

Standards Update

POSIX IO
pNFS
OSDv2

Brent Welch, Panasas Inc.
HECIWG-FSIO 2007



APIs for HPC IO

- POSIX IO APIs (open, close, read, write, stat) have semantics that can make it hard to achieve high performance when large clusters of machines access shared storage.
- A working group (see next slide) of HPC users has drafted API additions for POSIX that will provide standard ways to achieve higher performance.
 - HECEWG – High End Computing Extensions Working Group
- Primary approach is either to relax semantics that can be expensive, or to provide more information to inform the storage system about access patterns.

HPC POSIX Enhancement Areas

- Metadata
 - optional attributes, bulk attributes
 - statlite(), readdirplus(), readdirlite()
- Coherence
 - last writer wins and other such things can be optional
 - lazyio_propagate(), lazyio_synchronize()
- Shared file descriptors
 - file opens for cooperating groups of processes
 - openg(), openfh()
- Ordering
 - stream of bytes idea needs to move towards distributed vectors of units
 - readx(), writex()

Contributors

- <http://www.opengroup.org/platform/hecewg/>
- Lee Ward - Sandia National Lab
- Bill Lowe, Tyce McLarty – Lawrence Livermore National Lab
- Gary Grider, James Nunez – Los Alamos National Lab
- Rob Ross, Rajeev Thakur, William Gropp - Argonne National Lab
- Tom Ruwart- U of Minnesota/IO Performance
- Roger Haskin – IBM
- Brent Welch, Marc Unangst - Panasas
- Garth Gibson- CMU/Panasas
- Alok Choudhary – Northwestern U
- *Many Others...*



statlite, fstatlite, lstatlite – Optional Attributes

- **Syntax**

- `int statlite(const char *file_name, struct statlite *buf);`
`int fstatlite(int filedes, struct statlite *buf);`
`int lstatlite(const char *file_name, struct statlite *buf);`

- **Description**

- This family of stat calls, the lite family, is provided to allow for file I/O performance not to be compromised by frequent use of stat information lookup. Some information can be expensive to obtain when a file is busy.
- They all return a *statlite* structure, which has all the normal fields from the stat family of calls but some of the fields (e.g., file size, modify time) are optionally not guaranteed to be correct.
- There is a *litemask* field that can be used to specify which of the optional fields you require to be completely correct values returned.

readdirplus & readdirlite

read dir and attributes

- **Syntax**

```
struct dirent_plus *readdirplus(DIR *dirp);
```

```
int readdirplus_r(DIR *dirp, struct dirent_plus *entry, struct  
    dirent_plus **result);
```

```
struct dirent_lite *readdirlite(DIR *dirp);
```

```
int readdirlite_r(DIR *dirp, struct dirent_lite *entry, struct dirent_lite  
    **result);
```

- **Description**

- **readdirplus(2)** and **readdirplus_r(2)** return a directory entry plus **lstat(2)** results (like the NFSv3 READDIRPLUS command)
- **readdirlite(2)** and **readdirlite_r(2)** return a directory entry plus **lstatlite(2)** results

O_LAZY - Lazy I/O data integrity

- Specify O_LAZY in *flags* argument to **open(2)**
- Requests lazy I/O data integrity
 - Allows network filesystem to relax data coherency requirements to improve performance for shared-write file
 - Writes may not be visible to other processes or clients until **lazyio_propagate(2)**, **fsync(2)**, or **close(2)** is called
 - Reads may come from local cache (ignoring changes to file on backing storage) until **lazyio_synchronize(2)** is called
 - Does not provide synchronization across processes or nodes – program must use external synchronization (e.g., pthreads, XSI message queues, MPI) to coordinate actions
- This is a hint only
 - if filesystem does not support lazy I/O integrity, does not have to do anything differently
 - {LAZY_ALIGNMENT} **pathconf()** setting indicates optimal granularity

openg – Map file name to portable file handle

- **Syntax**

- `int openg(char *path, int flags, fh_t *handle, int mode);`

- **Description**

- The *openg()* function opens a file named by path according to flags (e.g., O_RDWR). It returns an opaque file handle corresponding to a file descriptor. The intent is that the file handle can be transferred to cooperating processes and converted to a file descriptor with *openfh()*.
 - The lifetime of the file handle is implementation specific. For example, it may not be valid once all open file descriptors derived from the handle with *openfh()* have been closed.

openfh – portable file handle to IO channel



- **Syntax**

- `int openfh(fh_t *fh);`

- **Description**

- The file offset used to mark the current position within the file shall be set to the beginning of the file.
 - The file status flags and file access modes of the open file description shall be set according to those given in the accompanying *openg()*.
 - The largest value that can be represented correctly in an object of type **off_t** shall be established as the offset maximum in the open file description.

readx writex – memory vector to/from file vector

- **Syntax**

- `ssize_t readx(int fd, const struct iovec *iov, size_t iov_count, struct xtvec *xtv, size_t xtv_count);`
- `ssize_t writex(int fd, const struct iovec *iov, size_t iov_count, struct xtvec *xtv, size_t xtv_count);`

- **Description**

- Generalized file vector to memory vector transfer. Existing `readv()`, `writew()` specify a memory vector and do serial IO. The new `readx()`, `writex()` calls also read/write strided vectors to/from files.
- The `readx()` function reads `xtv_count` blocks described by `xtv` from the file associated with the file descriptor `fd` into the `iov_count` multiple buffers described by `iov`. The file offset is not changed.
- The `writex()` function writes at most `xtv_count` blocks described by `xtv` into the file associated with the file descriptor `fd` from the `iov_count` multiple buffers described by `iov`. The file offset is not changed.

Declined Features

- Layout control
 - API to tune data layout within the clustered data storage devices
- lockg
 - Group lock for cooperating nodes to fence a file from naive access
- statlite
 - Existing fstatat() Linux API may be morphed to do what we wanted, and the idea of “fuzzy” attribute values not widely appreciated



POSIX ACLs → New NFSv4 Semantics

- Legitimize NFSv4 ACLs in POSIX, allowing users to choose methodology and over time maybe POSIX ACLs will fade away.
 - Note that “POSIX ACLS” are really only a proposed part of the standard and not widely implemented or used
 - NFSv4 ACLs are aligned with the Windows ACL model, which is more widely used and more sensible
 - The two models differ in how ACLs are inherited, and in the rules for processing a long set of ACE (access control entries)
- draft-falkner-nfsv4-acls-00.txt is an Internet Draft from Sun that explains how they are exposing NFSv4 ACLs for Solaris 10.

POSIX HPC IO

- **statlite, fstatlite**
 - optional attributes
- **readdirplus, readdirlite**
 - expose NFS bulk attribute op
- **O_LAZY , lazyio_propogate
lazyio_synchronize,**
 - Hint to buffer cache management
- **openg, openfh**
 - expose file handles to applications
- **readx, writex**
 - memory vector to/from file vector
- <http://www.opengroup.org/platform/hecewg/>



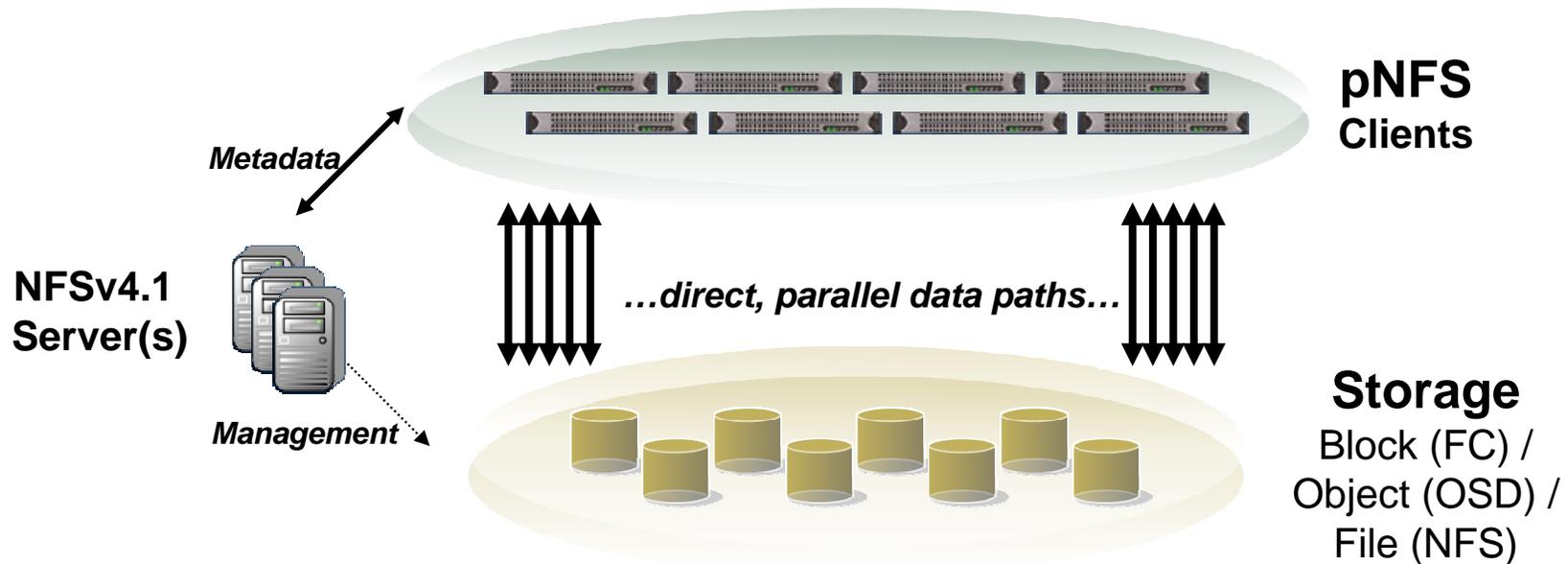
NFSv4 and pNFS

- NFS created in '80s to share data among engineering workstations
- NFSv3 widely deployed
- NFSv4 eight years in the making, lots of new stuff
 - Integrated Kerberos (or PKI) user authentication
 - Integrated file locking
 - ACLs (hybrid of Windows and POSIX models)
- NFSv4.1 adds even more
 - Details learned from early NFSv4.0 experience
 - pNFS for parallel I/O
 - Directory delegations for efficiency
 - Sessions for better at-most-once semantics



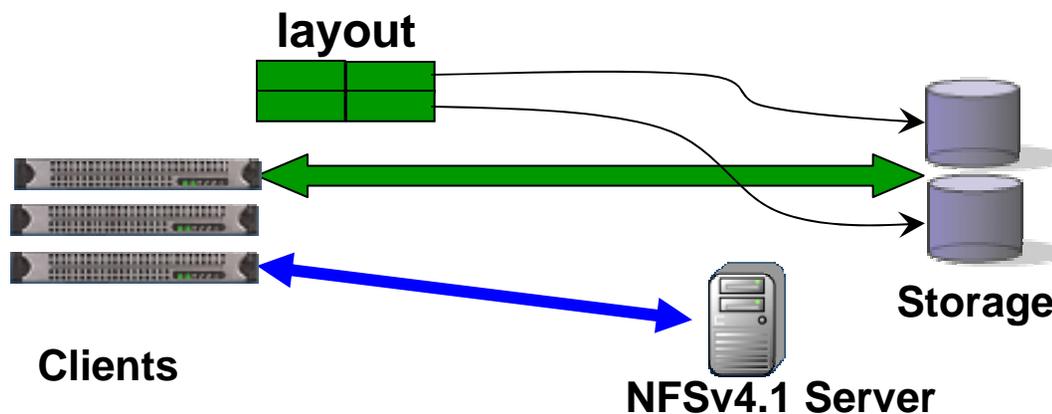
pNFS: The standard for parallel NAS

- pNFS is an extension to the Network File System v4 protocol standard
- Allows for parallel and direct access
 - From Parallel Network File System clients
 - To Storage Devices over multiple storage protocols
 - Moves the Network File System server out of the data path



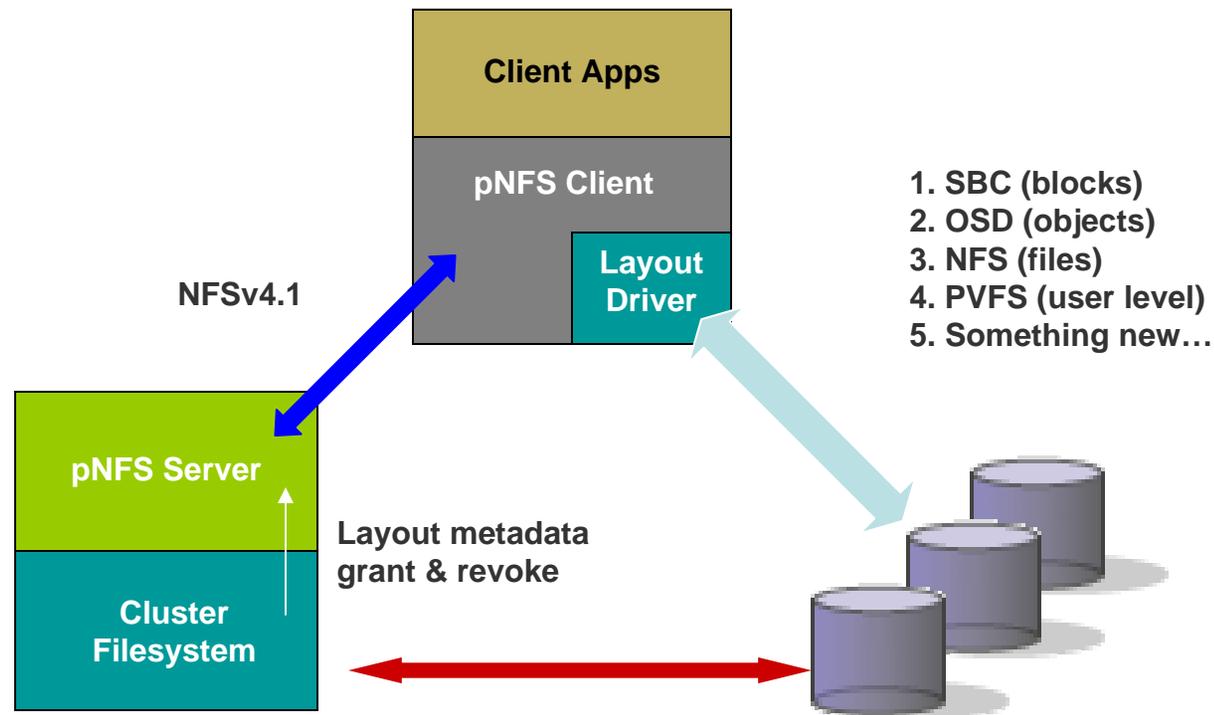
pNFS Layouts

- Client gets a *layout* from the NFS Server
- The layout maps the file onto storage devices and addresses
- The client uses the layout to perform direct I/O to storage
- At any time the server can recall the layout
- Client commits changes and returns the layout when it's done
- pNFS is optional, the client can always use regular NFSv4 I/O



pNFS Client

- Common client for different storage back ends
- Wider availability across operating systems
- Fewer support issues for storage vendors



pNFS Protocol Operations

- LAYOUTGET
 - (filehandle, type, byte range) -> type-specific layout
- LAYOUTRETURN
 - (filehandle, byte range) -> server can release state about the client
- LAYOUTCOMMIT
 - (filehandle, byte range, updated attributes, layout-specific info) -> server ensures that data is visible to other clients
 - Timestamps and end-of-file attributes are updated
- GETDEVICEINFO
 - Map deviceID in layout to type-specific addressing information

pNFS Protocol Callbacks

- NFS Version 4 servers are “stateful”, and they generate callbacks to clients to reclaim state about delegated locks and delegated layouts
 - pNFSv4.1 adds these callback operations
- **CB_LAYOUTRECALL**
 - Server tells the client to stop using a layout, or all layouts
- **CB_RECALL_ANY**
 - Server tells the client to release delegations of its own choosing, in order to let the server reduce the amount of state it is maintaining

Key pNFS Participants



- Panasas (Objects, based on Panasas Storage Cluster OSDs)
- Network Appliance (Files over NFSv4)
- IBM (Files, based on GPFS)
- EMC (Blocks, based on HighRoad MPFSi)
- Sun (Files over NFSv4)
- U of Michigan/CITI (Files over PVFS2, Files over NFSv4)

Current Status



- pNFS is part of the IETF NFSv4 minor version 1 standard draft
 - draft-ietf-nfsv4-minorversion1-13.txt
 - Weekly editorial review meetings started this May
 - Anticipate working group “last call” this October
 - Anticipate RFC being published late Q1 2008
- Prototype interoperability testing began in 2006
 - Connect-a-thon and Bake-a-thon multi-vender testing sessions two or three times a year.
 - March 2007 San Jose. June 2007 Austin. October 2007 Ann Arbor.
- Expect Linux integration into kernel.org by late 2008
- Expect other vendor releases by late 2008

Object Storage Architecture

- Raises storage's level of abstraction
 - From logical blocks to objects (object is a container for data and attributes)
 - Allows storage to understand how different blocks of a object are related
 - Provides storage with necessary info to optimize storage resources
- An evolutionary improvement to standard (SCSI) storage interface

Block Based Disk

Operations:

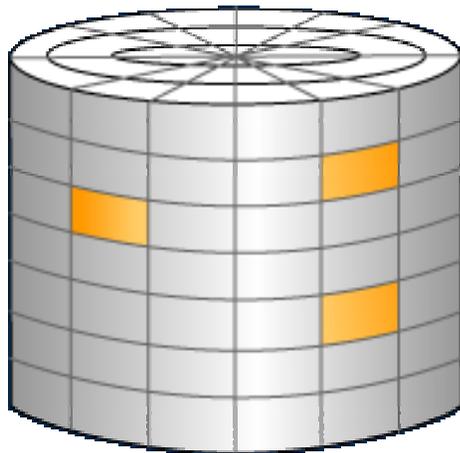
Read block
Write block

Addressing:

Block range

Allocation:

External



Operations:

Create object
Delete object
Read object
Write object
Get Attribute
Set Attribute

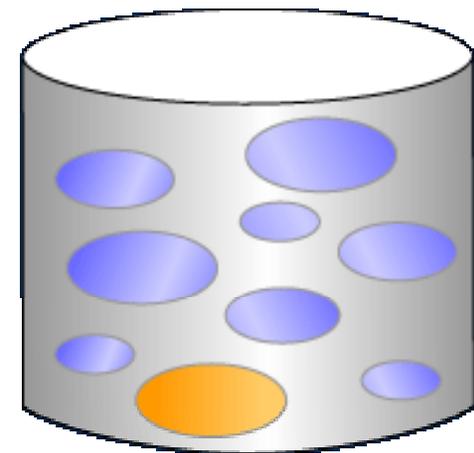
Addressing:

[object, byte range]

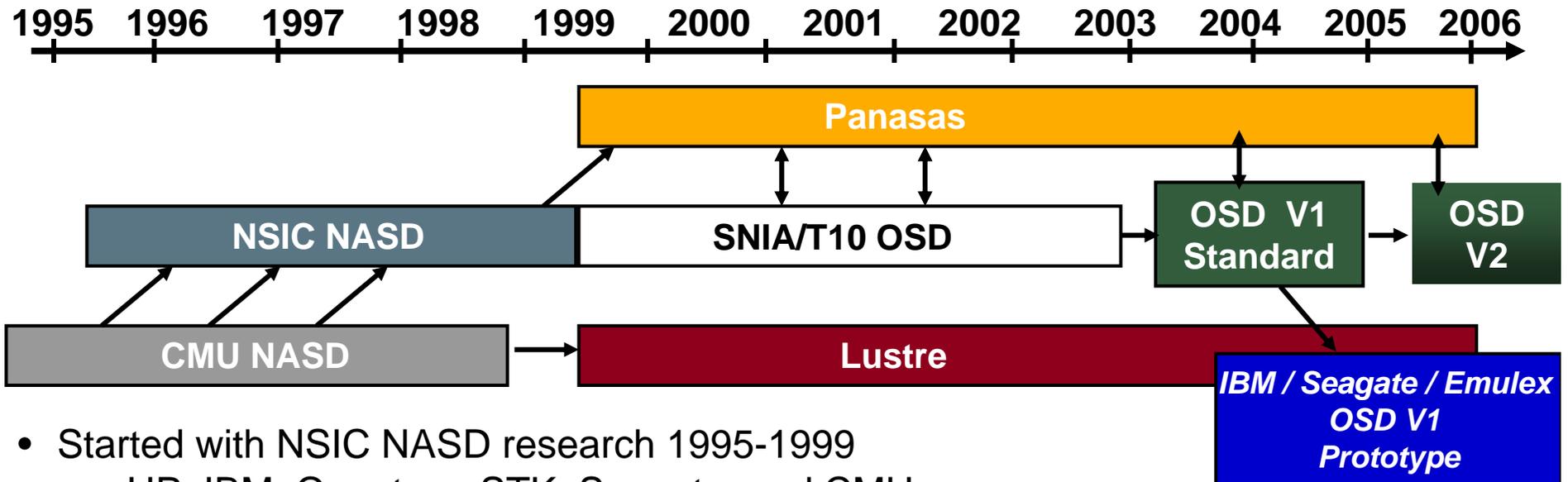
Allocation:

Internal

Object Based Disk



Object Storage Timeline



- Started with NSIC NASD research 1995-1999
 - HP, IBM, Quantum, STK, Seagate, and CMU
 - Eventually became SNIA Technology working group in '99
 - 45 participating companies
- 1999 moves to SNIA/T10 working group
- 1/2005: ANSI ratifies V1 T10 OSD standard (ANSI/INCITS 400-2004)
 - SNIA TWG already working on OSD V2 features
 - Snapshots, import/export, multi-object capabilities and extended attributes



T10 OSD V2.0 Work in Progress

- Collections and Multi-object collection operations
 - Single command instructs OSD to operate on set of objects
- Snapshot support
 - Copy-on-write for objects, collections of objects
- Efficient error handling
 - Fast error detection and recovery
 - Error handling for multi-object operations



Collections

- OSD Collection Object is an object used to store a list of user object IDs
- User objects are added to or removed from a collection by performing a SETATTR on the user object's collection page
 - Enables collection manipulation as side effect of other command (e.g., WRITE)
- Use case 1: Fast Index
 - Transaction needs to record all objects it touches
 - Using piggyback SETATTR(into collection X) to add each object into the collection as the object is dirtied
 - If client fails (e.g., reboots), it can discover which objects are dirty by listing the collection
- Use case 2: List of related objects
 - EX: Pseudo directory of all MP3 objects
- Basic collection commands
 - CREATE COLLECTION, REMOVE COLLECTION

Multi-object Operations

- **GET MEMBER ATTRIBUTES**
 - Returns the specified attribute(s) from every object listed in the collection
- **SET MEMBER ATTRIBUTES**
 - Sets the specified attribute(s) on every object listed in the collection
- **REMOVE MEMBER OBJECTS**
 - Deleted every user object listed in the collection
- **QUERY**
 - Match against one or more specified <attribute, value> pairs, returning the list of a user objects that successfully matched
- **Ordering of internal command processing is unspecified**
 - Allows for efficient disk-directed processing

OSD Snapshot

- OSD V2.0 defines snapshots to be point-in-time copies of partitions
 - Used partition as basis for snapshot because partitions are the basic unit of space management
- OSD keeps list of snapshots
 - parent / child relationships in snapshot attribute page
- Snapshots may be implemented as
 - Efficient copy-on-write
 - Sync byte-by-byte copy
 - Async byte-by-byte copy
- Set of snapshot commands
 - CREATE, DELETE SNAPSHOT
 - REFRESH, RESTORE SNAPSHOT
 - READDIFF, READMAP(peek under object abstraction)



Error Handling – Damaged Objects

- Objects can be damaged for several reasons, including media defects and software bugs
- Manager should be alerted when a damaged object is detected
 - Proactively: OSD sends message to manager
 - Discovered: Object damage is recorded inside OSD and may be queried by manager
- OSD marks damaged objects
 - Specific object is fenced (topmost bit of object version # is set)
 - Prevents client access to object until manager can examine object
 - Partition and root attribute set to timestamp of latest discovered damage
 - Manager can poll timestamps to discover new damage has been detected by OSD

OSD impact on iSCSI stack

- Large CDB
 - OSD commands are big (256 bytes), using the extended CDB format
 - E.g., Linux normally has small in-line struct, now keeps a pointer to large CDB
- Bi-Directional Data Transfer
 - Three buffers involved in a command:
 - Data payload
 - Piggy-back set attributes
 - Piggy-back get attributes
- Open Solaris, Linux support (patches available)
 - Panasas working with Linux maintainers to push changes through iSCSI / SCSI / Block layers

Status

- SNIA working group working through issues
- SNIA goal is to complete OSDv2 document in Fall 2007
 - There will be a general call for comments from SNIA members
 - After this is goes up to the ANSI T10 Standards body

