Filesystems for Network-Attached Secure Disks (NASD)

Garth Gibson
D. Nagle, K. Amiri, H. Gobioff, E. Riedel, J. Zelenka
Computer Science and Computer Engineering, CMU

What functions should storage offer?
Taxonomy for Network-Attached Storage (NAS)

Server-Attached, Server-Integrated Disk (SAD, SID)

- (specialized) workstation running file server code

Networked SCSI (NetSCSI)

- minimal differences from SCSI; manager inspects requests

Network-Attached Secure Disk (NASD)

- new (SCSI-4) interface enables direct, preauthorized access

Contrasting extremes: NetSCSI vs. NASD

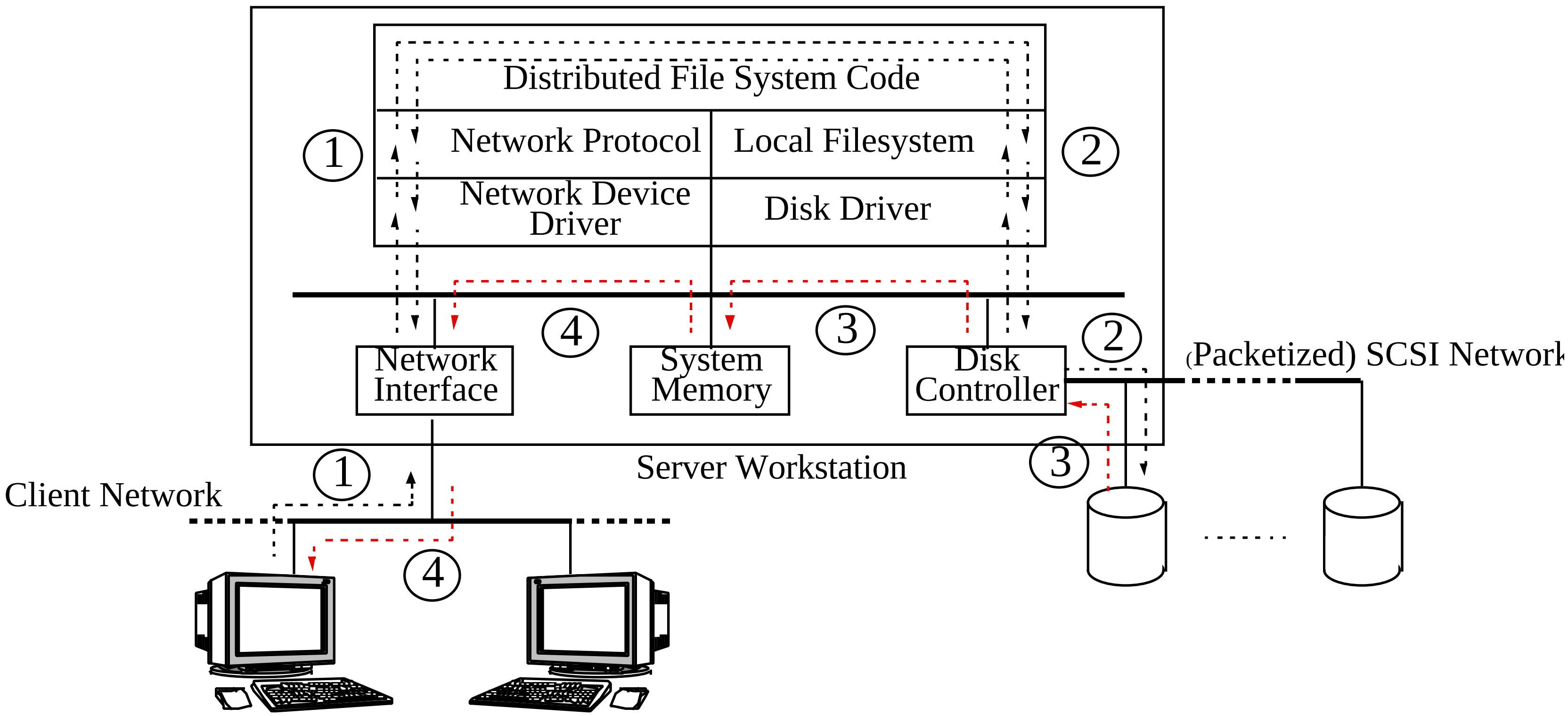
- both scale bandwidth with large, striped accesses
- what impact on workloads of current LAN file servers?

Problems with current Server-Attached Disk (SAD)

Store-and-forward data copying thru server machine

- translate & forward request (1,2), store & forward data (3,4)

Limited bandwidth, slots in low-cost server machine

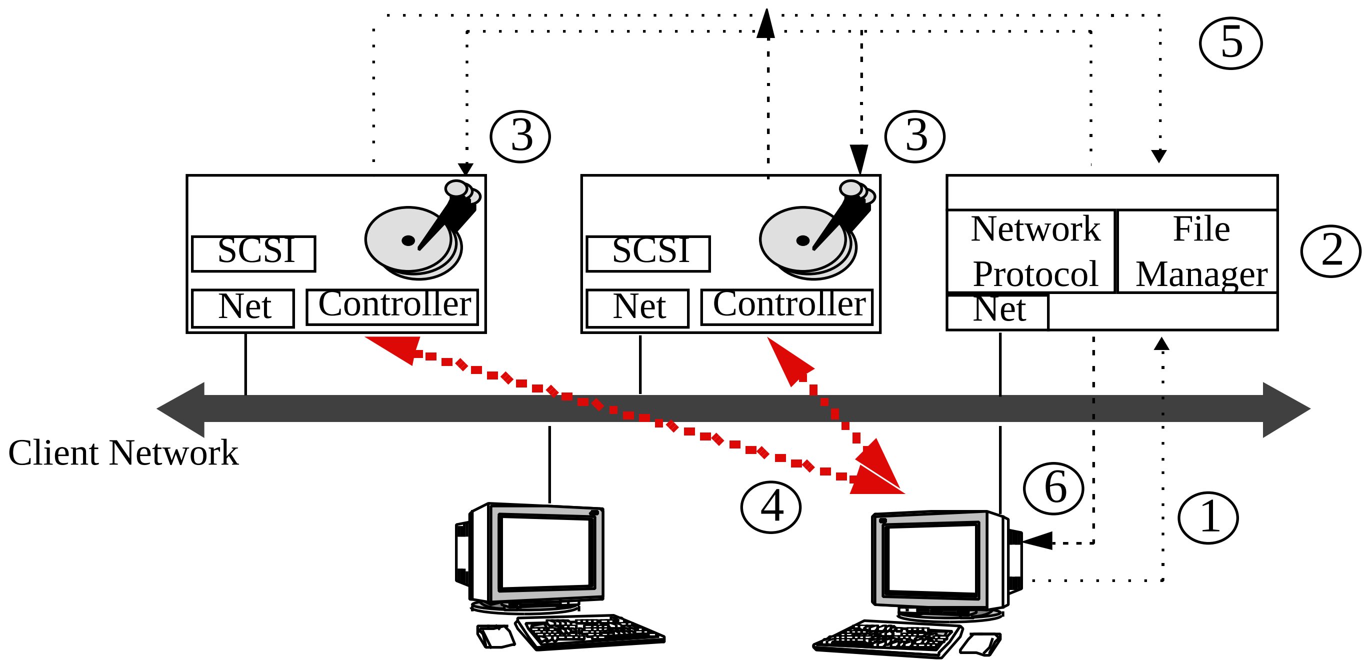


Networked SCSI (NetSCSI)

Minimize change in drive HW, SW, IF: RAID-II

- server translates (2) and forwards (3) request (1)
- drive delivers data directly to client (4)
- drive status to server (5), server status to client (6)

Scalable bandwidth through network striping

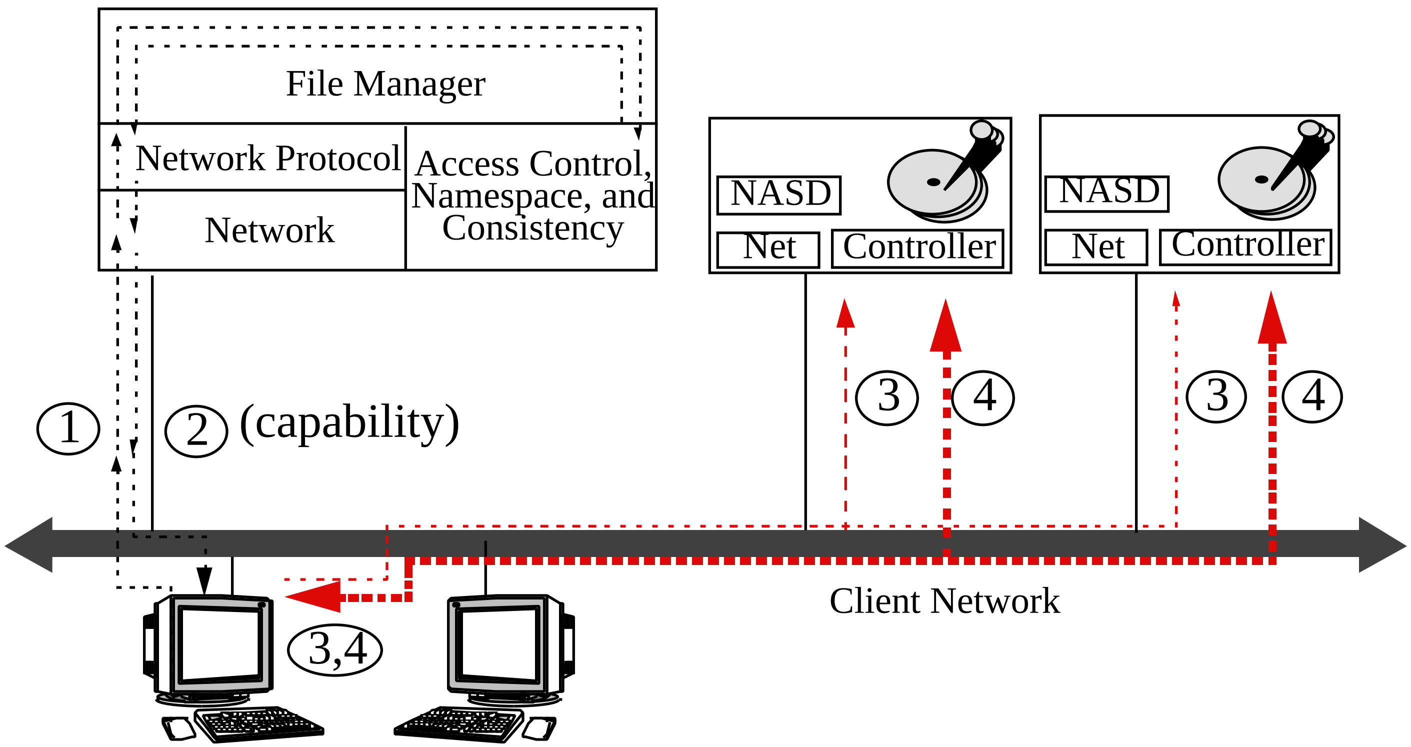


Network-Attached Secure Disk (NASD)

Avoid file manager unless policy decision needed

- access control once (1,2) for all accesses (3,4) to drive object
- spread access computation over all drives under manager

Scalable BW, off-load manager, fewer messages



Impact of NASD vs. NetSCSI on current file systems

Analytic & trace-driven agree; talk presents analytic

Analyze FS traces; instrument SAD server, count instrs

Model change in operation counts and costs at manager

For SAD, use numbers as measured

For NetSCSI, data transfer is off-loaded

- manager does work of 1-byte access per request
- attribute/directory assumed no less work

For NASD, off-load file write and file/attr/dir read

- updates to attributes/directory are no less server work
- manager must do new “authorization” work when file opened (synthesized as first touch after long inactive)

NFS on network-attached storage projections

Berkeley NFS traces [Dahlin94] (230 clients, 6.6M reqs)

Directory/attributes dominate SAD manager work

NetSCSI, therefore, little benefit for manager load

NASD off-loads over 90% of manager load

NFS Operation	Count in top 2% by work (thousd)	SAD		NetSCSI		NASD	
		Cycles (billions)	%of SAD	Cycles (billions)	%of SAD	Cycles (billions)	%of SAD
Attr Read	792.7	26.4	11.8%	26.4	11.8%	0.0	0.0%
Attr Write	10.0	0.6	0.3%	0.6	0.3%	0.6	0.3%
Block Read	803.2	70.4	31.6%	26.8	12.0%	0.0	0.0%
Block Write	228.4	43.2	19.4%	7.6	3.4%	0.0	0.0%
Dir Read	1577.2	79.1	35.5%	79.1	35.5%	0.0	0.0%
Dir RW	28.7	2.3	1.0%	2.3	1.0%	2.3	1.0%
Delete Write	7.0	0.9	0.4%	0.9	0.4%	0.9	0.4%
Open	95.2	0.0	0.0%	0.0	0.0%	12.2	5.5%
Total	**3542.4**	**223.1**	**100.0%**	**143.9**	**64.5%**	**16.1**	**7.2%**

AFS on network-attached storage projections

CMU AFS traces (60-250 clients, 1.6 M reqs)

Data transfer dominates SAD

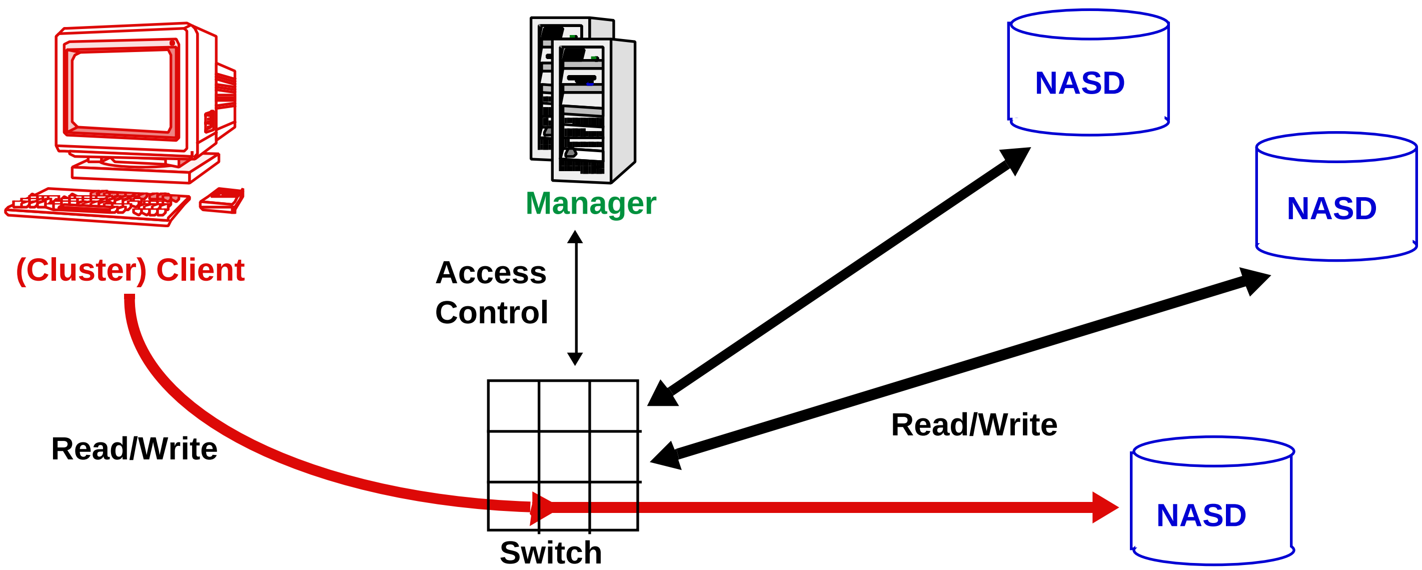
NetSCSI is able to reduce manager load by 30%

NASD is able to reduce manager load by 65%

AFS Operation	Count in top 5% by work (thousand)	SAD		NetSCSI		NASD	
		Cycles (billions)	%of SAD	Cycles (billions)	%of SAD	Cycles (billions)	%of SAD
FetchStatus	770.5	98.6	37.9%	98.6	37.9%	0.0	0.0%
BulkStatus	91.3	36.6	14.1%	36.6	14.1%	0.0	0.0%
StoreStatus	16.2	3.1	1.2%	3.1	1.2%	3.1	1.2%
FetchData	193.7	83.7	32.1%	24.8	9.5%	0.0	0.0%
StoreData	23.1	15.1	5.8%	3.0	1.1%	3.0	1.1%
CreateFile	12.1	3.7	1.4%	3.7	1.4%	3.7	1.4%
Rename	6.4	1.8	0.7%	1.8	0.7%	1.8	0.7%
RemoveFile	14.6	4.8	1.9%	4.8	1.9%	4.8	1.9%
Others	57.3	13.0	5.0%	13.0	5.0%	13.0	5.0%
Open	480.8	0.0	0.0%	0.0	0.0%	61.5	23.6%
Total	**1665.9**	**260.5**	**100.0%**	**189.4**	**72.7%**	**90.9**	**34.9%**

CMU's Functional Definition of NASD

- **Direct** client/drive **transfer** in networked environment
- **Asynchronous** filesystem **oversight** of rights, semantics
- **Cryptographic** capabilities ensure command integrity
- **Self-management** by more abstraction, independence
- **Extensible** features for application, not just client OS



NASD Interface Design: Storage Objects

Layout is best (actually) done below SCSI-4

- real-time support possible; accurate geometry
- simplifies, strengthens transparent performance optimization

Direct access means file layout known to client or drive

- don't trust client, so drive has map of stored object
- rejected temporary map (DVD), more drive self-knowledge

Drive serves storage objects on behalf of manager

Objects have attributes

- Inodes: (name), size, protection, type, timestamps, layout
- what attributes should NASD objects support?

NASD Interface Design: Storage Objects Con't

Synchronous metadata updates must be done in drive

- logical and physical sizes
- modify timestamps: honest and user resettable

Object layout guidance from higher level

- sequential within object requests contiguous (can preallocate)
- related objects can be clustered with "nearby to" attribute

Capacity increasing optimizations exposed to manager

- honest capacity-consummed attribute

Attributes are extensions of object name

- give higher level (filesystem) uninterpreted attribute
- big enough to contain another object name (soft link)

NASD Sharing: Multiple File Managers

Split capacity among different managers: partions

- **managers can use attributes differently; no need to integrate**
- **boundary should be soft: resize partition should be fast**
- **flush partition enables fast acquiese**
- **partition key hierarchy: (partition key, working keys)**

Operation	Arguments	Description
createpartition	partition	create a new partition (zero-sized)
removepartition	partition	remove a partition
resizepartition	partition, new size	set a partition’s size
flushpartition	partition	commit any cached writes for a partition to stable store
setpartitionkey	partition, key name, key value	set “master” key for a partition

NASD Interface Design: Well-known Objects

SCSI mode sense/select replacement

- published format and interpretation

Per-drive and per-partition separated

Partition table of contents (mini-disk directory)

Assist simple boot code with easily found "first object"

- replace with uninterpreted fields in partition control object?

Name	Description
Drive Control	basic control information for drive: clock, physical parameters, extensions supported, bytes allocated
Partition Control	basic control information for partition: current size, byte usage, number of object supported, number of objects allocated
Partition Contents	read-only list of identifiers of NASD objects allocated in the partition
First Object	ordinary, read/writable NASD object which is always created with length 0 when a partition is initialized

Adapting Filesystems to NASD

Reorganize decomposition of function (aka port)

Primitives become drive responsibility

- data transfer, synchronous/automatic metadata updates

Policy remains manager responsibility

- namespace definition/navigation
- access control policy
- client cache managment
- multi-access atomicity

Managers retain control through capabilities

- exploiting attributes for naming and revocation
- restricting client operations to protect “set attribute”

Mapping Filesystem to NASD Objects

Objects: attributes, access control, clustering

Simple model

- each file and directory bound to separate NASD object
- file attributes inherit object attributes (times, logical size)

Multiple objects per file?

- internal structure: database pages, mpeg group-of-pictures
- NASD striping, redundancy

Multiple files/directories per object?

- probable contiguity, prefetching; shared metadata overhead
- capabilities can be restricted to object region

NFS, AFS simple model; Striped NFS multiple per file

NFS NASD Prototype

Files, directories, links are all separate NASD objects

Unix attributes are stored in fs-specific field

Partition is a filesystem; partition root is filesystem root

NFS handle is well-known convolution of
NASD identifier and drive identifier

Server only entity to directly reads/write directories

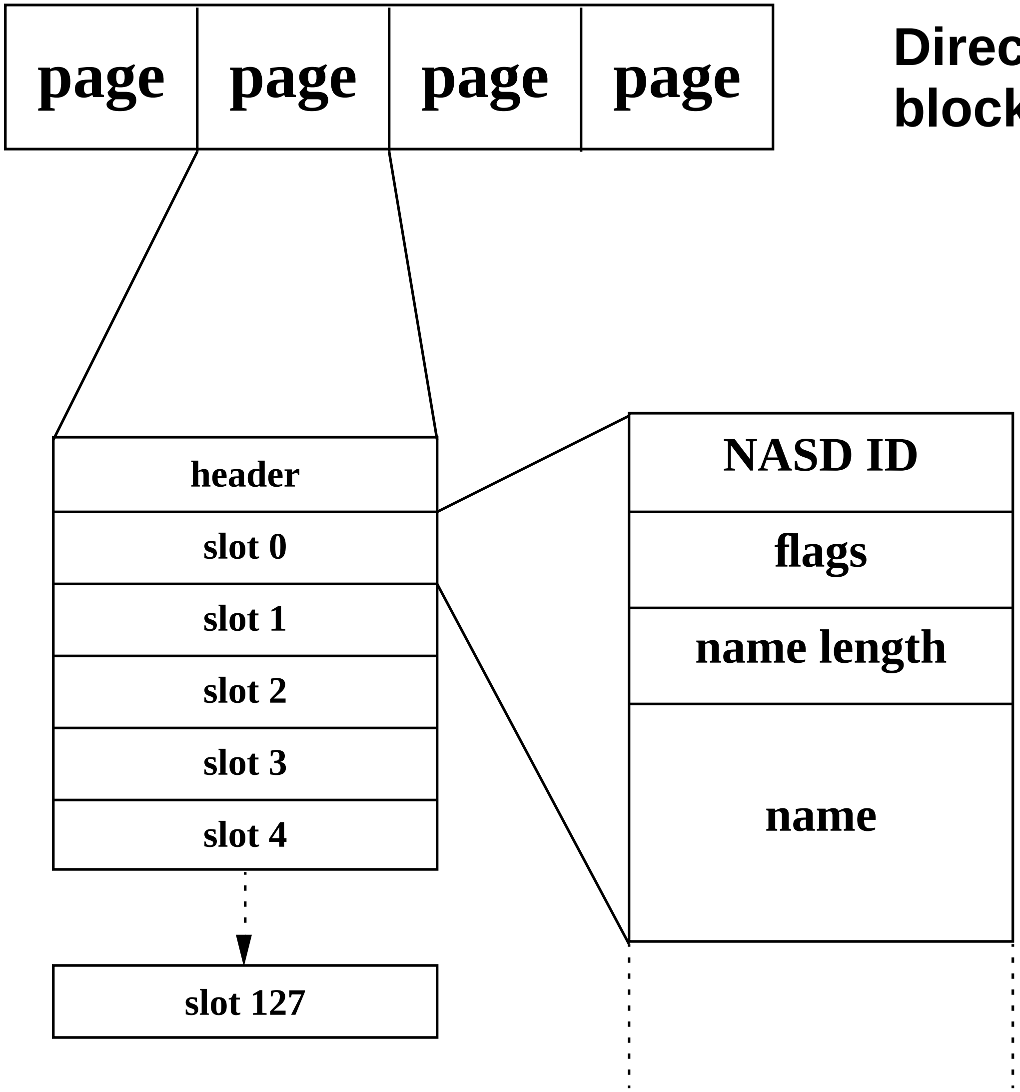
- No client changes directory contents or directory attributes
- Directory objects cached in manager

No write caching (asynchronous writeback)

- stick to NFS semantics for comparison fairness and simplicity

Mount list (drives, partitions) only manager local state

Directory Format



Directories divided into blocksized pages

slots hold entries

entries with long names span multiple slots

NASD NFS Performance

Drive, manager implemented in Digital UNIX kernel

Communication via DCE RPC

- **pipes for bulk transfer, application marshalling & crypto (off)**

Andrew Benchmark

- **phase 1 create directories**
- **phase 2 copy files**
- **phase 3 recursive directory stat**
- **phase 4 scan each file (grep)**
- **phase 5 compilation**

One NFS server/file manager with multiple disks

Each NASD a separate filesystem (no sharing)

No READDIRPLUS for NASD

NASD NFS Performance

Andrew Benchmark (run seconds for each phase)

	phase 1	phase 2	phase 3	phase 4	phase 5
1x1 SAD	1.0	3.3	2.3	2.7	19.3
1x1 NASD	0.5	3.3	4.2	5.5	18.8
5x5 SAD	1.9	10.9	4.2	3.7	22.7
5x5 NASD	0.5	3.8	5.3	5.8	18.7

Read and write bandwidth (KB/s)

	8k read	8k write	64k read	64k write
1x1 SAD	2261.8	2601.3	6392.3	825.38
1x1 NASD	4099.9	5253.8	4399.2	3506.7
5x5 SAD	1851.0	1750.0	5140.6	726.0
5x5 NASD	4084.1	4952.1	4236.0	3764.4

AFS NASD Prototype

AFS built on UNIX FS inode interface

- UFS inode interface replaced with NASD object interface

Base design similar to NFS except

- directory objects read, parsed, cached at clients
- AFS cache coherence protocol independent (almost) on NASD
- AFS quota enforced by capability escrow (using write range)

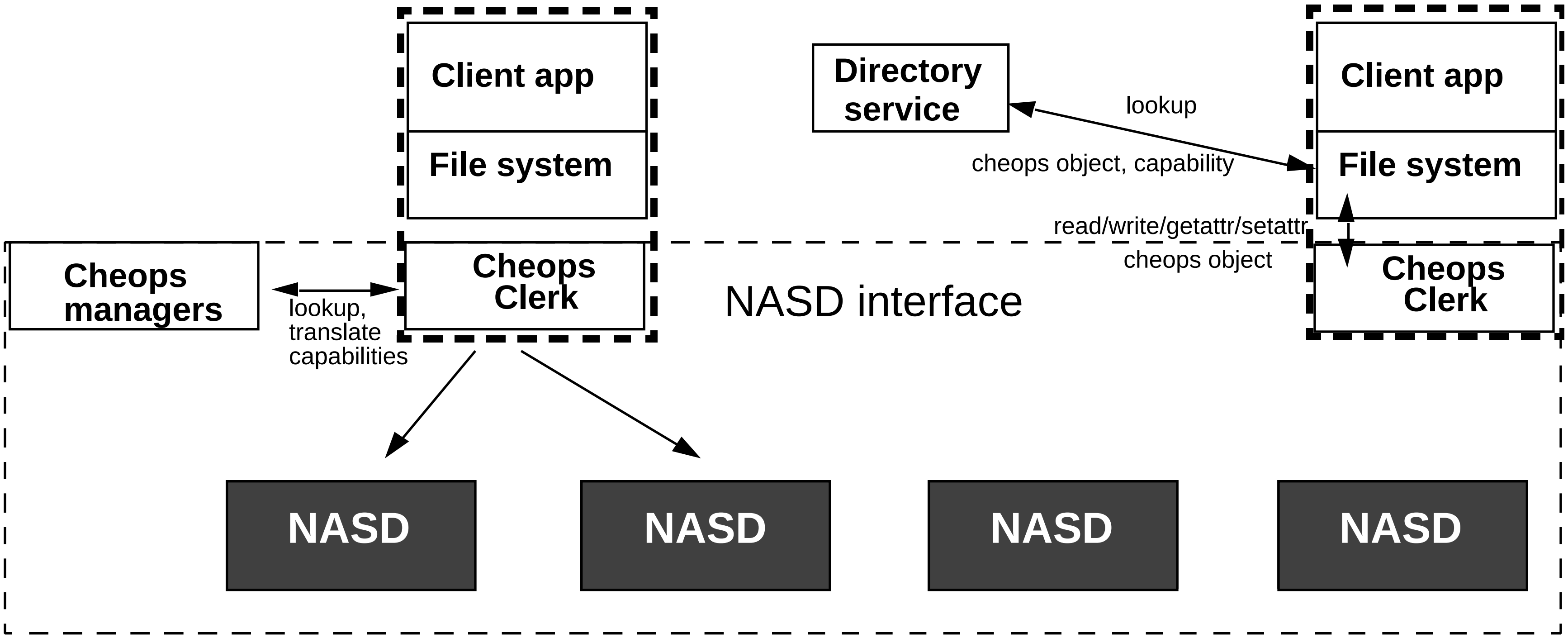
Operation disposition

- to drive: FetchStatus, BulkStatus, FetchData (w/cap),
  StoreData (w/cap)
- Read w/o cap: GetCap (callback, attributes),
  (GetAttr from drive), FetchData
- Write w/o cap: GetWCap (break callbacks, short lifetime),
  StoreData,
  ReturnCap (signals early re-enable of callbacks)

Cheops: Striping storage middleware

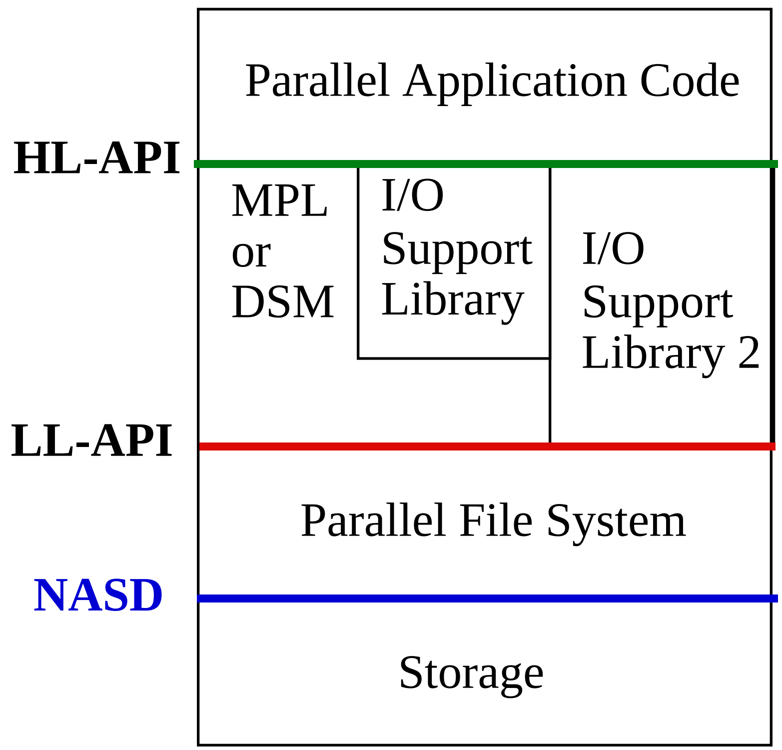
Transparent, scalable bandwidth, RAID, optimistic client synchronization (fs-specific attributes)

Storage management architecture parallels file management architecture (uses capabilities)



SIO Parallel File System Low Level API

- Scatter/Gather
- Asynch
- Collective Transfer
- Copy-on-write
- Client cache control
- Hints to/from storage



MPL = Message Passing Library
DSM = Distributed Shared Memory

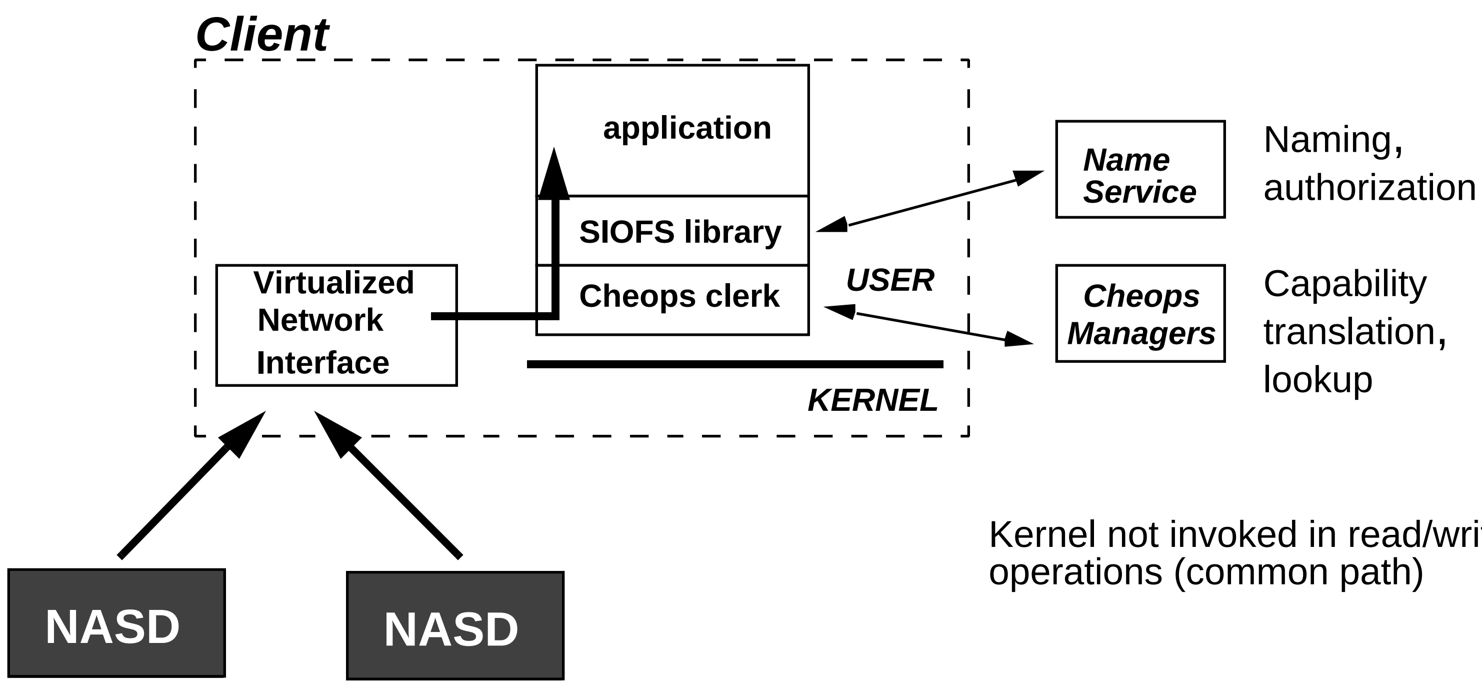
SIO PFS on Cheops on NASD

Client asking for service pays for it (synchronizer)

- striping, RAID, consistent caches, collective operations

Entirely user level, incl. messaging, for low latency

Impl exploits local FS (AFS) for file-level semantics



Kernel not invoked in read/write operations (common path)

Recap: NASD Filesystems are Policy Servers

Direct transfer for wire-once, scalable bandwidth

- NetSCSI for large object bandwidth
- NASD for object bandwidth and server offloading

NASD filesystems serve policy (async oversight)

- namespace, access control, consistency, atomicity
- capabilities encode policy, metadata; crypto integrity
- capabilities cause drive to understand variable length object

Storage management middleware

- clients pay for requested synchronizing semantics
- striping, RAID, incremental capacity, migration
- optimistic synchronization using fs-specific attributes