Object-Based Storage

NAS Industry Conference

Alexander Krimkevich

Panasas, Inc

New Object Storage Architecture

- An evolutionary improvement to standard SCSI storage interface
- Raises level of abstraction: Object is container for “related” data
  - Storage understands how different blocks of a “file” are related
- Offload most datapath work from server to intelligent storage
  - Enable through security w/in storage device

Block Based Disk

Operations:

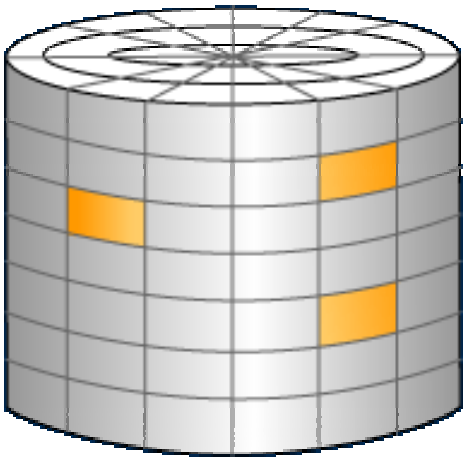
Read block
Write block

Addressing:

Block range

Allocation:

External



Object Based Disk

Operations:

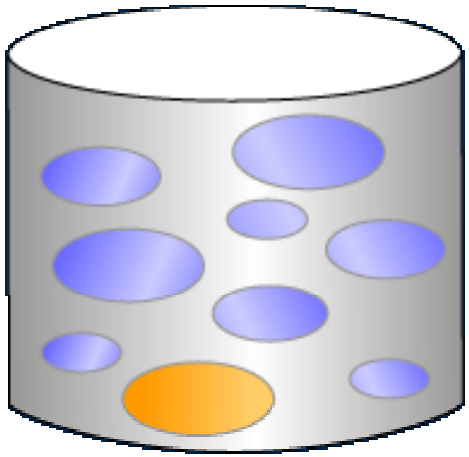
Create object
Delete object
Read object
Write object

Addressing:

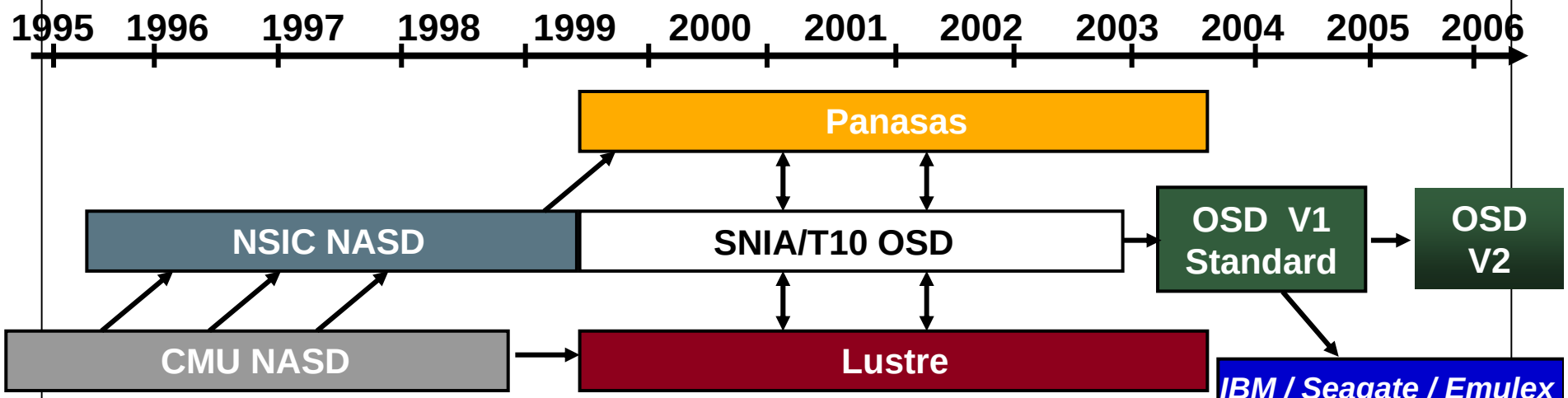
[object, byte range]

Allocation:

Internal



Object Storage Standardization



- Started with NSIC NASD research 1995-1999
  - CMU NASD +
  - Eventually became SNIA Technology working group in 1999 (26 members strong)
- 2001 Moves to SNIA/T10 working group
- 1/2005: ANSI ratifies V1 T10 OSD standard (ANSI/INCITS 400-2004)
  - SNIA TWG already working on V2 features
    - Snapshots, import/export, multi-object capabilities and extended attributes

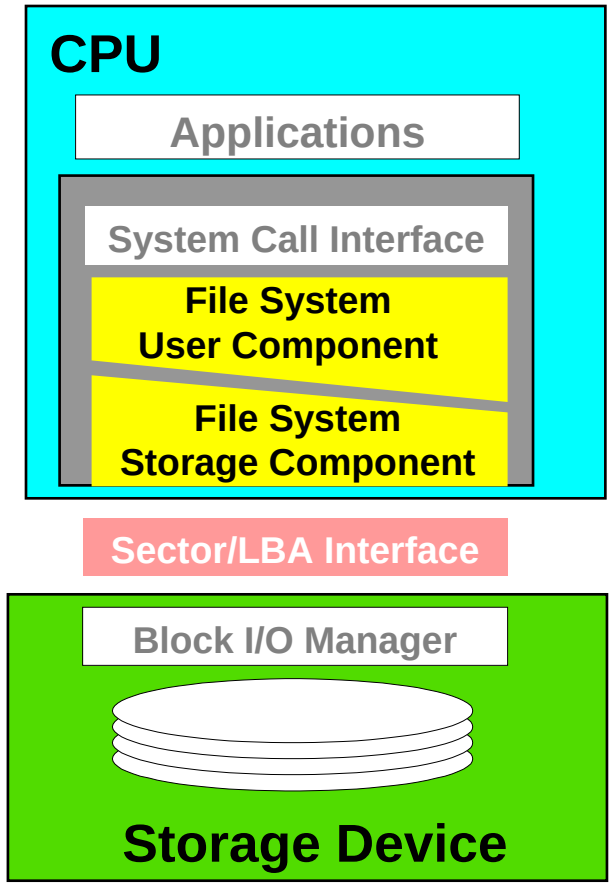
KEY: *Resulting standard will built options for customers looking to deploy objects*

Agenda

- Object Introduction
- T10 Object Standard
- Storage System Architectures
- Files over Objects
- Security
- Panasas File System
- Performance
- pNFS

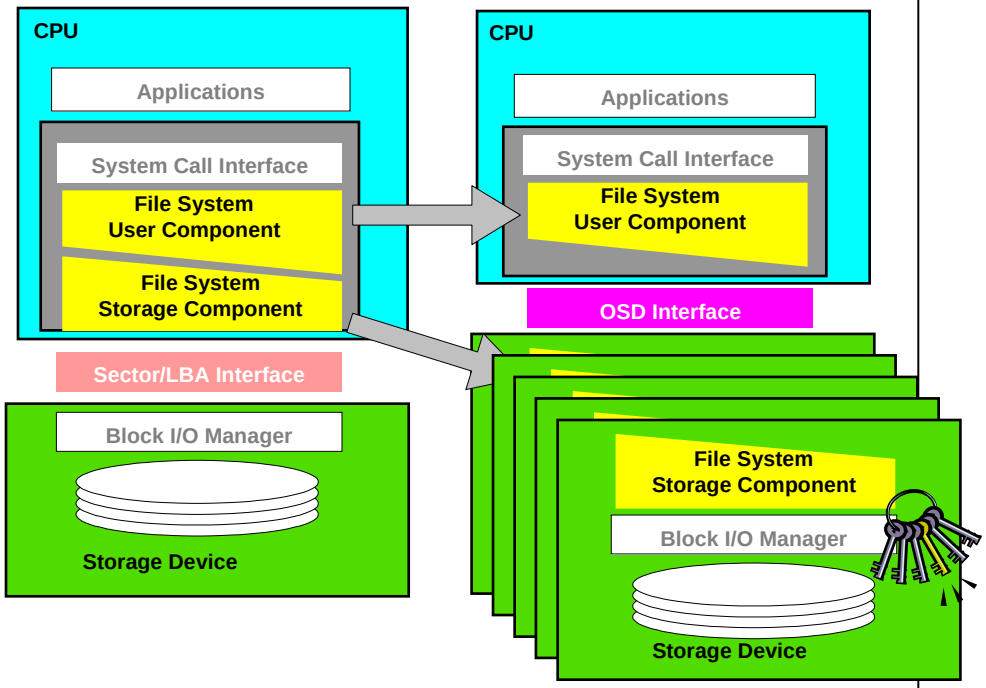
File System: Datapath and Policy

- OS / File System
  - Translates file to sector mapping
    - OS is responsible for performance of storage device
      - Layout, organization of storage requests
      - Prefetching to the clients
    - Disk has primitive caching and prefetching algorithms
      - Based on physical layout
      - No knowledge of application
    - Best way to improve IO performance … avoid touching the platter at all cost
    - Next best way, layout contiguous data on contiguous sectors/tracks
  - OS performs file system policy checks
    - User authorization
    - Permission checks (read, write)
    - File attributes



Object-based Storage (OSD)

- At 10,000 feet, migrate the lowest part of the client file system into the storage device
  - Storage understands logical data layout and not just sectors
- This fundamentally changes the interface of storage from blocks to "objects"
  - Re-divides responsibility for managing the access to data
  - Assigns additional activities to the storage device (space management)
- Offloads inode processing to OSD
  - Management of block to file mapping distributed to storage device
  - Up to 90% of workload offloaded as compared to traditional file servers



Overcoming Block-based Storage Limitations

- Performance
  - All knowledge of application is in the client(s)
    - File layout, client access patterns, specific application streams
  - All knowledge of storage is in the disks
    - Caching, prefetching, head scheduling
- File System Policy
  - All file system policy decisions are made in the client
  - Storage has no means to interpret or enforce file system policy
    - Storage is a dumb device
    - But most storage devices have significant processing power

➔ Migrate knowledge to storage device

- Embed layout information in the storage device
- Provide storage with information about applications' behavior
- Allow storage to better cache prefetch and schedule

➔ Keep file system policy in servers

- But allow storage to ENFORCE policy decisions

➔ Separate datapath from policy

Motivation for Object Storage

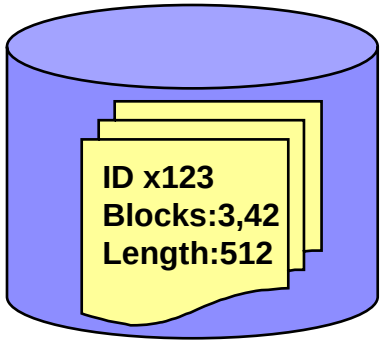
A highly scalable, interoperable, shared storage system

- Improved device and data sharing
  - Storage-dependent metadata moved to device
- Improved performance and scalability
  - Devices directly handle client requests
- Improved security
  - Object security w/ application-level granularity
- Improved storage management
  - Storage devices can be self-managed, policy-driven storage

Object Storage Model

- An object is a logical unit of storage
  - Lives in flat name space with ID
  - Contains application data AND **metadata**
    - Metadata: block allocation, physical length (similar to inode)
  - Possess user accessible **attributes**
    - QoS requirements, capacity quota, etc.
- Objects have file-like methods: open, close, read, write
- Three types of objects:
  - Root Object (one per device)
  - Group Object (a “collection” of objects)
  - User Object (for user data)
- Objects can take many forms
  - Intelligent disk drives
  - A storage appliance
  - Storage controllers

An Object



Objects vs. Blocks

- OSD is a Secure Shared Device
  - Multiple hosts (clients) can access OSD simultaneously
  - OSD enforces access rights with capability check
- Object Interface is higher level, so fewer round trips
  - Create, Delete, Read, Write, GetAttributes, SetAttributes
  - Compact location information. (objID vs. list of blocks)
- Object (File) operations map to several I/Os on data and metadata
  - Private disks: hosts manage blocks and do local I/O
  - Shared disks: block mgr is bottleneck; hosts do more remote ops & scalability is issue
  - Object Storage Device: hides block management, better scalability

T10 Standard Object Types

- Object Hierarchy and Types
  - User-object
    - Up to 2^{64} per partition
    - Basic storage unit for user data + attributes
  - Partition-object
    - Up to 2^{64} per device
    - Container for user-objects that share security & space mgmt characteristics
  - Root-object
    - One per device, defines device characteristics (e.g., device capacity)
    - Container for group objects
  - Collection-object
    - Up to 2^{64} per partition
    - Fast index that applications can use to create arbitrary set of user-objects
    - Useful for support fault-tolerance/recovery
    - V2 support for multi-object operations (e.g., delete, query, list)

T10 OSD Object Attributes

- Every object has a set of associated attributes
  - Stores various information (e.g., capacity used, last-access time, object_version_number)
  - Some attributes defined by standard ... others available for higher-level software
- Attribute pages different by object type and categories
  - Defined by T10 OSD Standard, other standards, manufacturers, etc
- Attribute Pages
  - 2^32 attribute pages
    with 2^32 attributes per page
  - All attributes virtually exist on create
  - Attributes written with explicit setattr()
  - Attributes read with explicit getattr()
- Most commands embed set- or getattr()
  - Minimizes messaging

T10 OSD Commands

- Basic Protocol
  - READ
  - WRITE
  - CREATE
  - REMOVE
  - GET ATTR
  - SET ATTR

very basic

space mgmt

attributes
- timestamps
- vendor-specific
  - opaque
  - shared

- Specialized
  - FORMAT OSD
  - APPEND – write w/o offset
  - CREATE & WRITE – save msg
  - FLUSH – force to media
  - FLUSH OSD – devicewide
  - LIST – recovery of objects

- Security
  - Authorization – each request
  - Integrity – for args & data
  - SET KEY
  - SET MASTER KEY

shared secrets

- Groups
  - CREATE COLLECTION
  - REMOVE COLLECTION
  - LIST COLLECTION
  - FLUSH COLLECTION
- Management
  - CREATE PARTITION
  - REMOVE PARTITION
  - FLUSH PARTITION
  - PERFORM SCSI COMMAND
  - PERFORM TASK MGMT

Read – 0x8805

- What do commands look like?
  - Follow SCSI CDB ... using extended CDB (up to 500 Bytes)
  - First 7 bytes of CDB have special code to specify extended CDB

Table 58 — READ command

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			SERVICE ACTION (8805h)				
9								(LSB)
10	OPTIONS BYTE							
11	Reserved		GET/SET CDBFMT		Reserved			
12	TIMESTAMPS CONTROL							
16	(MSB)			PARTITION_ID **64 bits**				
23								(LSB)
24	(MSB)			USER_OBJECT_ID **64 bits**				
31								(LSB)
32				Reserved				
35				**byte addressable**				
36	(MSB)			LENGTH				
43								(LSB)
44	(MSB)			STARTING BYTE ADDRESS				
51								(LSB)

List – 0x8803

- List command allows retrieval of all object IDs w/in a partition
  - Necessary for recovery

Table 49 — LIST command

Bit / Byte	7	6	5	4	3	2	1	0
8	(MSB)			SERVICE ACTION (8803h)				
9								(LSB)
10	Reserved							
11	Reserved		GET/SET CDBFMT			SORT ORDER		
12	TIMESTAMPS CONTROL							
16	(MSB)			PARTITION_ID				
23								
24	Reserved							
31								
32	(MSB)			LIST IDENTIFIER				
35								
36	(MSB)			ALLOCATION LENGTH				
43								
44	(MSB)			INITIAL OBJECT_ID				
51								(LSB)

only one option – ascending object id

continuation across LISTs

buffer size available

Commands Access Attr's

- To minimize number of commands (messages), almost every command can read and write the objects attributes
  - Read (object=0x1234, offset=0, length=100bytes, readAttr=physicalSize)

Table 33 — Page oriented get and set attributes CDB parameters format

Bit Byte	7	6	5	4	3	2	1	0
51	⋮ Other CDB fields							
52	(MSB)	GET ATTRIBUTES PAGE						
55								(LSB)
56	GET ATTRIBUTES ALLOCATION LENGTH							
59								(LSB)
60	RETRIEVED ATTRIBUTES OFFSET							
63								
64	SET ATTRIBUTES PAGE							
67								
68	(MSB)	SET ATTRIBUTE NUMBER						
71								(LSB)
72	SET ATTRIBUTE LENGTH							
75								(LSB)
76	SET ATTRIBUTES OFFSET							
79								
80	⋮ Other CDB fields							

which attrib

how much buffer host has available

which attrib

Num of attribs bytes being sent

Object attributes

Table 76 — User Object Information attributes page contents

Attribute Number	Length (bytes)	Attribute	May Be Set	OSD Logical Unit Provided
0h	40	Page identification	No	Yes
1h	8	Partition_ID	No	Yes
2h	8	User_Object_ID	No	Yes
3h to 8h		Reserved	No	
9h	variable	Username	Yes	No
Ah to 80h		Reserved	No	
81h	8	Used capacity	No	Yes
82h	8	User object logical length	Yes	Yes
83h to FFFF FFFEh		Reserved	No	

size

length

OSD Security Goals

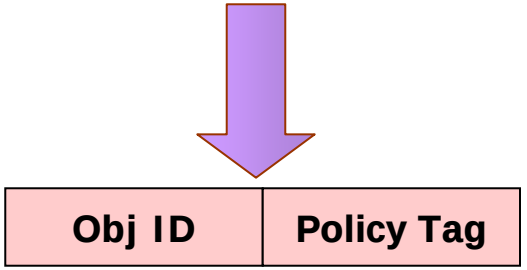
- Increased protection/security
  - At level of objects rather than LU
  - Hosts do not access metadata directly
- Allow non-trusted clients to sit on SAN
- Allow shared access to storage without giving clients access to all data on volume
- Division of responsibility between policy decisions and enforcement
  - Security manager implements a policy decision
  - Storage device enforces the decision

Concepts

- **capability:** The fields in a CDB that specify what command functions the command may request and on what objects. It is included in every request to an OSD
- **integrity check value:** A value computed using a security algorithm (e.g., HMAC-SHA1), a secret key and an array of bytes (i.e., signature)
- **credential:** A capability plus a signature generated by the security manager. This is sent to an application client in order to grant defined access to an OSD
- **capability key:** The signature part of the credential. This is used by the client to sign each OSD command
- **security manager:** The component of an OSD configuration that manages secret keys and prepares secure credentials containing capabilities thus granting application clients specified access to a specified object

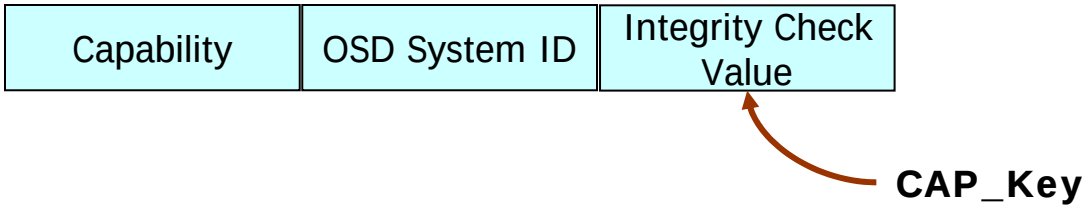
Capability Structure

Key version	Alg.	Sec. Method	Expiry	Audit	Discriminator	Obj Creation time	Obj desc	Permissions



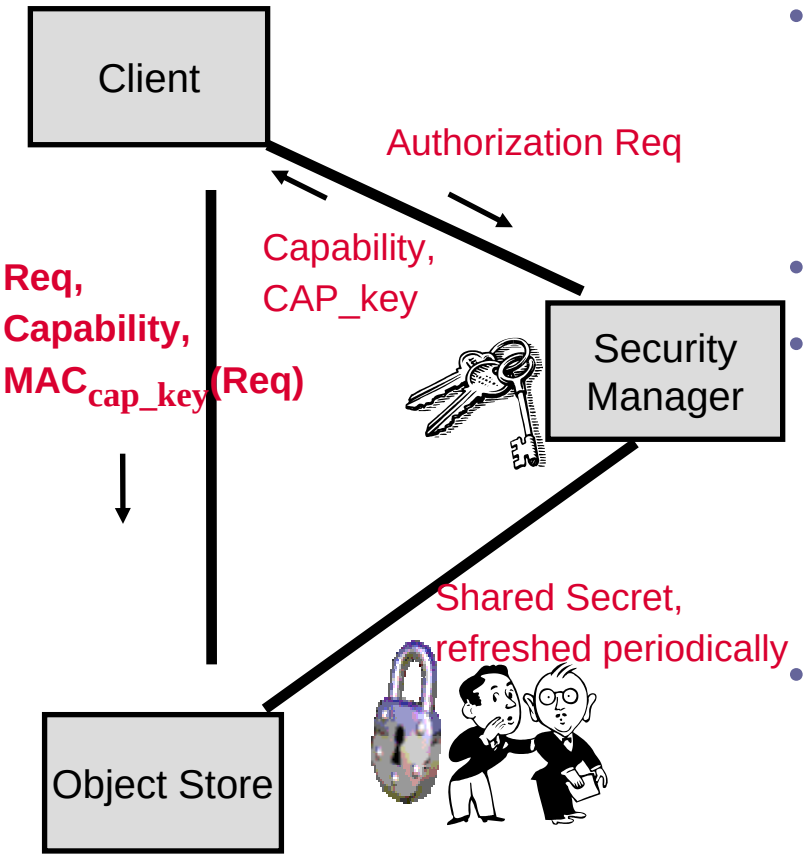
- Key Version – identifies secret key to be used for integrity check
- Algorithm – algorithm used for integrity check
- Security Method
  - Verified that it is allowed
- Expiry – expires a credential
  - Allowing different lifetimes for credentials
- Audit
  - Allows Manager to associate the capability with a specific client
- Capability Discriminator – ensures all credentials are unique
- Creation time – creation time of the object
  - Distinguished between objects with the same Object ID
- Permission Bit Mask – operations the client is entitled to perform
  - Multiple operations can be set
- Object Descriptor – Partition ID, Object ID and Policy tag
  - Policy Tag
    - Version Tag - Used to invalidate all credentials for an object
    - Fence – allows OSD to prevent access to an object

Credential Structure



- Secret Key is the key shared between the Manager and Client
- A **Credential** is derived from the capability arguments.
  - CAP_Key = MAC $_{\text{secret key}}$ (CAP_Args, OSD System ID)

Basic Security Model



- All operations are secured by a capability
  - –Is the command allowed to access the specified object ?
  - –Is the command valid?
- Manager and OSD are trusted
- Security achieved by cooperation of:
  - –Manager – authenticates/authorizes clients and generates credentials
  - –OSD -- validates credential that a client presents
- Credential is cryptographically hardened
  - –OSD and Manager share a secret

Reprise: Strengths of Object Storage

- Variable length data
  - Decisions on data layout can be optimized based on object size
- Layout metadata encapsulated at device
  - Layout metadata managed at Object Storage Device
  - Offloads up to 90% of metadata management workload from central controller
- Extensible attributes
  - E.g. size, timestamps, ACLs, etc.
- Access Control decisions are signed, cached at clients, enforced at device
  - Clients can be untrusted (bugs & attacks expose only authorized object data)
  - Cache decisions & maps replaced transparently (dynamic remapping)
- Command set works with SCSI architecture model (SAM)
  - Encourages cost-effective implementation by storage device vendors

Object Storage Devices

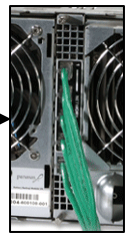
Expect wide variety of Object Storage Devices

- Disk array subsystem
- Ie. LLNL with Lustre

- "Smart" disk for objects
- 2 SATA disks – 240/500 GB

- Prototype Seagate OSD
- Highly integrated, single disk

- Orchestrates system activity
- Balances objects across OSDs



16-Port GE Switch Blade

- 4 Gbps per shelf to cluster

Stores up to 5 TBs per shelf

Storage Architecture – Interfaces and Dataflow

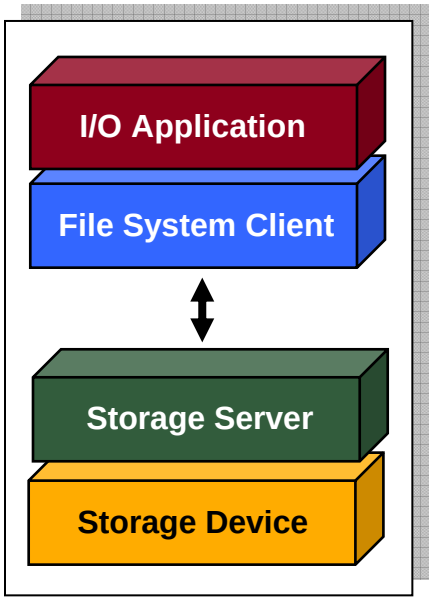
- Storage Interface
  - Block-based architecture: fast but private
    - Traditional SCSI and FC approaches
    - Expensive fabric, difficult to share between hosts
  - File-based architecture: sharable, but bottlenecked performance
    - NAS storage (NFS, CIFS, AFS and DFS)
  - Object-based: fast, shared, secure device
    - Storage device exports objects (collection of related data) instead of blocks
- Dataflow
  - In-band
    - Data flows through server or cluster of servers
  - Out-of-band
    - Data flows directly from client to storage
    - Metadata is managed off the datapath

Ideally, Storage Would Provide

- Direct and parallel storage access between storage and clients
  - Get rid of central bandwidth bottlenecks
- Commodity technology with integrated functionality
  - Keep costs low
- Shared-nothing clusters of data & metadata
  - Enable scaling of metadata, management, and fault-tolerance
- Allowing you to realize
  - Performance scales as capacity increases
    - Bandwidth proportional to number of storage nodes
  - Load-balance storage workload across system
  - Maintain level of reliability as system scales

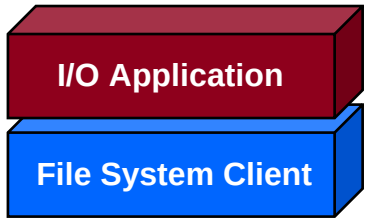
OSD System Architecture

Object-Based Architecture
Distributes **the System**

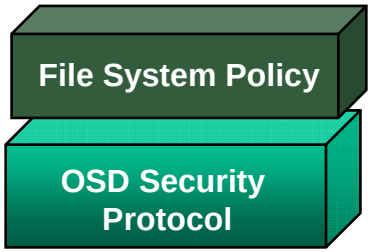


Clients

DirectorBlade
(aka Manager)

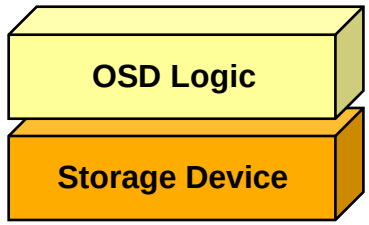


Control



Data

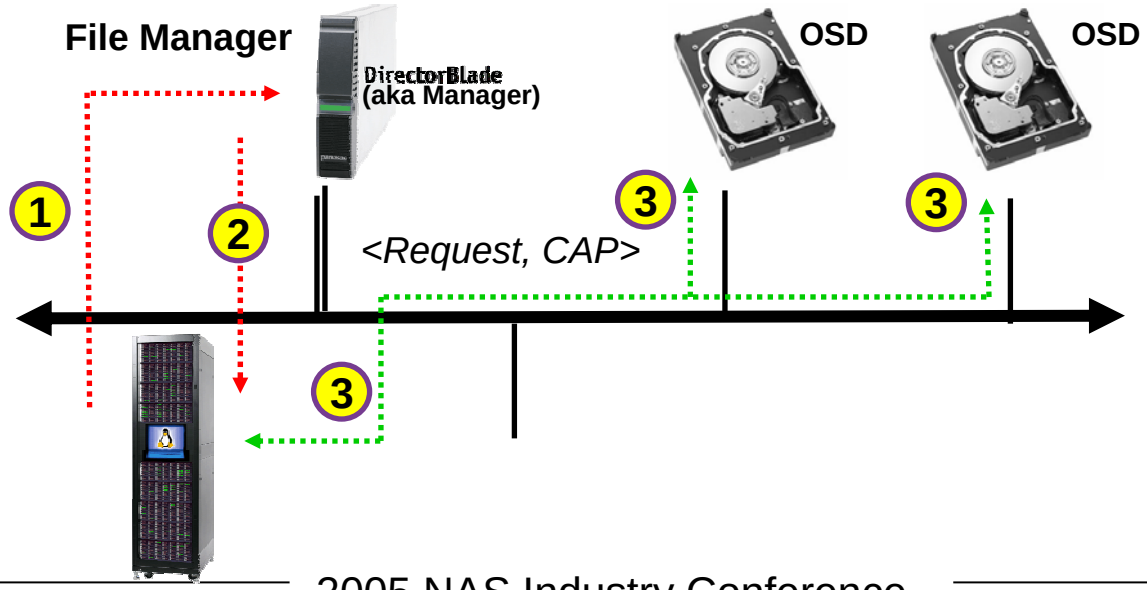
Configuration



File manager is not in the data path

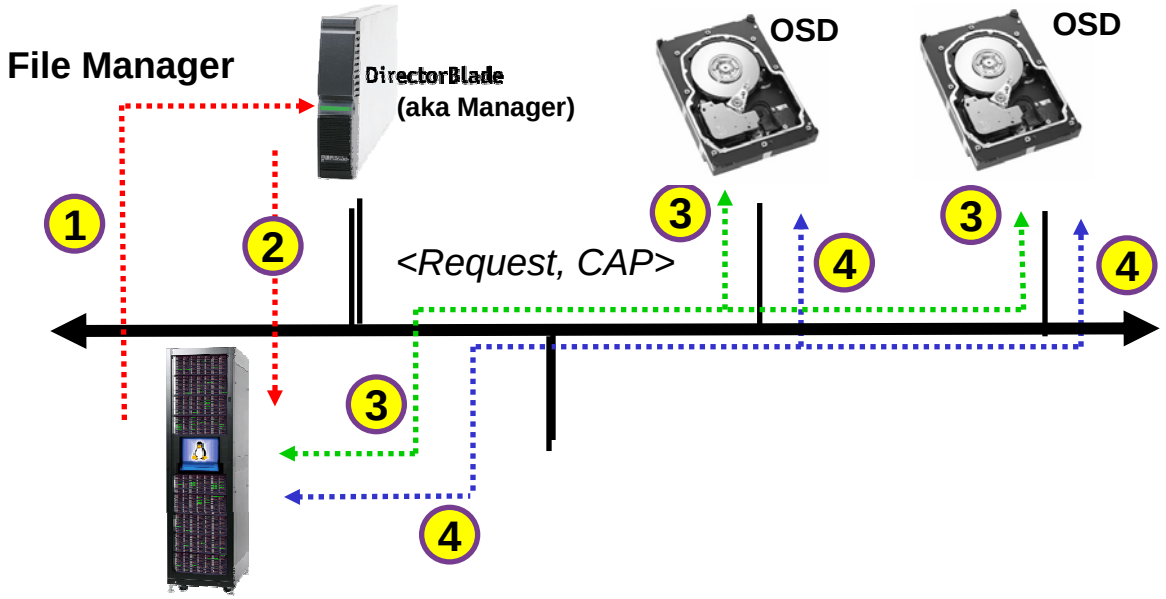
Direct Transfer Architecture

- (1) Client initially contacts File Manager
  - File Manager checks file system policy
- (2a) File Manager returns list of objects <drive a, obj 23, drive b, obj 43>
- (2b) File Manager also returns a security CAPABILITY
  - Capability is security token used by client to access OSD - CAPABILITY authorizes client requests
- (3) Client sends requests (read, write) directly to OSD along with CAPABILITY
  - OSD checks CAPABILITY & OSD performs request



Direct Transfer Architecture

- (4) Direct-data transfer between client and OSD
- Examples:
  - CMU NASD, Panasas, Lustre, Seagate/IBM T10 OSD

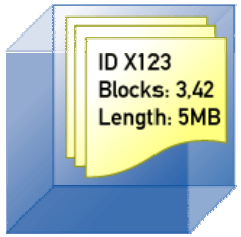


Aggregating Objects

- By striping a file across multiple OSDs, the application(s) can realize the aggregate bandwidth of all OSDs in the system
  - Bandwidth
  - Load balancing
  - Reliability – RAID

Objects as Building Blocks

Object



Comprised of:

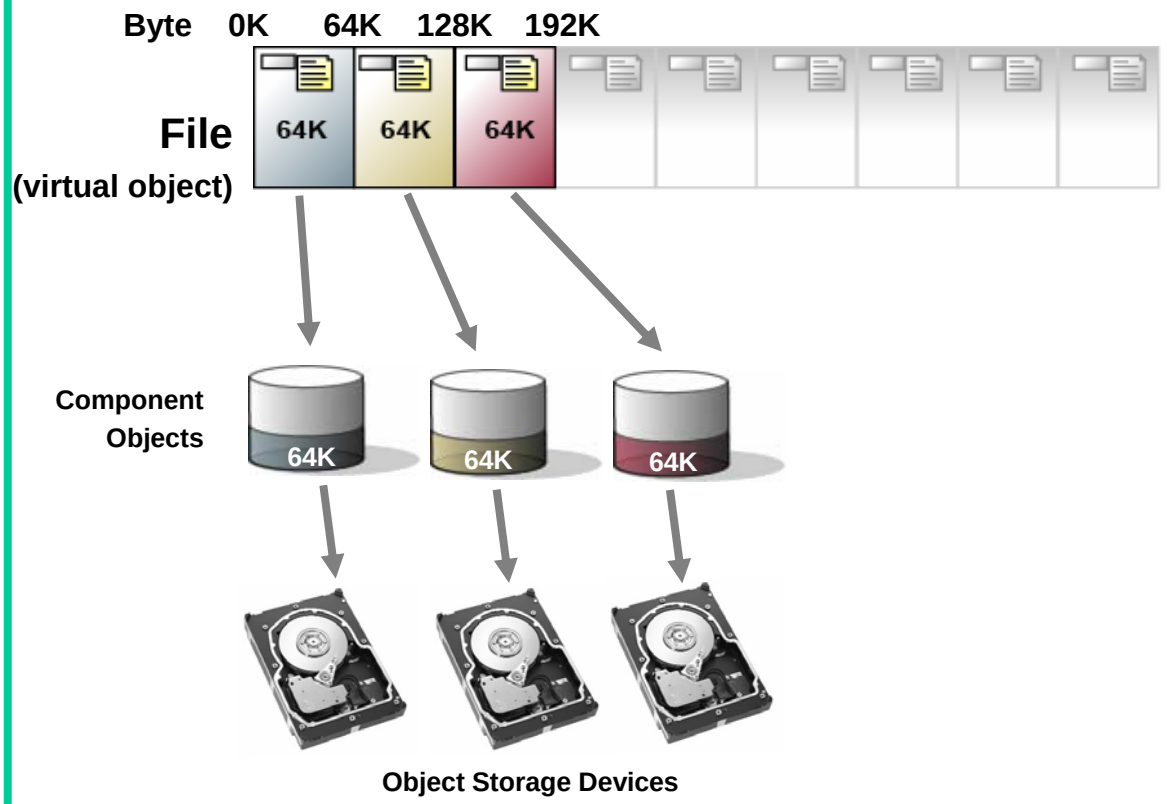
- User Data
- Attributes
- Layout

Interface:

- ID <dev,grp,obj>
- Read/Write
- Create/Delete
- Getattr/Setattr
- Capability-based

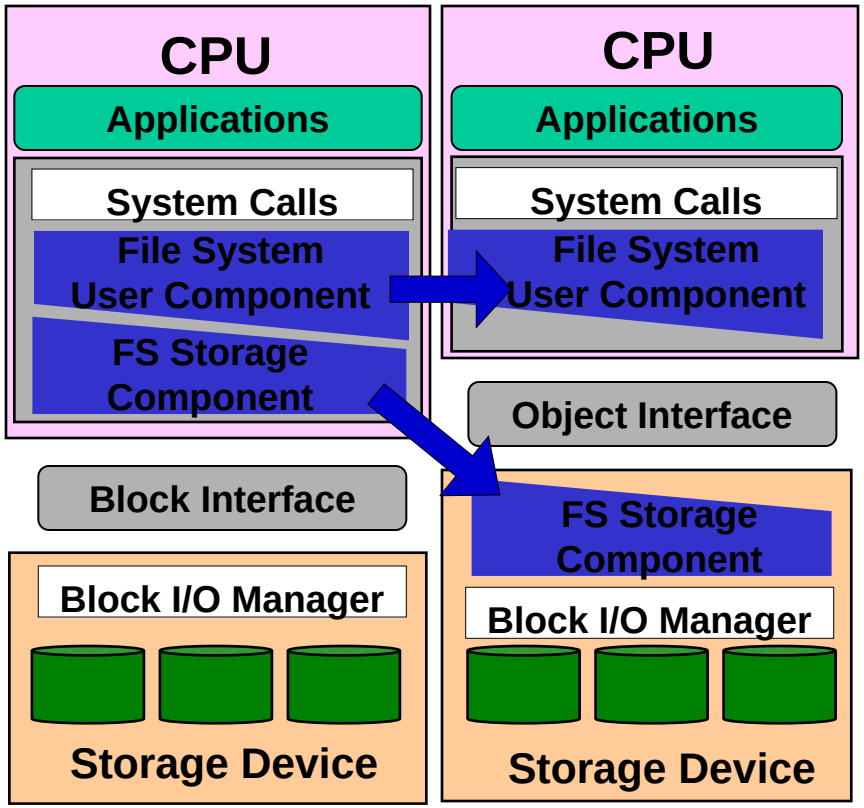
File Component:

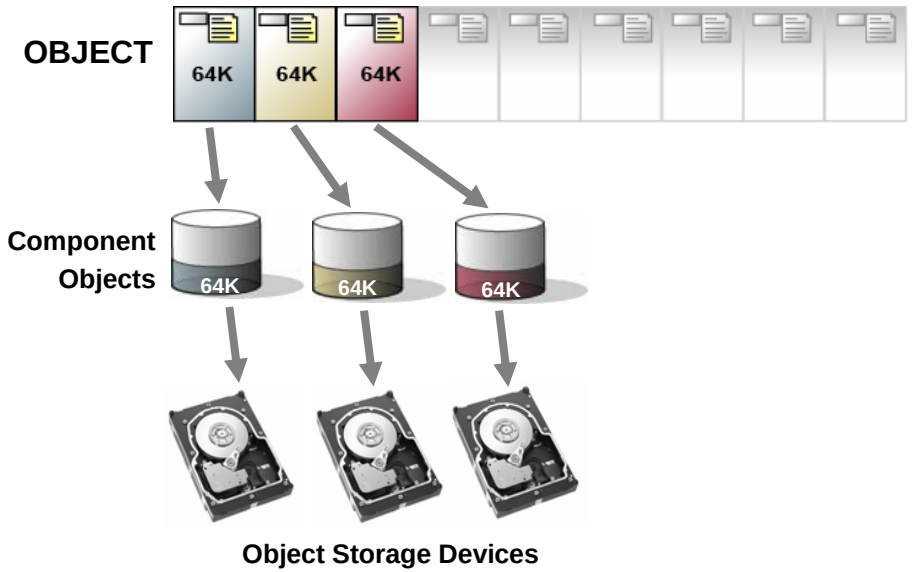
- Stripe files across storage nodes
- 1 File == 1 Virtual Object striped across 1 or more physical objects



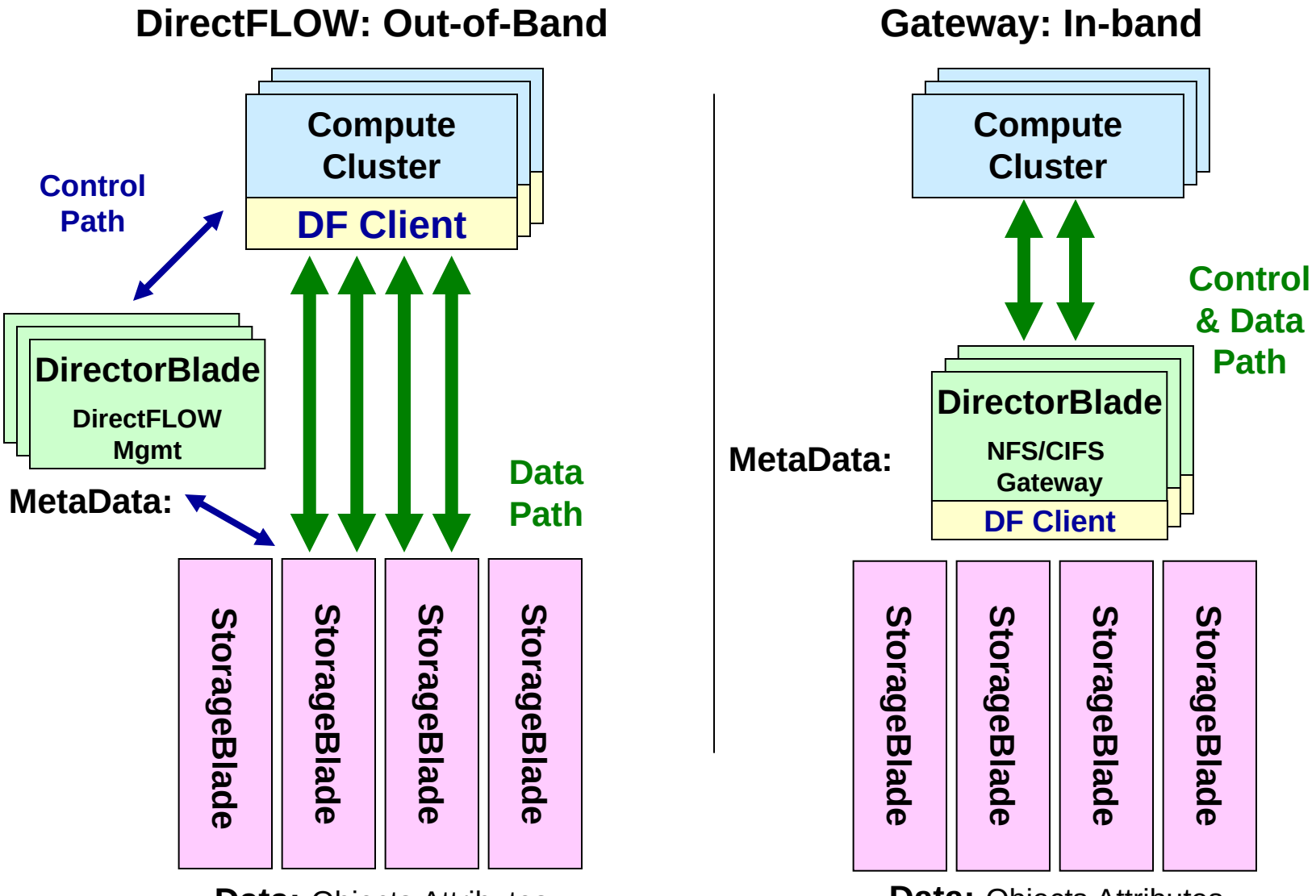
Aggregation: Virtual Objects are Scalable

- Clients accessing file get 3 OSD's worth of bandwidth or IOPs





Panasas ActiveScale System



DirectFLOW Client

- DirectFLOW client is kernel loadable FS module
  - Implements standard OS/filesystem interface (POSIX)
  - Some extensions for performance tuning (layout control, concurrent write)
  - Uses native Panasas network protocols (RPC and iSCSI) over TCP
- Mount options
  - Local (NFS-style) mount anywhere in client, or
  - Global (AFS-style) mount at “/panfs” on all clients
- Caches data, directories, attributes, capabilities
- Responds to callbacks for cache consistency
- Does RAID I/O directly to StorageBlades
- Optimized for Linux

Opening Files & I/O

- Mount
  - At mount time, client learns the object ID of the root directory
  - Client asks DirectorBlade for capability to read directory and a callback to cache directory data
- Pathname resolution
  - Client iterates, continually checks cache to see if it can read directories, or directory data
  - Client gets object ID for the file it wants and requests map and capability
  - Director checks ACL on the object, returns capability and RAID map
- I/O
  - Client does parallel I/O to all the StorageBlades that store component objects
  - StorageBlades verify capability to enforce access control

Creating Files & Metadata

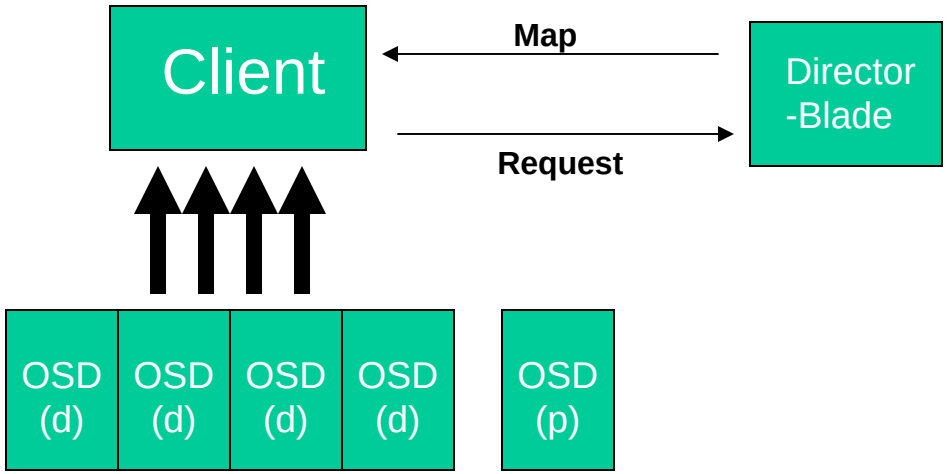
- Creating a file
  - Client asks metadata manager to create a file
  - Manager creates a pair of component objects (for RAID1/5 file)
  - Manager updates file system directory, which is another object pair
  - Manager returns map and capabilities to client
  - Create file in 2.4 msec, delete a file in 1.9 msec, w/ distributed fault tolerance
- Growing a file
  - Small files (<64K) are mirrored on the first two component objects (RAID1)
  - Large files use additional component objects, up to a full stripe worth (RAID5)
  - Storage manager issues capabilities for data ranges, creating additional components and updating the file's map if necessary

Attributes

- Most PanFS metadata stored in OSD-maintained attributes on objects
- Object attributes distinct from object data
  - Timestamps
  - File/object sizes
  - Directory linkage information
  - Internal fields used for recovery
  - Etc.
- Attribute fetch and manipulation very important to PanFS performance
- Aggressively cached by FM, SM, and client

Client Read Operation

- Steps in a client read operation
  - Client determines file ID and responsible director from the directory entry
  - Client requests permission to read from the director (read cap)
  - Director returns permission + file map identifying components
  - Client determines byte ranges within components and initiates a network transfer for each, all in parallel, ignoring the parity blocks
  - Client can re-use permission and map until told otherwise by director

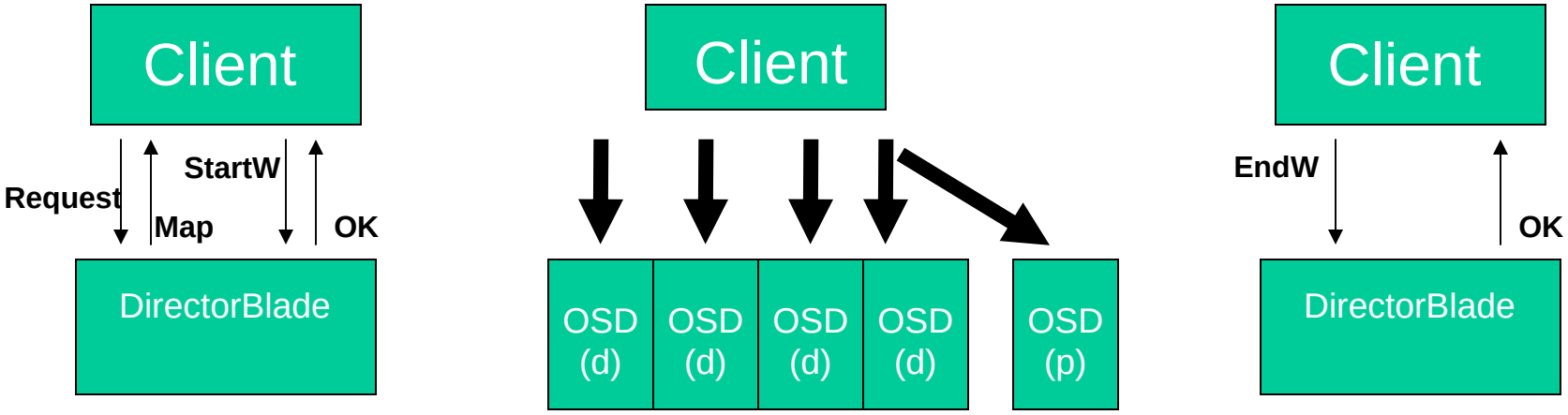


Client Read Operation

- Different files can use different sets of OSDs
- Bandwidth is achieved via parallel transfers from multiple OSDs
  - More OSDs ⇒ higher parallelism ⇒ higher bandwidth
- No serialization at a server machine, no bottleneck at a RAID controller
- Many OSDs and ample switch infrastructure together allow many cluster nodes to simultaneously achieve high bandwidth from storage
- Client need not keep track of sector mapping: it requests a single byte range from a single named object
- OSD is free to optimize: cache, read-ahead, write-behind, relocate

Client Write Operation

- Director must assure that concurrent writes do not corrupt parity, and that clients see coherent data in storage
- Therefore follow “start-write / end-write” policy
- Client accumulates “enough” dirty data to get high bandwidth transfer
- Client requests permission to write, writes all dirty data and updates parity, releases write permission back to the director
- Distinct from reads in that permission to read is long-lived



Caching and Cache Consistency

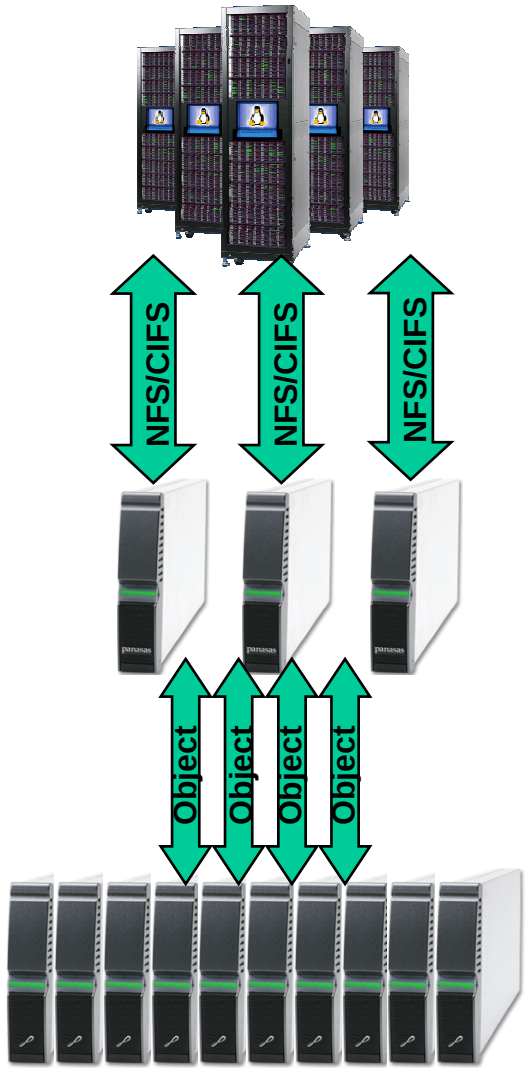
- Caching information on client avoids interaction between blades
  - Clients cache file data, directory data, attributes, and capabilities
  - Clients cache both clean (read) and dirty (written) data
- DirectorBlades keep “callbacks”
  - Client indicates interest in file by registering callback
  - Director promises to notify the client if cached data or attributes become invalid, or other client wants to access file, via “callback break” message
  - Callback type indicates sharing state, similar to CIFS opportunistic lock (oplock)
    - Exclusive callback = no other clients using file; we can cache reads and writes
    - Read-only shared callback = other clients reading, but no writers; we can cache reads
    - Read/write shared callback = other clients reading or writing; don’t cache anything
    - Concurrent write callback = special sharing mode for cooperating apps
  - Callbacks are leased & expire after ~8 hrs unless renewed

Gateways

- Gateway is a NFS/CIFS server on a PanFS client
  - Same PanFS client as DirectFLOW client
  - FreeBSD NFS server, Samba CIFS server
- All clients, including Gateways, have coherent view
  - All gateways provide identical view of PanFS, coherent with DirectFLOW
  - Cache coherency provided by PanFS client
  - Each gateway accesses all data in system—one NFS mount or CIFS share for whole Panasas cluster

NFS/CIFS Access

- Platform versatility
- Legacy systems does not need special software
  - NFS / CIFS are built in modern operating systems
- DirectorBlades translate file requests
  - From NFS / CIFS to object and exports data to realm
  - Maintaining permissions and other key attributes
- Scalable performance via additional DirectorBlades
- Same volumes available
  - Via DirectFLOW, NFS and CIFS simultaneously

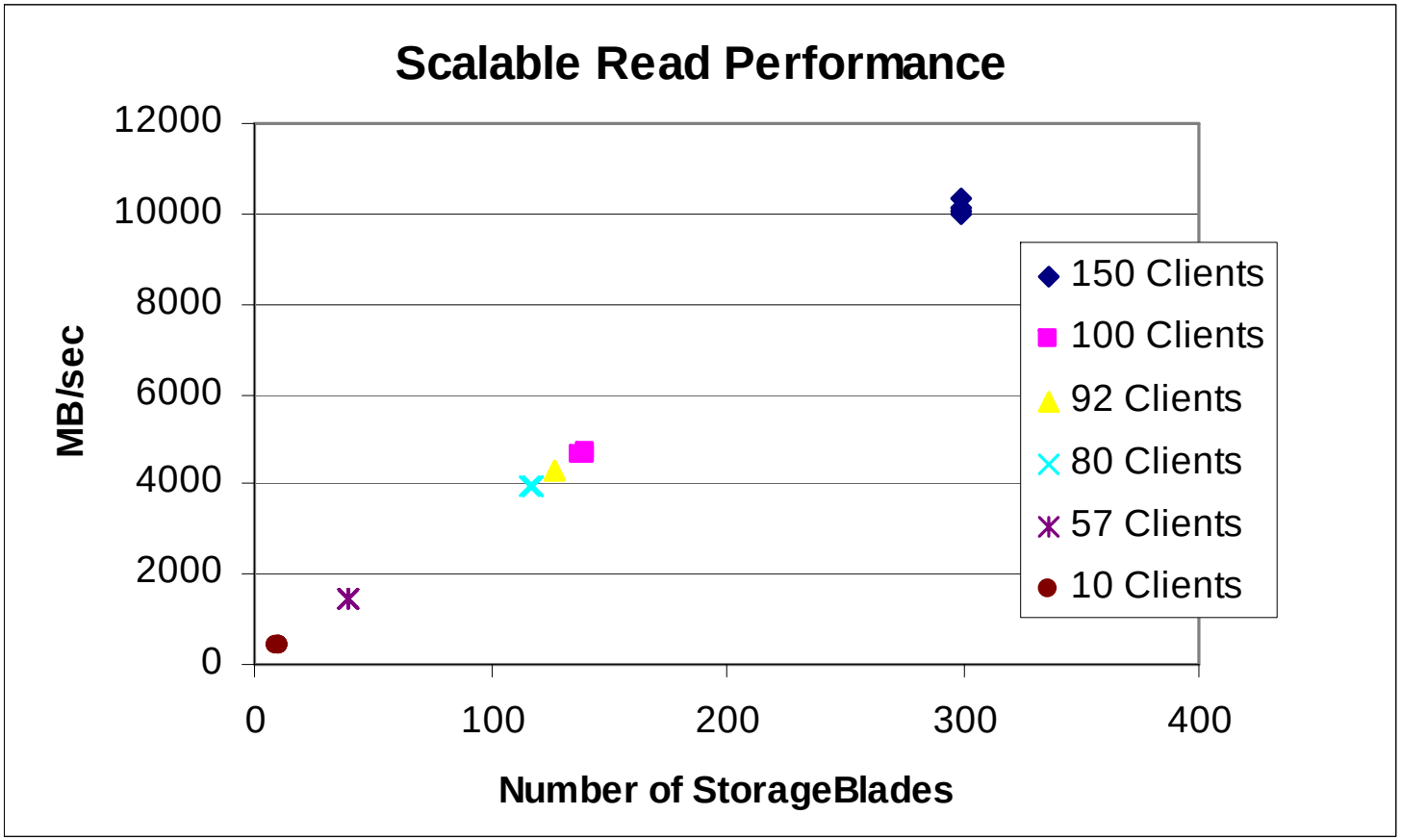


Measurements

- Scalable System Bandwidth
- Scaling clients
- N-to-1 Read / Write Performance
- Random I/O
- Reconstruction

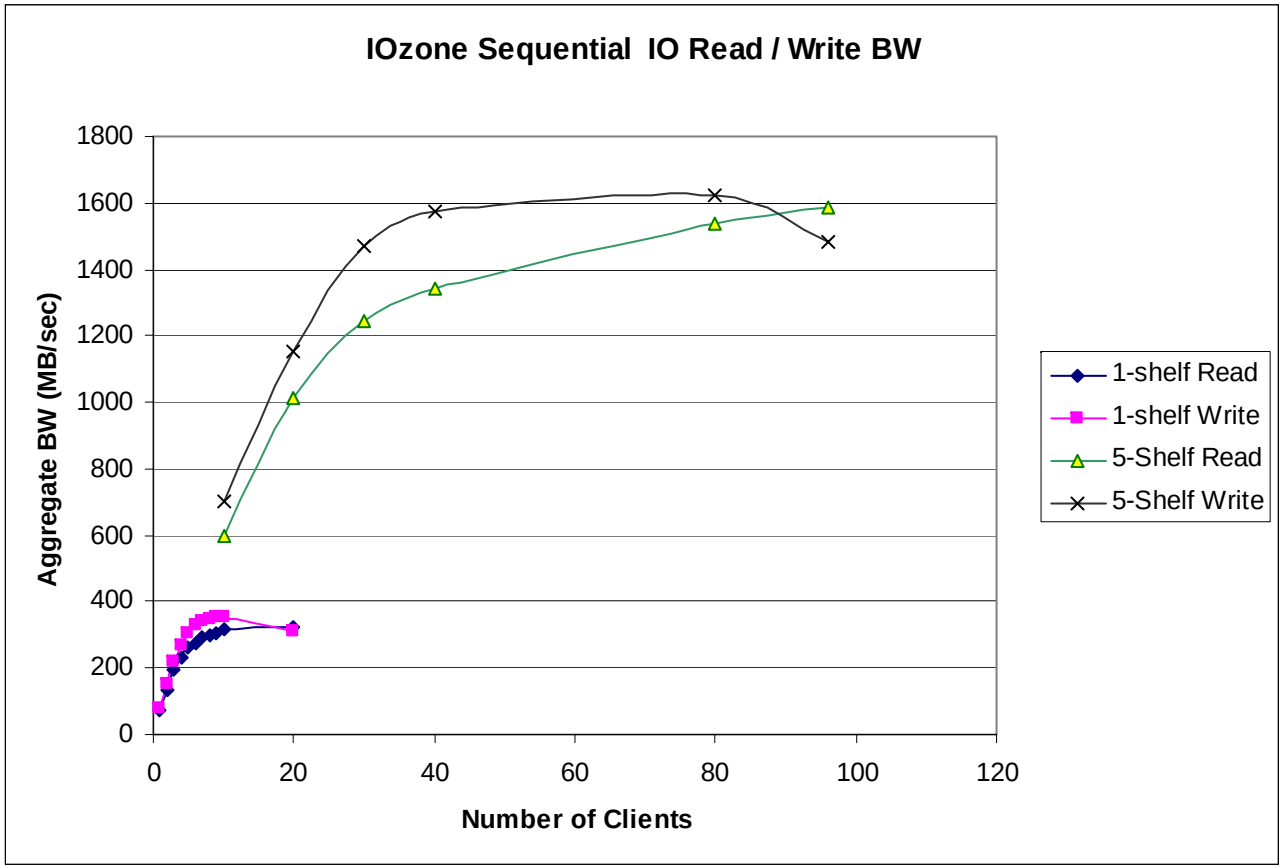
More system scaling

- Scaling up to 30 shelves (N-to-N) over Direct Flow



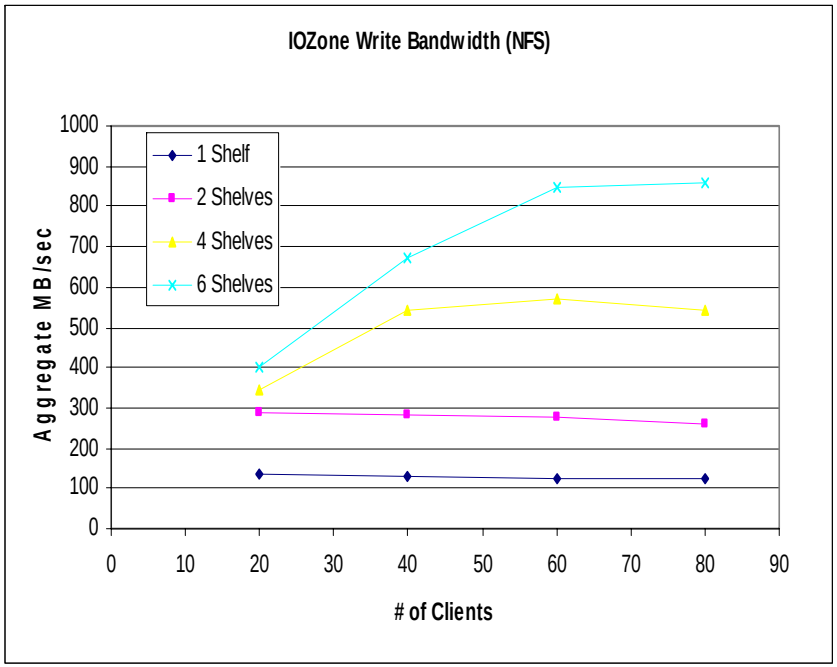
Scaling Clients

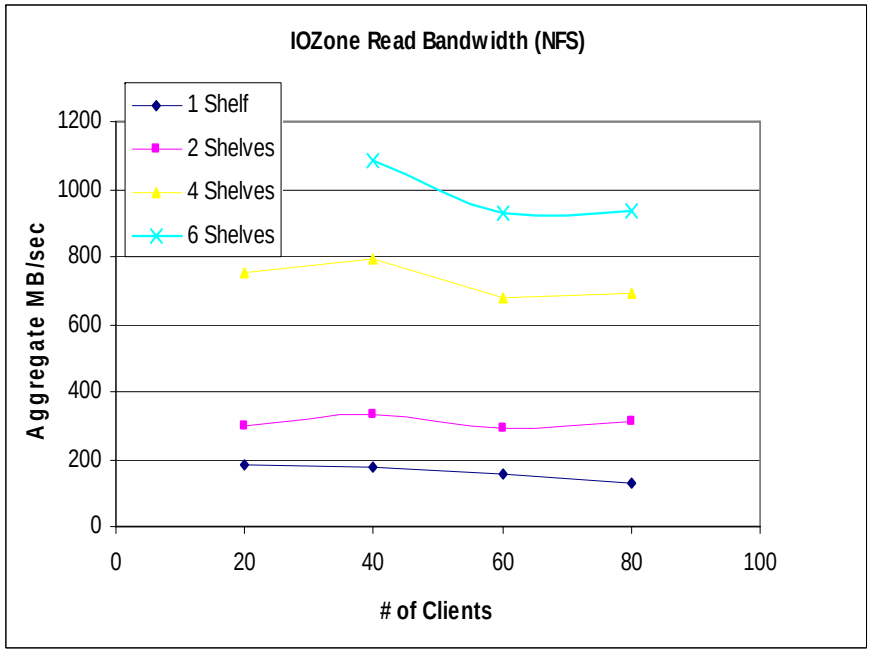
- Fixed system size, grow the number of clients (N-to-N over Direct Flow)



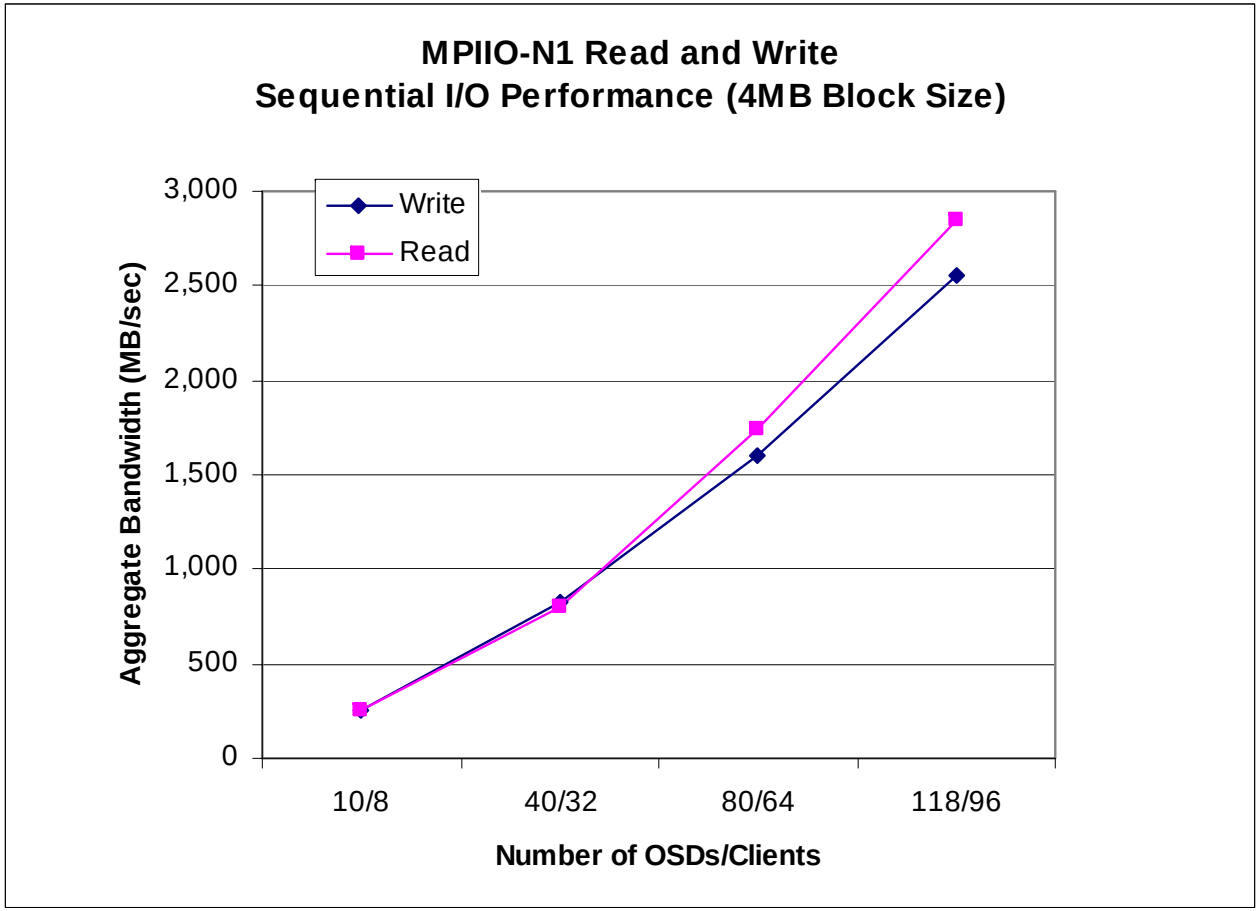
Scaling NFS

- IOZone over Panasas NFS (each shelf is 3 Directors + 8 Storage Blades)

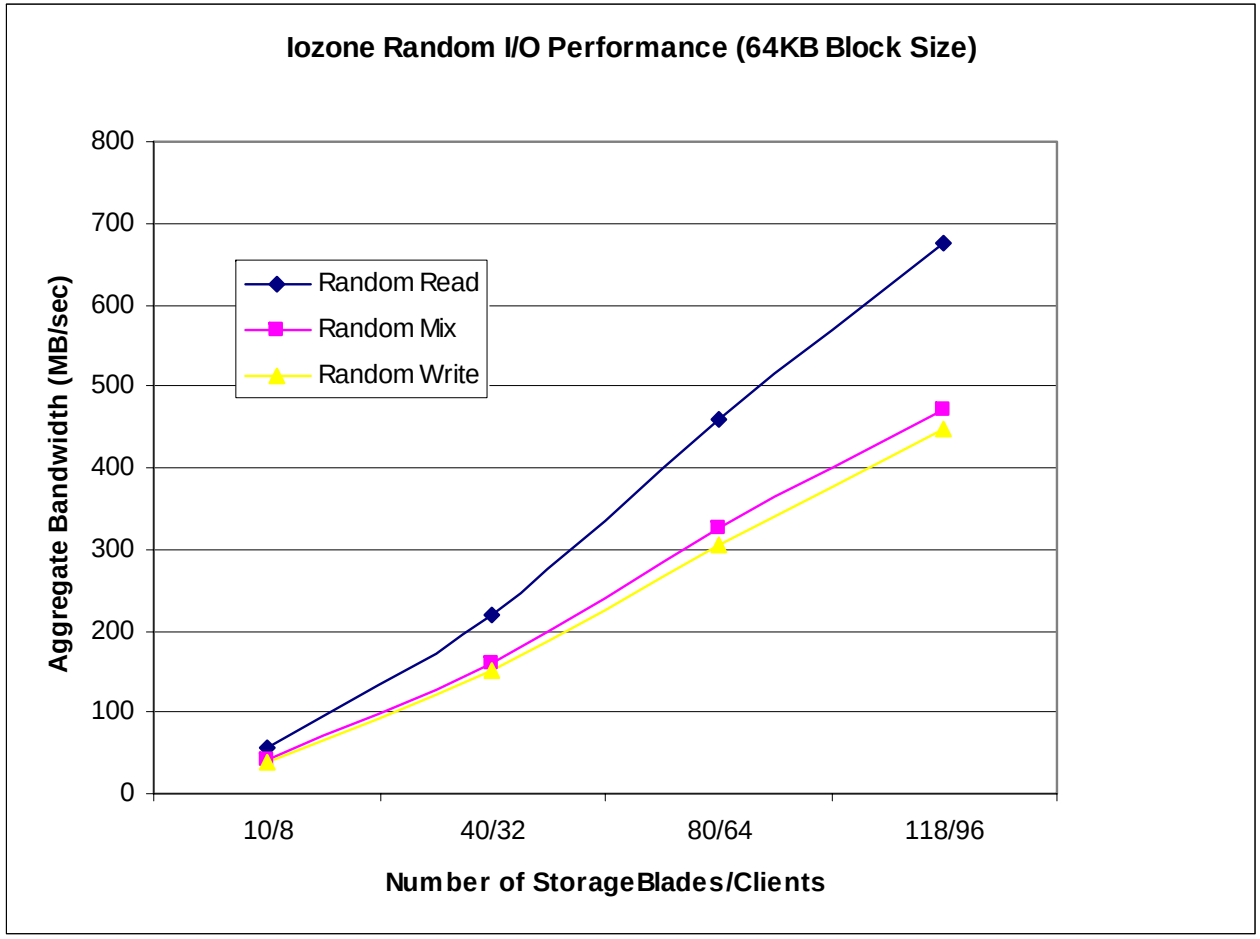




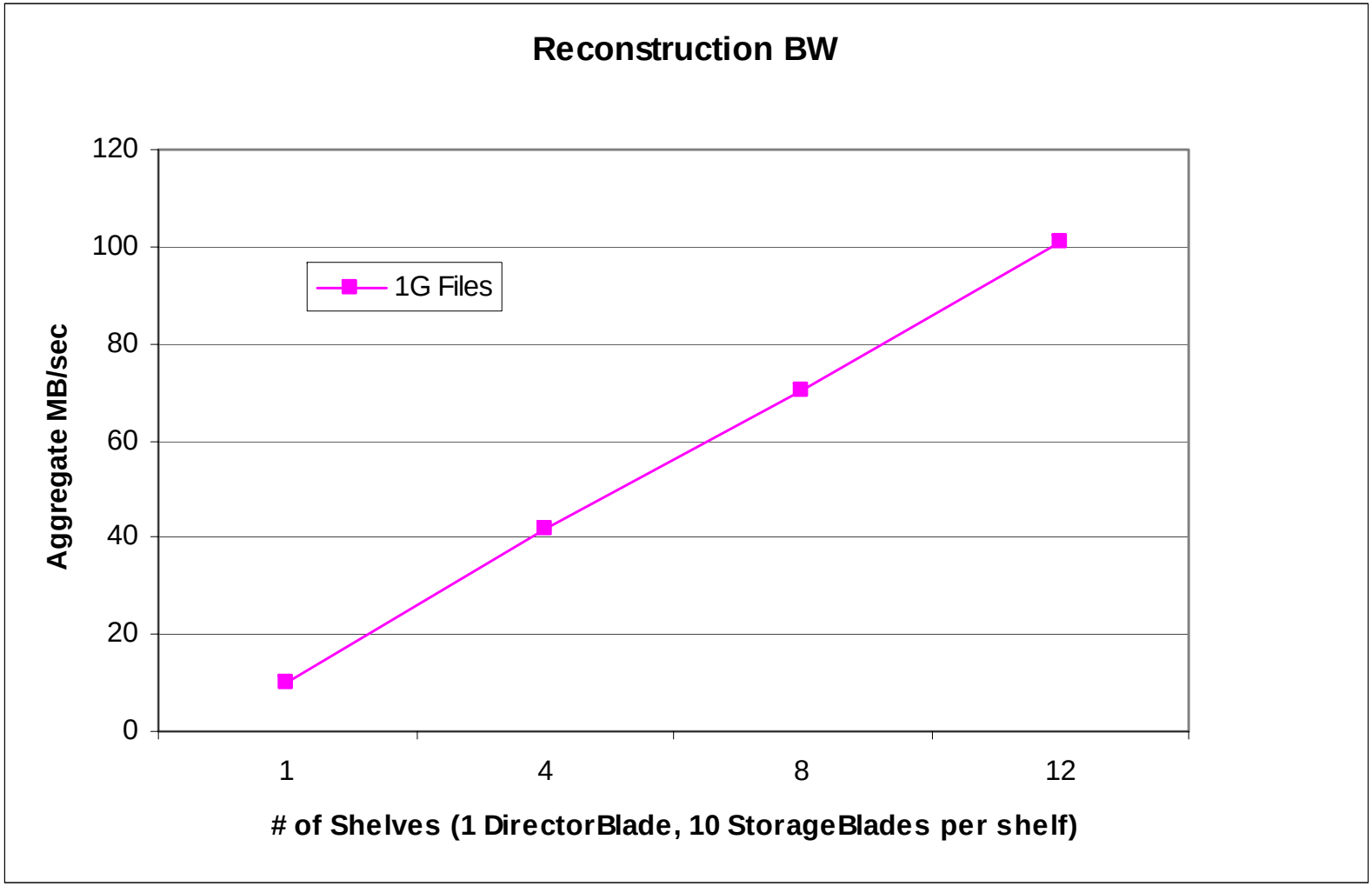
Concurrent Read/Write Performance



Random IO


Iozone Random I/O Performance (64KB Block Size)
Aggregate Bandwidth (MB/sec)
800
700
600
500
400
300
200
100
0
Random Read
Random Mix
Random Write
10/8
40/32
80/64
118/96
Number of StorageBlades/Clients

Reconstruction Performance

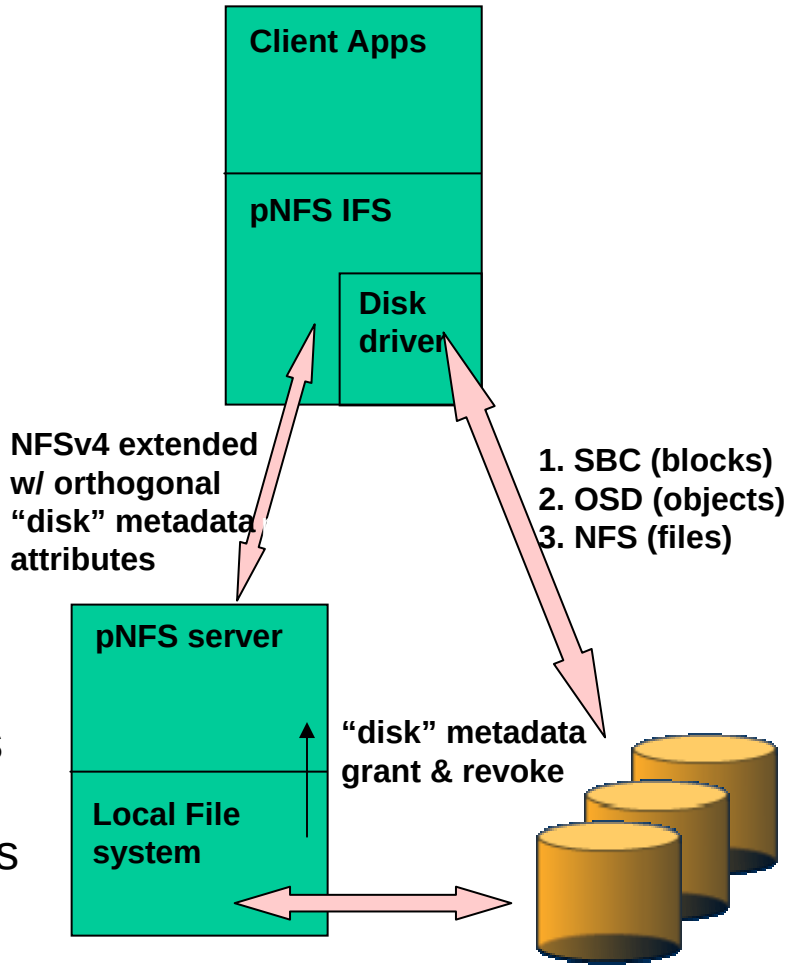


OSD – Real Technology

- SNW - World's first demonstration of ANSI OSD standard
- 3 company collaboration
  - Emulex - FC HBA w/ bidirectional transfers, long CDBs
  - IBM - Full host software stack & iSCSI OSD subsystem
  - Seagate – OSD in a standard FC HDD
- Demonstration: video files written into file system and played
  - Files were written to explicit devices by OSD system
  - All in a common file system
  - File system did not know or care where the files were located

File Systems Standards: Parallel NFS

- IETF NFSv4 initiative
  - U. Michigan, NetApp, Sun, EMC, IBM, Panasas, ….
  - Enable parallel transfer in NFS
- Extension to NFSv4 for parallel I/O
  - Parallel NFS requests routing / file virtualization
- Provides asymmetric architecture (metadata and data servers) for NFS
- Framework supports
  - Blocks: SBC/FCP/FC or SBC/iSCSI for flies built on blocks
  - Objects: OSD/iSCSI/TCP/IP/GE for files built on objects
  - Files: NFS/ONCRPC/TCP/IP/GE for files built on subfiles
- Inode-level encapsulation in server & client code



pNFS

- Active in the NFSv4 working group
  - Panasas, Sun, NetApp, EMC, IBM, others …
- IETF pNFS Documents:
  - draft-gibson-pnfs-problem-statement-01.txt
  - draft-gibson-pnfs-reqs-00.txt
  - draft-welch-pnfs-ops-00.txt

References

- http://www.pdl.cmu.edu/NASD
- http://www.t10.org/scsi-3.htm
- http://www.t10.org/ftp/t10/drafts/osd
- http://www.intel.com/labs/storage/osd
- http://www.panasas.com