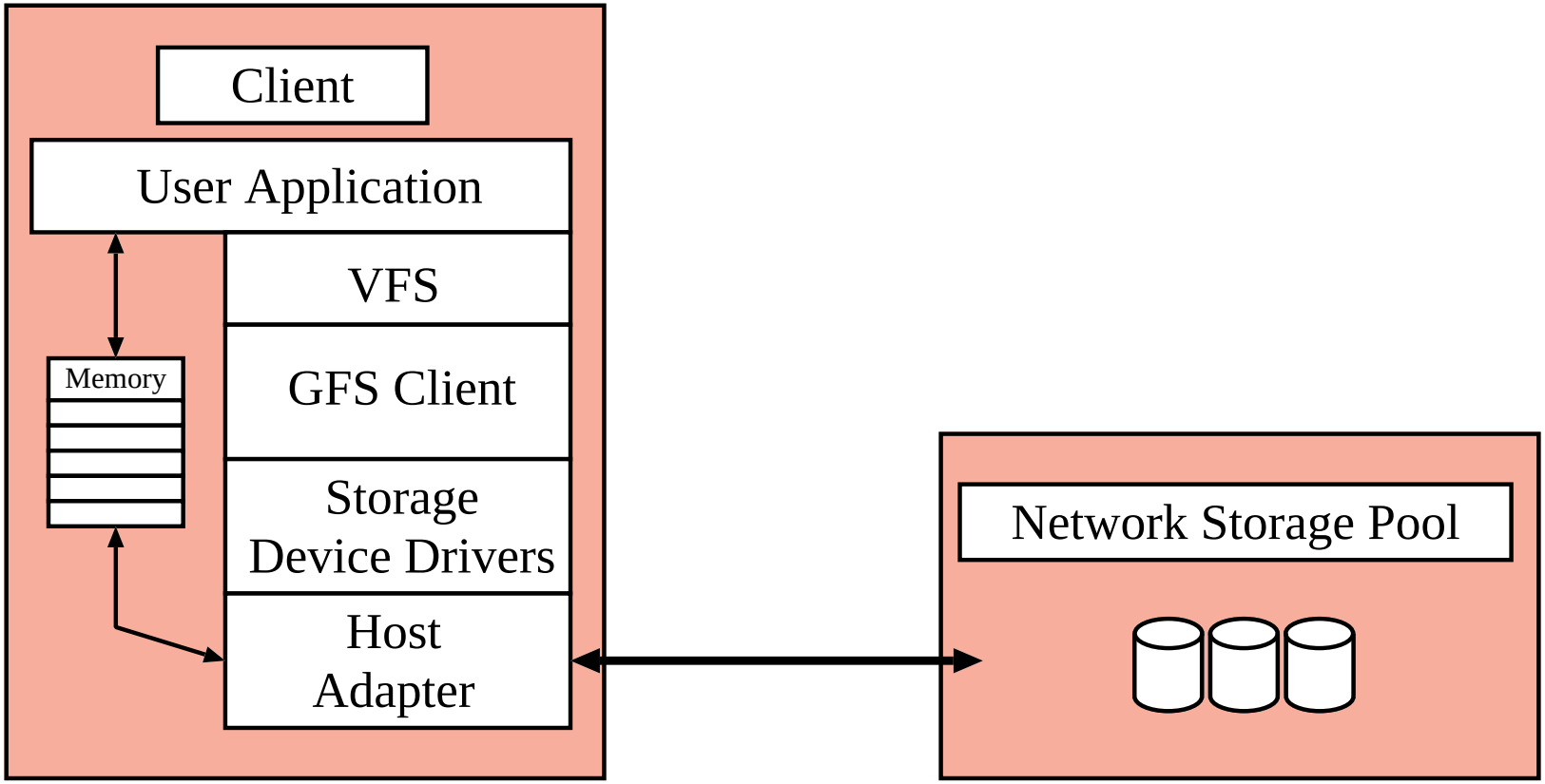
NASD Summary

- *Symmetric or asymmetric?* Asymmetric as currently proposed
- *Locking?* Performed on clients or file manager *Proprietary or open storage interconnect?* SCSI, so its open
- Existing file systems like AFS and NFS modified to work with NASD
  - Good scaling so far from 4 to 8 clients

The Global File System (GFS)

- A shared file system developed at the University of Minnesota
- A collaboration with *Seagate, Prisa, Brocade* and other FC vendors — an open system
- Hardware solution to the problem of distributed file system design
- Oriented towards applications that have lots of big files and that require high bandwidth
- New design will provide more general architecture

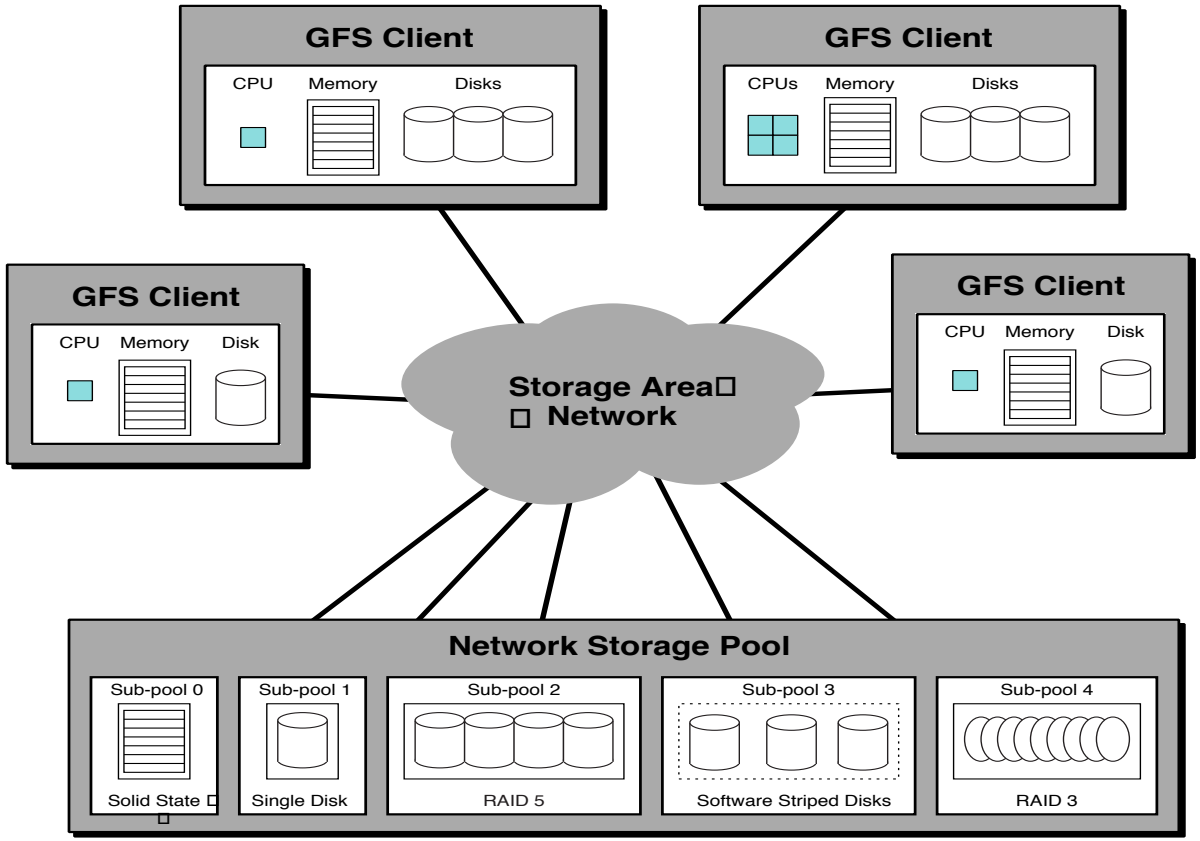
GFS Execution Path



Global File System (GFS)

- Network Storage Pool — a shared address space of disk blocks
  - like a shared memory in an SMP (symmetric multiprocessor)
  - implemented in the “pool driver” layer beneath the file system
  - pool layers implements:
    - locks
    - striping
    - `pool_assemble` to assemble network devices into shared pool of devices
    - Fibre-Channel-specific stuff like name service etc.
  - pool partitioned into subpools

A Distributed GFS Environment



Global File System (GFS)

- A symmetric shared file system
  - nodes are independent and act as peers relative to the storage devices: no file manager
  - like an SMP
    - clients are like processors
    - network storage pool is the shared memory
    - any client can execute any portion of the kernel or any application
  - SMP architecture dominates today's multiprocessor designs
    - Most efficient for load-balancing and throughput efficiency

GFS Consistency

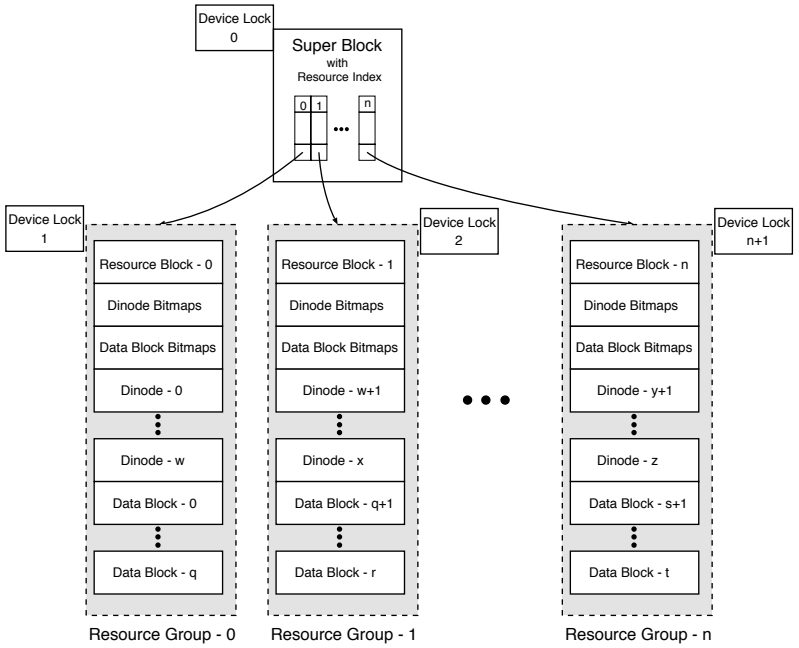
- similar to test-and-set locks in memory
  - *Test-and-set, Clear*
  - many locks per device: multiple lock accesses across parallel devices can be made
- locking performed on the devices
  - nodes compete for pool of locks kept on the storage devices
  - DLOCK SCSI command developed to implement fine-grain locking
  - SCSI RESERVE/RELEASE granularity is too high (whole device)
  - SCSI DLOCKS provides a pool of 1000s of locks which the file system may then map to shared metadata as required

GFS Organization and Structure

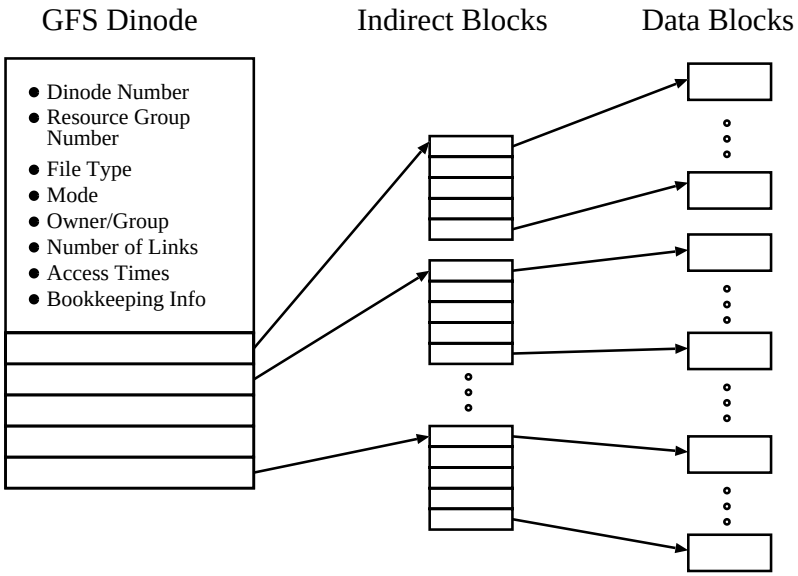
- Super block: contains Mount information and static Resource Group Index
- Multiple Resource Groups:
  - metadata partitioned into groups for scalability and load balancing
  - *Resource Group* Block
    - bit maps for free data blocks
    - data blocks
      - dinodes (disk inodes) — one per file, dynamically allocated
      - metadata — pointer blocks and indirect pointer blocks
      - real data blocks

GFS Organization and Architecture

- Super block
  - Maintains mount information and static file system attributes
- Resource Groups
  - Partitions and distributes file system resources for parallel accesses
  - Allocated per subpool in the network storage pool
  - Contains bitmaps used for block allocation
  - Similar to Allocation Groups in Silicon Graphics' *XFS*

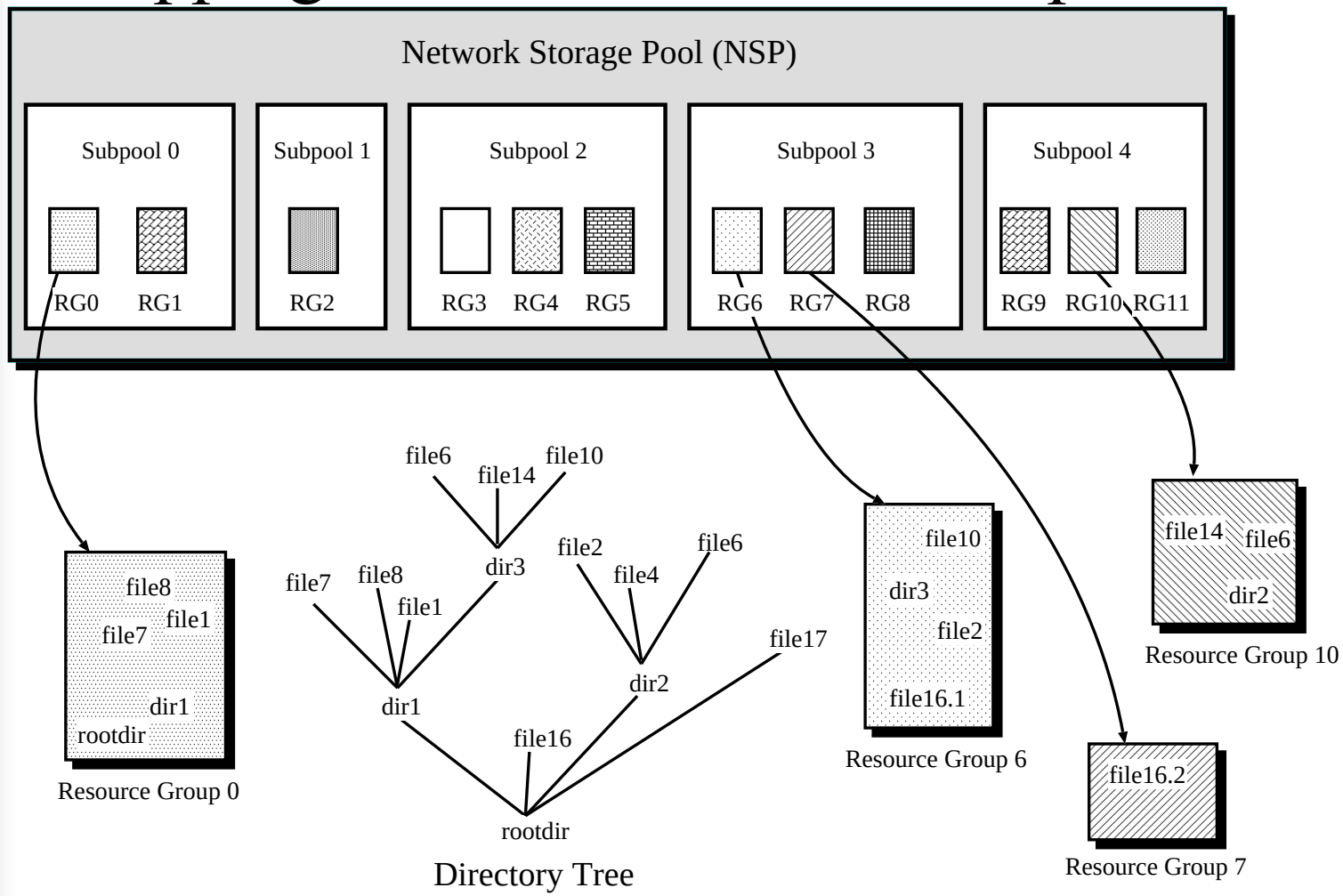


GFS Organization and Architecture



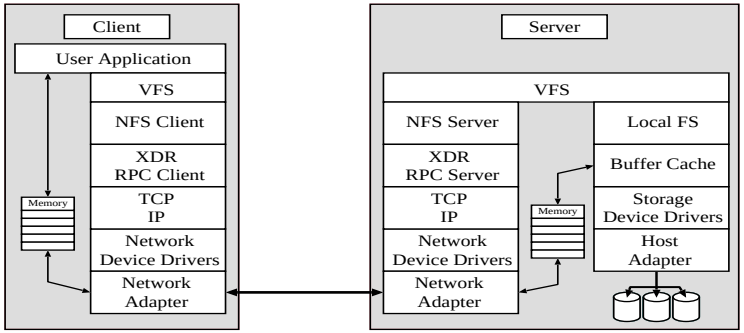
- Dynamic Block Allocation
  - Available file system blocks may be freely allocated to directory or file dinodes, pointer blocks, or data blocks
  - Inode and dinode numbers based on storage pool address eliminating lookup indirection

Mapping Files to Resource Groups and

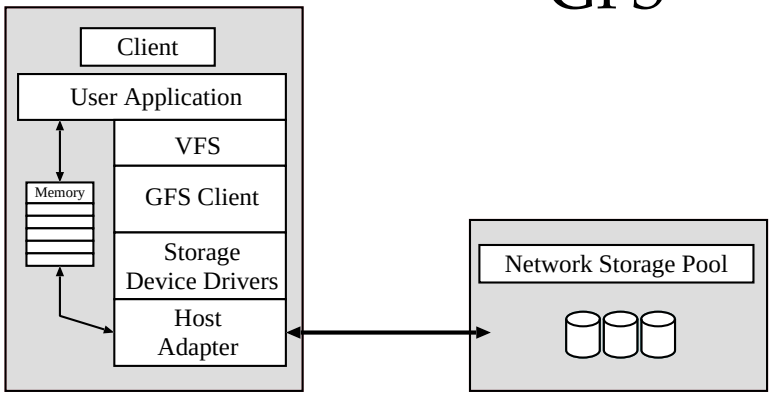


Comparison of GFS and NFS Control and Data Paths

NFS

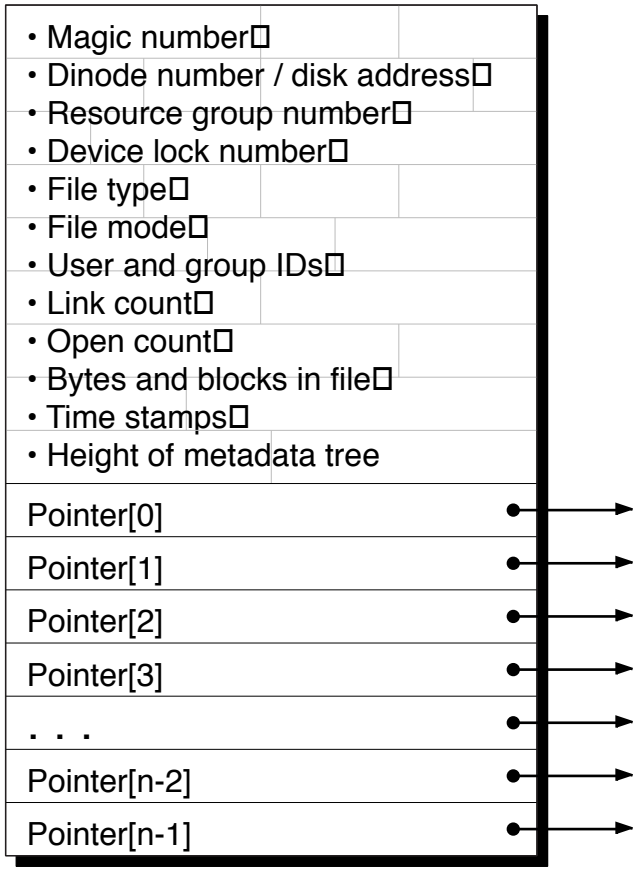


GFS



Dinode Stuffing

- Directory and file dinodes occupy an entire file system block
  - As block size increases header information stays constant
  - Block utilization decreases decreases leading to internal fragmentation
- Place user data in the unused dinode space
  - Reduce internal fragmentation
  - Eliminate pointer indirection
  - Eliminate an additional read operation



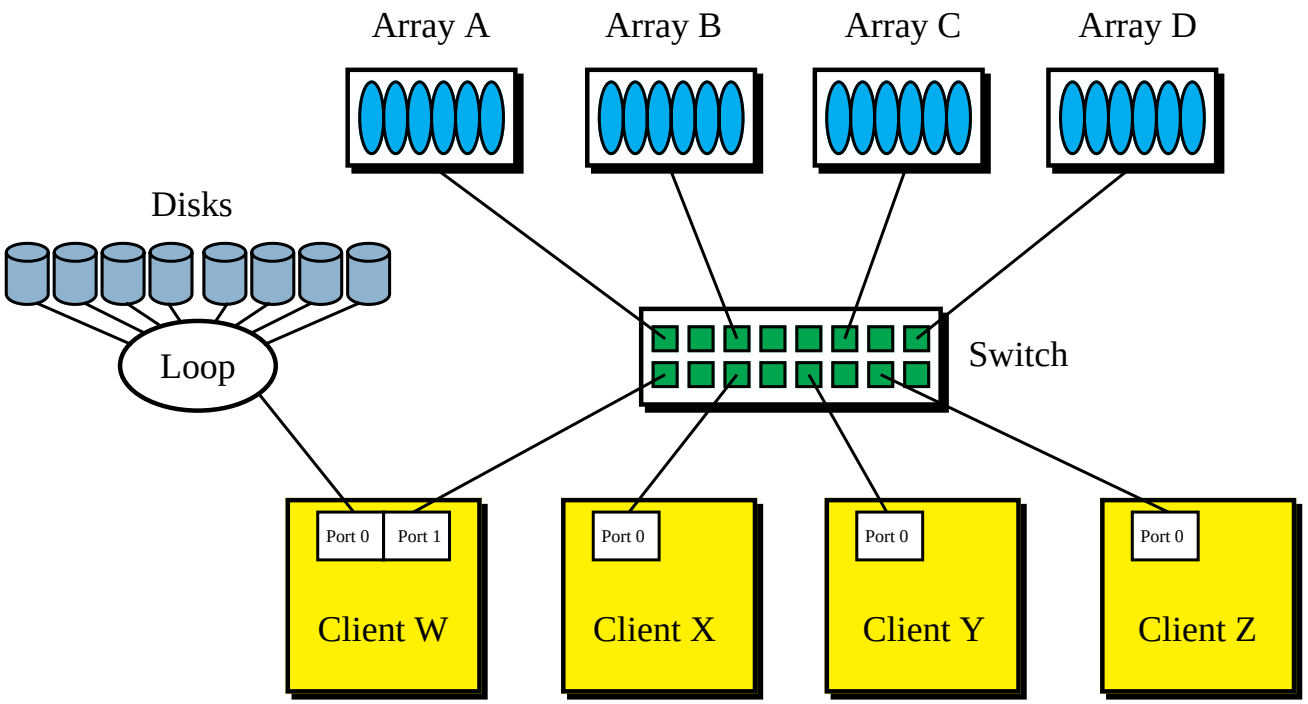
GFS Caching

- Some metadata is cached on the clients
  - locks are polled to determine if metadata is stale or not
  - callbacks in SCSI-3 would be nice
- Small files cached in client via dinode stuffing
  - stuff data in same file system block as the dinode (disk inode)
- We would like to do much more explicit data and metadata caching on the device
  - shared by all systems
- fast, low overhead

Performance Testing (May 1997)

- Test scaling as both clients and disk arrays were added
- 4-client, 4-array configuration
  - Seagate disk drives, Ciprico 7000 arrays
  - SGI Challenges, Brocade switch, Prisa HAs
- Parameters tested
  - file transfer time and bandwidth
  - file size and request size are varied
- DLOCK performance:
  - about 1 millisecond for both Seagate and Ciprico

Performance Tests: Configuration

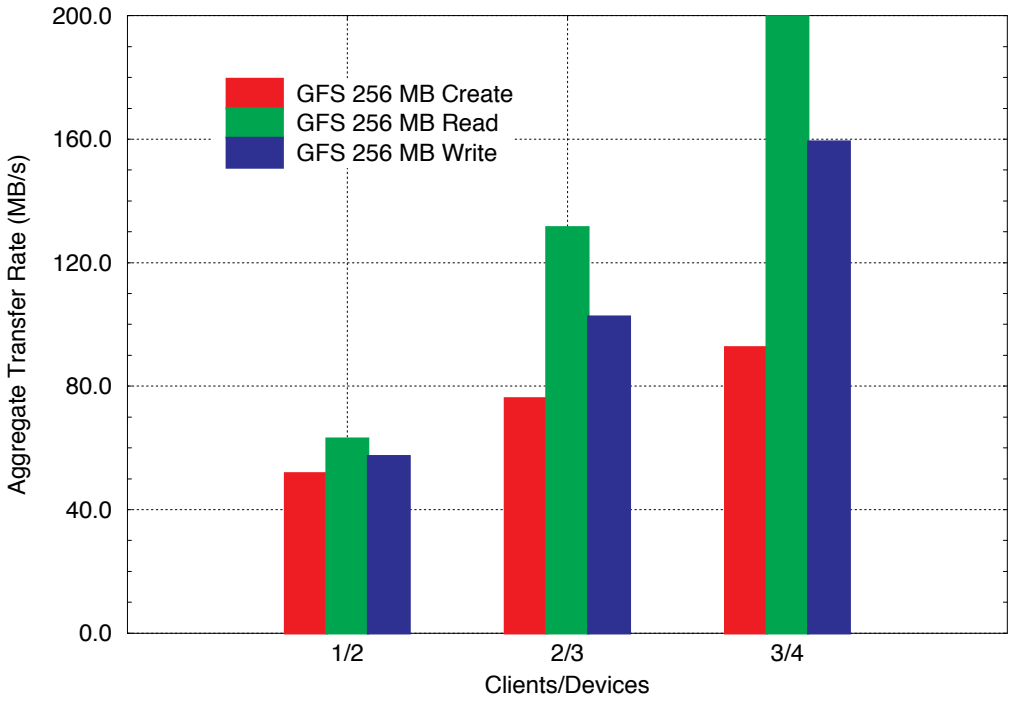


Performance Results: Single Host

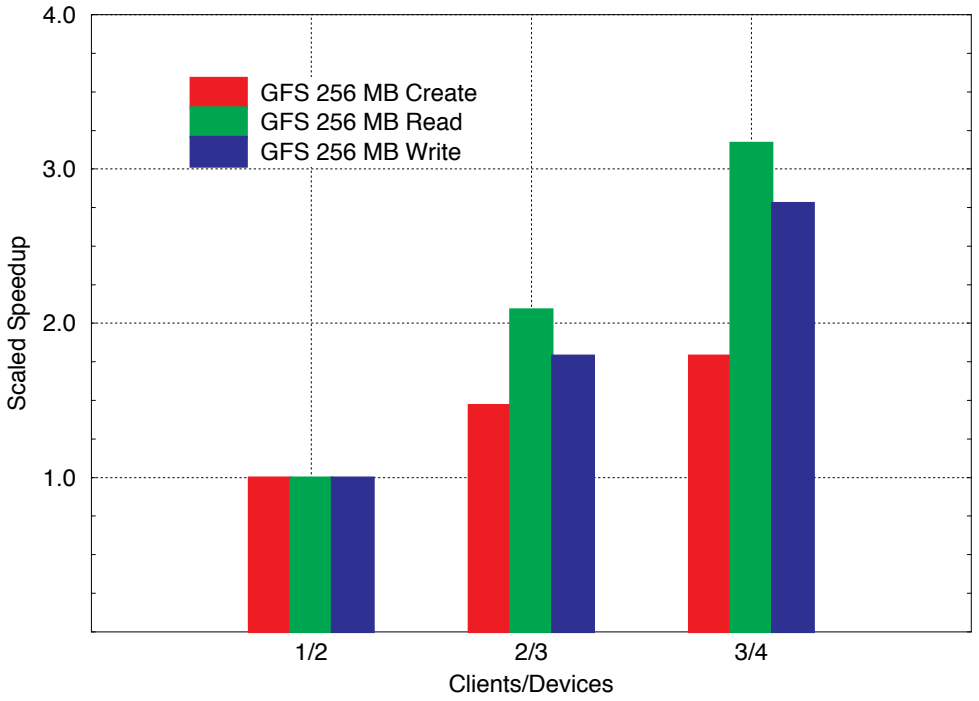
Sizes		**Single Disk**			**8-Wide Disks**			**Disk Array**		
File (MB)	Request (MB)	Raw (MB/s)	GFS (MB/s)	ratio (%)	Raw (MB/s)	GFS (MB/s)	ratio (%)	Raw (MB/s)	GFS (MB/s)	ratio (%)
4	2	10.5[1]	8.22	**78.3**	60.9[1]	24.8	**40.7**	74.8[1]	33.5	**44.8**
4	4	10.4[1]	8.38	**80.6**	58.5[1]	25.5	**43.6**	76.9[1]	33.8	**44.0**
16	8	10.5	9.81	**93.4**	59.1	43.4	**73.4**	78.2	61.1	**78.1**
16	16	10.4	9.90	**95.2**	58.5	45.4	**77.6**	79.0	61.0	**77.2**
128	8				64.8[2]	49.9	**77.0**	81.4[2]	68.5	**84.1**
128	16				64.9[2]	54.0	**83.2**	82.0[2]	73.1	**89.1**
256	8							81.4	70.1	**86.1**
256	16							82.0	74.8	**91.2**

[1] Estimated from 16 MB sequential performance tests
[2] Estimated from 256 MB sequential performance tests

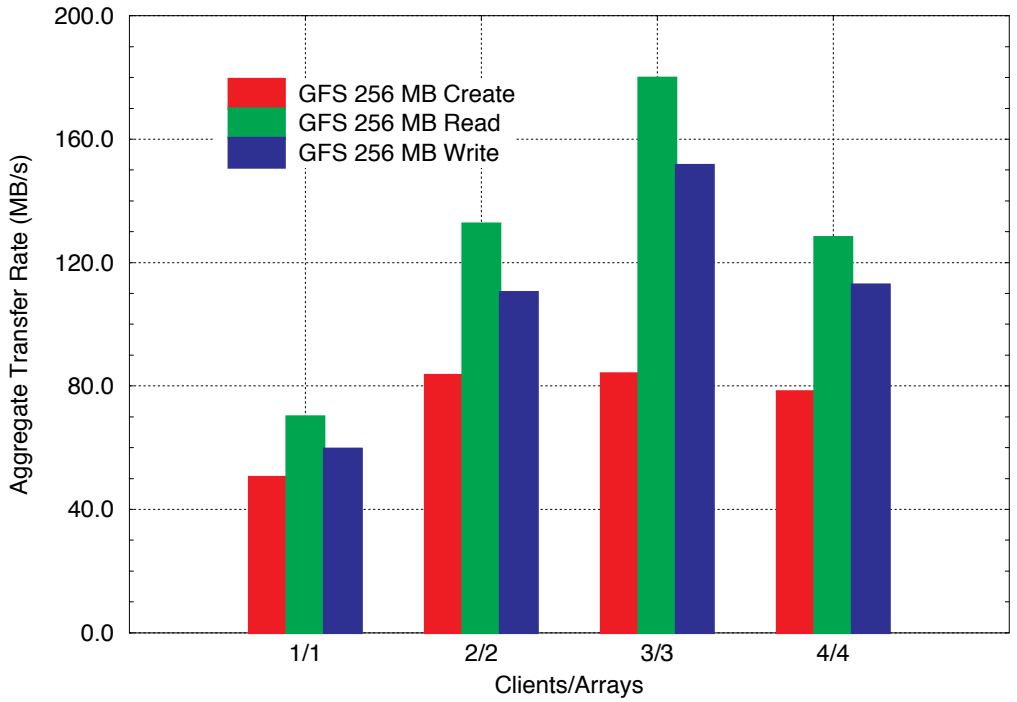
Performance Results: Transfer Rate



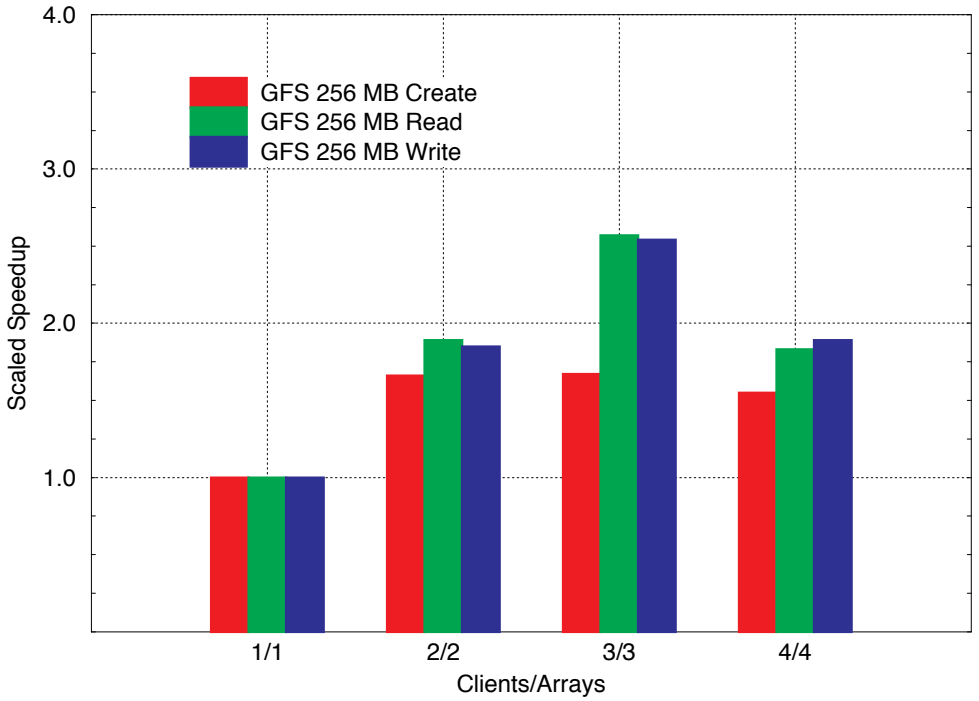
Performance Results: Speedup



Performance Results: Transfer Rate



Performance Results: Speedup



Summary of Performance Results

- Scaling was good with separate root directory device
  - locking and dir searches isolated to one device not transferring data
- Scaling was bad without separate root dir device
  - resource groups were not allocated uniformly across devices
  - DLOCK commands intermingled with SCSI data commands: this is bad
    - DLOCK sitting behing a big read or write will hurt performance of unrelated operations

Summary of Performance Results

- Bottom line: poor scaling caused by implementation not the GFS architecture
  - DLOCKs can and will be made faster since they are implemented in hardware
  - more caching where useful internal to GFS client
  - minimize use of locks in GFS client code
    - recently reduced number of locks used by factor of 3

Summary of Scaling Study

- Our results show the GFS approach to cluster file system design is at least feasible
- Architecture scales well (so far)
  - 4 clients and 4 arrays: Power Challenge
  - currently testing 8-way scalability on a cluster of 8 SGI O2's
  - goal: scales to 32 or 64 clients
    - past 64 clients will need hierarchy of GFS "domains" both for performance and to ease administration

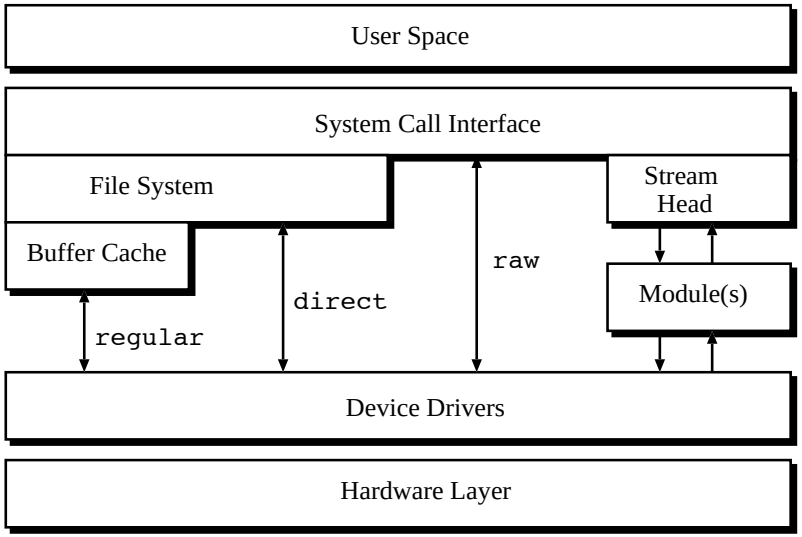
Summary of Scaling Study

- Lots of unanswered questions
  - What if you actually have decent caching on the devices
  - Faster locks with richer parallel semantics (multiple readers single writer)
  - Head-of-queue tagging of DLOCK commands
  - More efficient lock usage by GFS (reduction by factor of three of number of locks necessary)
  - Single client optimizations
  - More aggressive client caching
  - More sophisticated mapping of metadata to locks to reduce bottlenecks

Bandwidth Characterization

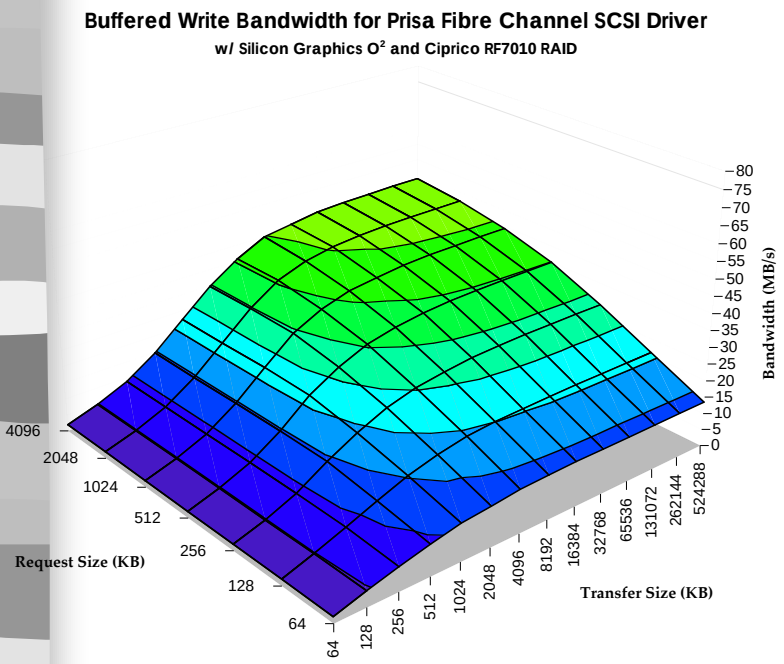
- Two parameter tests
  - Request size varied exponentially from 64 KB to 4 MB
  - Transfer, or file, size varied exponentially from 64 KB to 512 MB
- Test configuration
  - Single *Silicon Graphics O2* desktop workstation
  - *Prisa NetFX PCI-32* Fibre Channel host bus adapter
  - Single *Ciprico Rimfire* 7010 Fibre Channel RAID-3
  - *Brocade Silkworm* 16-port Fibre Channel switch

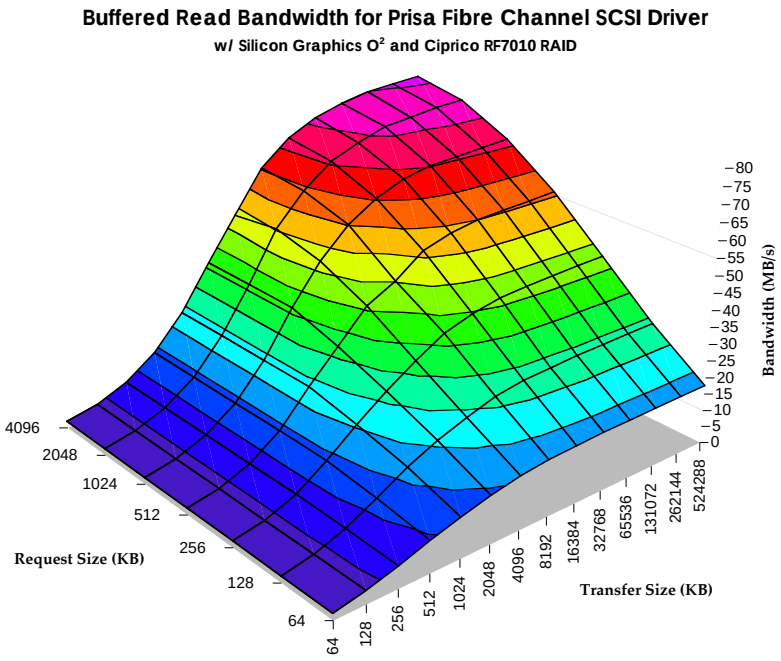
Bandwidth Characterization



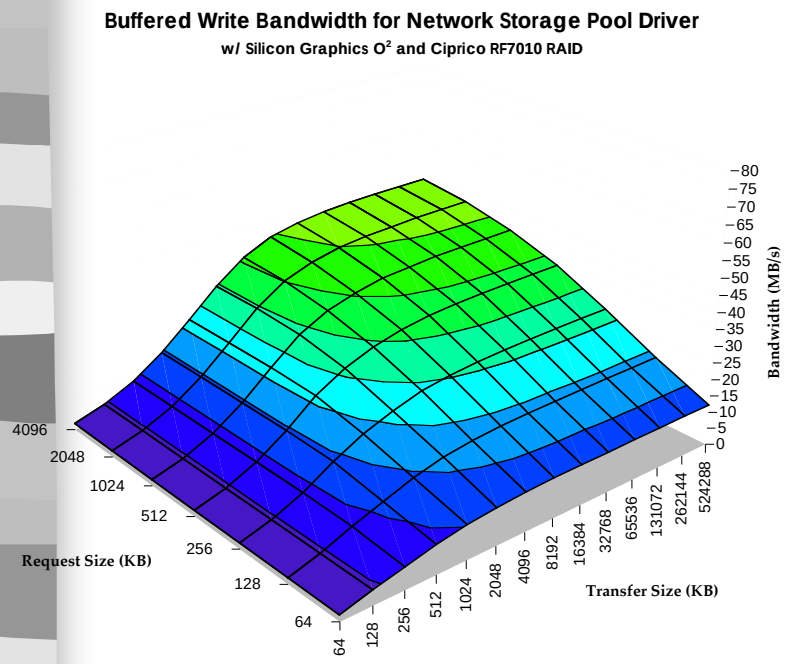
- Characterize the bandwidth for the given test configuration for each subsystem
- Quantify the amount of overhead incurred by each subsystem by examining bandwidth losses

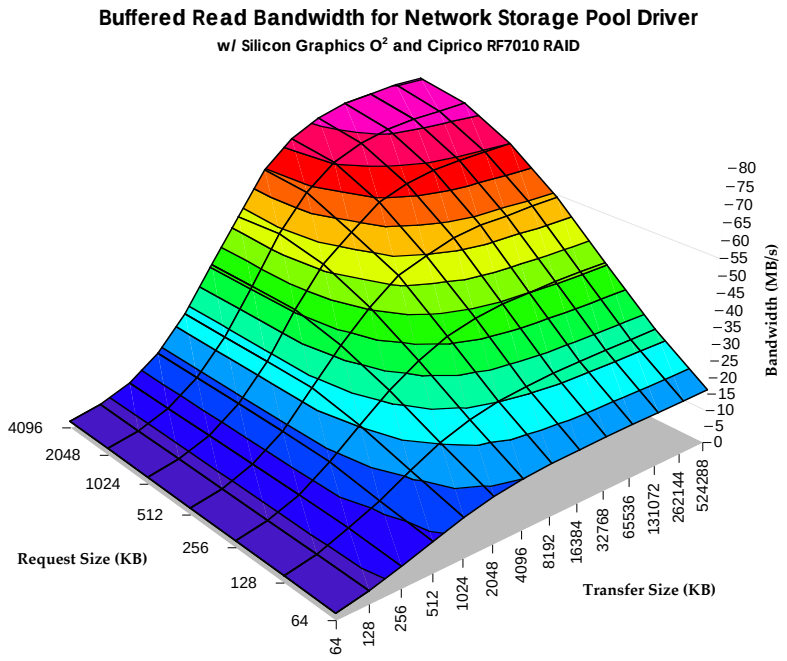
Host Adapter Bandwidth





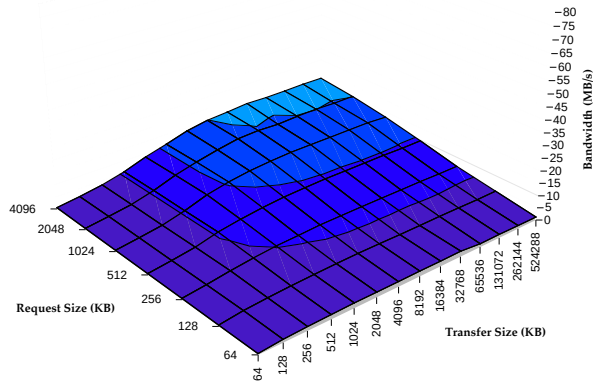
Network Storage Pool Bandwidth



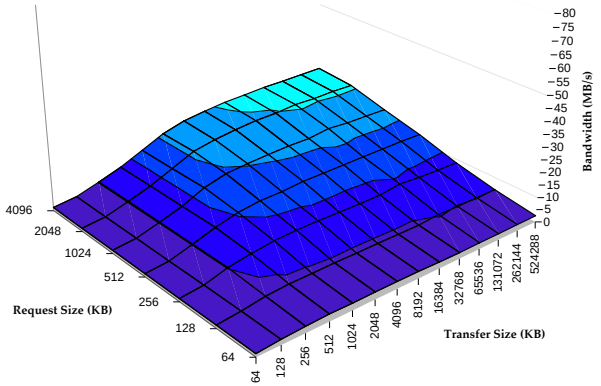


GFS Bandwidth

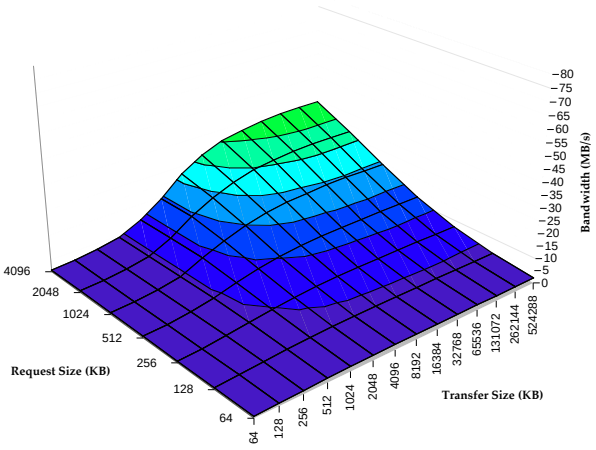
Buffered Write Bandwidth for Global File System
w/ Silicon Graphics O² and Ciprico RF7010 RAID



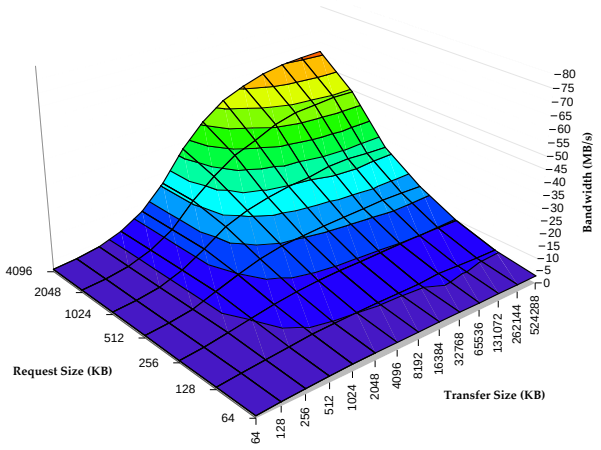
Buffered Read Bandwidth for Global File System
w/ Silicon Graphics O² and Ciprico RF7010 RAID



Direct Write Bandwidth for Global File System
w/ Silicon Graphics O² and Ciprico RF7010 RAID

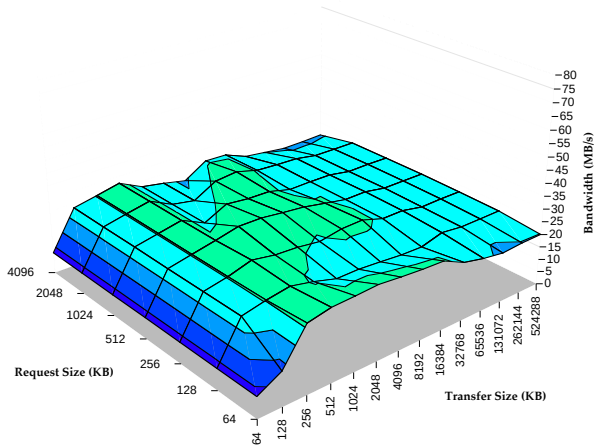


Direct Read Bandwidth for Global File System
w/ Silicon Graphics O² and Ciprico RF7010 RAID

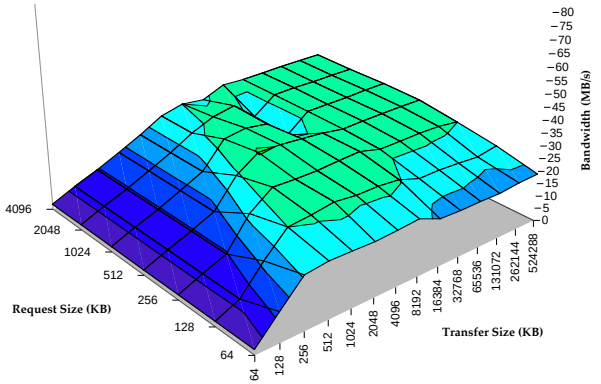


XFS

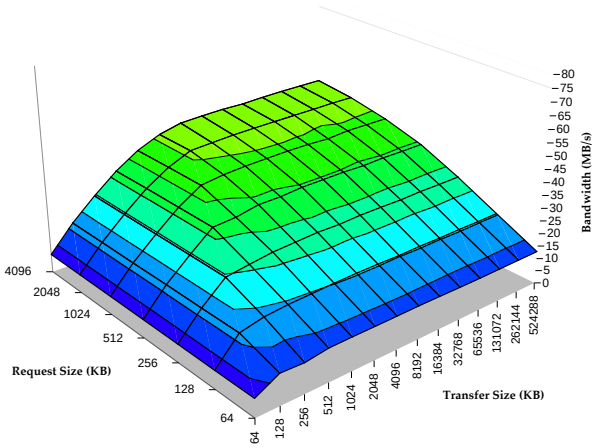
Buffered Write Bandwidth for XFS
w/ Silicon Graphics O^2 and Ciprico RF7010 RAID



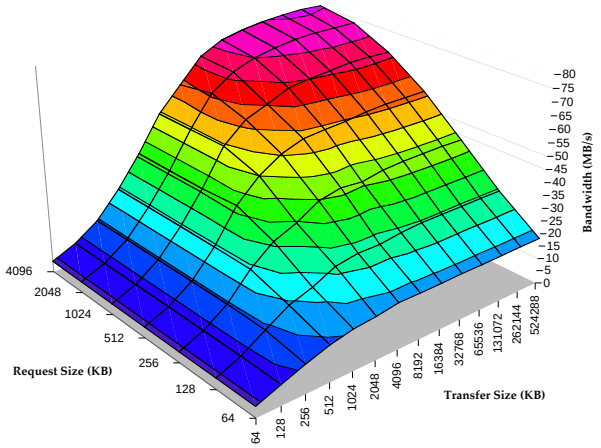
Buffered Read Bandwidth for XFS
w/ Silicon Graphics O^2 and Ciprico RF7010 RAID



Direct Write Bandwidth for XFS
w/ Silicon Graphics O^2 and Ciprico RF7010 RAID



Direct Read Bandwidth for XFS
w/ Silicon Graphics O^2 and Ciprico RF7010 RAID



Relative Subsystem Efficiencies

	Relative Efficiency			
	Prisa NetFX Driver	**Network Storage Pool Driver**	**Global File System**	**XFS**
Buffered I/O				
Writes				
Mean	**100.0%**	**95.6%**	**24.9%**	**87.8%**
Standard Deviation	0.00	0.03	0.08	0.90
Minimum	100.0%	90.4%	9.7%	43.7%
Maximum	100.0%	100.5%	38.8%	475.0%
Reads				
Mean	**100.0%**	**97.2%**	**25.7%**	**64.5%**
Standard Deviation	0.00	0.02	0.12	0.58
Minimum	100.0%	92.3%	14.7%	31.8%
Maximum	100.0%	103.6%	68.8%	319.0%
Direct I/O				
Writes				
Mean			**29.6%**	**112.3%**
Standard Deviation			0.19	0.53
Minimum			16.4%	94.6%
Maximum			79.1%	375.8%
Reads				
Mean			**31.4%**	**115.4%**
Standard Deviation			0.17	0.29
Minimum			16.9%	99.4%
Maximum			74.5%	215.9%
Overall				
Mean	**100.0%**	**96.4%**	**27.6%**	**90.0%**
Standard Deviation	0.00	0.03	0.15	0.63
Minimum	100.0%	90.4%	9.7%	31.8%
Maximum	100.0%	103.6%	79.1%	475.0%

The SCSI Device Lock Command

- Technical Description
- Current Implementation and Proposed Improvements
- Standardization
- Collaboration with Seagate

GFS: Future Plans

- NT port in collaboration with industry
- Open Source version on Linux
- IRIX version given away to film and video industry, others

Overview and Conclusions

- New open, high-volume, relatively low-cost NAS interfaces make widespread use of shared (disk) file systems potentially possible
- More research needed:
  - Previous systems have not succeeded in scaling past a small number of clients
  - Distributed coherence, caching, and recovery issues must be resolved and the solutions standardized by the industry
    - perform in client, file manager, network, or device
  - NASD keys in on two most important issues
    - objects and security
  - Promise is better performance, better storage management by the devices, and new applications in parallel clusters