

Silicon Graphics, Inc.

# **XFS Overview & Internals**

## **12 - DMAPI**

November 2006

# DMAPI

- **Data Management Applications Programming Interface**
- Developed by Data Management Interfaces Group consortium (DMIG), 1993-1996
- Provides standardized access to filesystems for Hierarchical Storage Managers (HSMs) and backup packages
- DMAPI in SLES is used by vendors other than just SGI
  - IBM
  - HP

# HSM Example

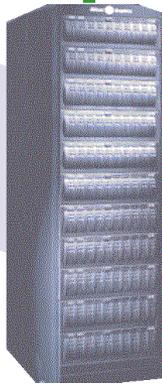
## Primary Storage

Online high-performance disk  
e.g. 2 to 10 hours  
current projects



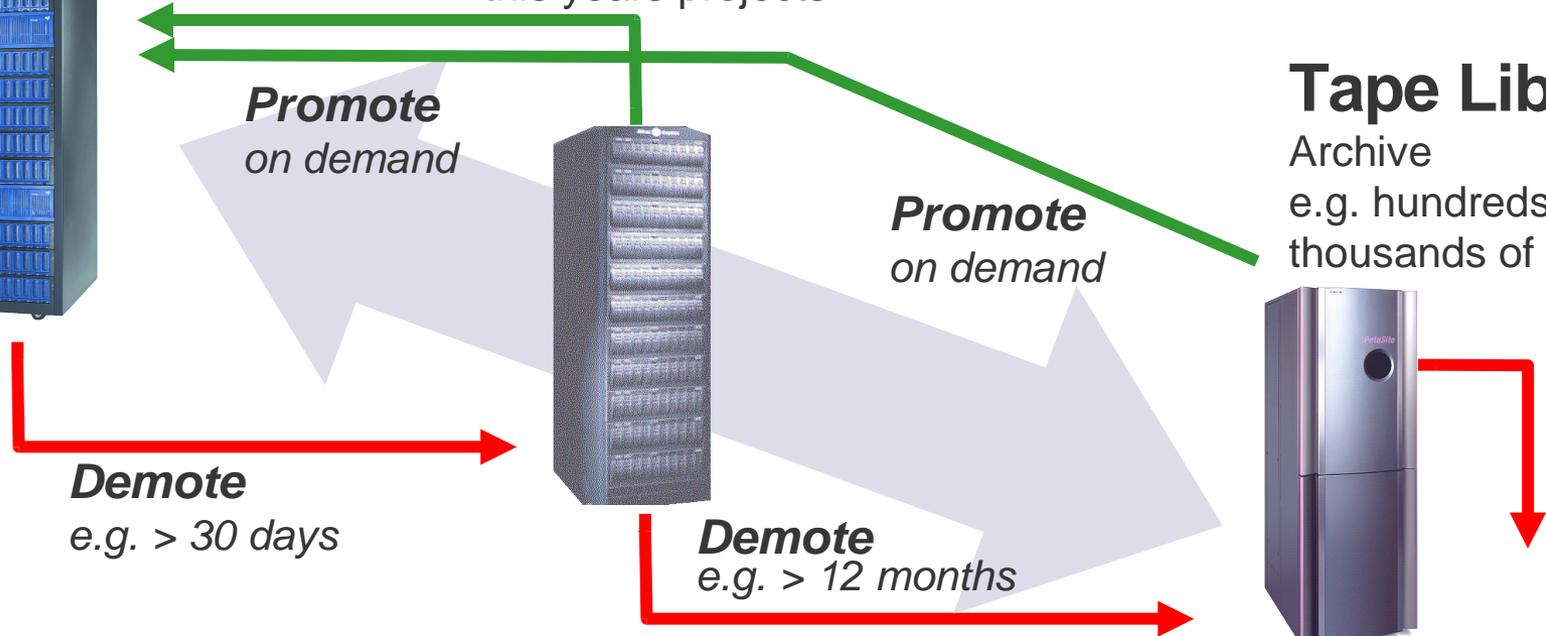
## Nearline Disk

High Capacity, Low cost,  
Lower performance  
e.g. 6 to 20 hours  
this years projects



## Tape Library

Archive  
e.g. hundreds to  
thousands of hours



# HSM Economic Benefit

## Example Environment

2TB data less than 30 days old  
 6TB data between 30 days and one year old  
 30TB data over one year old

### Without HSM

High-Perf. Disk  
 30TB



### With HSM

High-Perf. Disk  
 2TB



Low Cost  
 Disk  
 6TB



Tape Library  
 30TB



<u>System</u>	<u>c/MB</u>	<u>w/o DMF</u>	<u>With DMF</u>
High-Perf. Disk	4c	\$900K	\$80K
Low Cost Disk	0.8c	-	\$50K
Tape Library	0.37c	-	\$110K
DMF		-	\$110K
<b>Total Investment</b>		<b>\$900K</b>	<b>\$350K</b>

# What do HSMs need?

- To be notified of accesses/changes to a file's metadata
- To be able to accept or reject such accesses and changes
- To have user processes block until the change has been approved
- To be able to access a file without knowing a pathname
- To be notified of changes to a file's metadata
- To be able to read and change the contents of a file without blocking
- To save HSM-specific information persistently with a file
- To scan all files in a filesystem quickly
- To be notified when filesystems become full

# DMAPI Standard

- DMAPI Versions 2.1 and 2.3
- X/Open Data Storage Management API (XDASM) - 2/97
  
- Not a rigorous standard.
  - ~70 function calls
  - ~40 structures
  - 24 events
  
- Spec available free in HTML format from <http://www.opengroup.org/pubs/catalog/c429.htm>
  
- Alex Miroshnichenko's paper from AUUG 1996:
- <http://www.csu.edu.au/special/auugwww96/proceedings/alex/alex.html>

# Handles

- Allow file access without pathnames
  - Similar to NFS file handles
- A handle is an opaque persistent identifier which is unique per host
  - Should not change for the life of the object
- Handles are represented as variable length byte streams
  - void\* and size\_t
- An existence of a valid handle does not guarantee existence of the object referenced by it.
  - The only way to guarantee the existence of the object referred to by a handle is to either obtain a right or hold on the object
- Three handle types:
  - Global handle - (used only in mount requests)
  - Filesystem handle - (fsid hash)
  - File handle - (fsid hash/fid\_t structure)

# Handles - DMAPI Functions

- `dm_path_to_handle`
  - create file handle from pathname
- `dm_fd_to_handle`
  - create file handle from file descriptor
- `dm_path_to_fshandle`
  - create filesystem handle from file descriptor
- `dm_handle_to_fshandle`
  - extract the filesystem handle from a file handle
- `dm_handle_cmp`
  - file handle comparison
- `dm_handle_is_valid`
  - determine if a handle is valid
- `dm_handle_hash`
  - hash contents of a handle
- `dm_handle_free`
  - free storage allocated to a handle
- `dm_handle_to_path`
  - get a pathname from two handles

# DMAPI Events and Tokens

- Events are messages from the kernel to a DMAPI application
- Provide notification of accesses/changes to a file or filesystem
- When a synchronous event is generated, the user process is suspended in the kernel until a DMAPI application issues an explicit response to the event
  - A token is a reference to the kernel state associated with a synchronous event
- Asynchronous events are for notification purpose only
  - May indicate a completion (or failure) of certain operations
  - Do not block user processes
- All contain handles, directory names, etc.

# Events - Filesystem Admin

- mount
  - a filesystem is about to be mounted
- preunmount
  - a filesystem is about to unmount
- unmount
  - an unmount succeeded/failed
- nospace
  - filesystem has run out of space
- Not implemented by XFS:
  - debut
    - inode has been read from disk

# Events - Namespace

- create/postcreate
  - create a file or directory
- remove/postremove
  - unlink a filesystem object
- rename/postrename
  - rename a filesystem object
- link/postlink
  - (hard) link a filesystem object
- symlink/postsymlink
  - create a symlink

# Events - Data

- read
  - the specified file byte range is about to be read
- write
  - the specified file byte range is about to be written
- truncate
  - file is about to be truncated to the specified address (away from or towards zero)

# Events - Metadata

- attribute
  - a filesystem object's attribute has changed
- destroy
  - a filesystem object has been destroyed
- Not implemented by XFS:
  - cancel
    - a previously issued event is cancelled
  - close
    - a filesystem object has been closed

# Events - Pseudo

- user event
  - a message between cooperating DMAPI processes

# Sessions and Dispositions

- DMAPI uses sessions as the primary communication channels between DM application and the kernel component of DMAPI
  - Almost all DMAPI calls require a session argument
- Sessions and event lists allow DM application to exercise a fine control of the event delivery and generations
  - Applications may direct different event types to different sessions and avoid session overloading
- Dispositions indicate which event types should be delivered to this session.
  - A DM application creates a session by calling `dm_open_session()` and then may register event dispositions for this session by calling `dm_set_disp()`

# Connecting Events with Sessions

- Select desired events to be generated
  - `dm_eventlist_t` bitmap
- Set by filesystem
  - not persistent
- Set by file
  - persistent
- Set globally
  - to trigger mount events
- Next set disposition of those events
  - recipient session

# Connecting Events - DMAPI Functions

- `dm_get_eventlist`
  - get list of enabled events for object
- `dm_set_eventlist`
  - set list of enabled events for object
- `dm_set_disp`
  - set disposition of events on a filesystem to a particular session
- `dm_set_return_on_destroy`
  - specify a DMAPI attribute to return with destroy events

# Managing Sessions - DMAPI functions

- `dm_create_session`
  - create a new session
- `dm_query_session`
  - query a session for information
- `dm_destroy_session`
  - destroy the specified session
- `dm_getall_sessions`
  - get all extant sessions

# Processing Events - DMAPI Functions

- `dm_get_events`
  - get next available event messages
- `dm_respond-event`
  - respond to one event
- `dm_pending`
  - notify FS of slow application event
- `dm_create_userevent`
  - generate a user pseudo-event
- `dm_send_msg`
  - send message to indicated session
- `dm_move_event`
  - move event to another session

# Session Recovery

- `dm_getall_sessions`
  - get all extant sessions
- `dm_query_session`
  - query a session for info
- `dm_create_session`
  - create/assume a session
- `dm_getall_tokens`
  - get outstanding tokens for a session
- `dm_find_eventmsg`
  - get message for an event
- `dm_getall_disp`
  - get event dispositions for all filesystems for a session

# Managed Regions

- A managed regions is the mechanism for an application to control file data access at a granularity level less than file size. It is an extent in the logical file space
  - starting offset
  - length
  - event generation flags
- Provide byte-granularity events for all accesses/modifications of a file's data
  - Only apply to file objects
- Cover non-overlapping byte ranges in a file
- Separately detect reads, writes, truncates
- Only one, persistent in XFS

# Managed Regions – DMAPI Functions

- `dm_get_region`
  - get managed regions for a file
- `dm_set_region`
  - set managed regions for a file

# File Access and Modification

- Must not change mtime or atime (ctime changes)
  - TODO: Bug
- Must not block
- Must not generate events

# File Access and Modification – DMAPI Functions

- `dm_probe_hole`
  - return rounded result of the area where a hole is to be punched
- `dm_punch_hole`
  - create a hole in a file
- `dm_read_invis`
  - read a file bypassing DMAPI events
- `dm_write_invis`
  - write a file bypassing events
- `dm_sync_by_handle`
  - sync a file's data to disk
- `dm_get_fileattr`
  - return file attributes
- `dm_set_fileattr`
  - set file time stamps, ownership, mode, length, etc.
- `dm_get_allocinfo`
  - return allocation info for a file

# Persistent DMAPI Data

- Stored per file as SGI\_DMI\_xxx extended attribute
- Persistent across reboots
- DMF stores in SGI\_DMI\_DMFATTR
  - bfid, file state
  - version info
  - Flags
- inodes should be 512 bytes (-i size=512)

# Persistent DMAPI Data – DMAPI Functions

- `dm_get_dmattr`
  - retrieve a DMAPI attribute
- `dm_set_dmattr`
  - set/replace a DMAPI attribute
- `dm_remove_dmattr`
  - remove a DMAPI attribute
- `dm_getall_dmattr`
  - retrieve all DMAPI attributes for a file

# Scanning Filesystems Quickly

- `dm_init_attrloc`
  - initialize a bulk attribute location offset
- `dm_get_bulkattr`
  - get bulk attributes for entire filesystem
- Not implemented by XFS:
  - `dm_get_bulkall`
    - get data management attributes for entire filesystem

# Rights

- DM applications use the tokens to obtain access rights to file system objects to guarantee stability of the objects.
  - Allow session to control access/updates
  - Provide for atomic updates over many functions
- Access rights may be *shared* and *exclusive*
  - Shared right allows read-only access (DM\_RIGHT\_SHARED)
  - Exclusive right allows modification to the object (DM\_RIGHT\_NULL)
- Pseudo-available in XFS

# Rights – DMAPI Functions

- `dm_release_right`
  - release all access rights to an object
- `dm_query_right`
  - determine set of access rights to an object
- `dm_upgrade_right`
  - upgrade current right to exclusive
- `dm_downgrade_right`
  - downgrade current right to shared
  
- Not implemented in XFS:
  - `dm_request_right`
    - request a specific access right

# XFS DMAPI Mount Options

- Options must be used together
- `dmapi`
  - Enable the DMAPI event callouts
  - `xdsm` and `dmi` options are equivalent
- `mtpt=mountpoint`
  - The mountpoint specified here will be included in the DMAPI mount event, and should be the path of the actual mountpoint that is used.

```
> grep dmapi /etc/fstab
/dev/sdb1 /mnt/scratch1 xfs dmapi,mtpt=/mnt/scratch1,defaults 0 0
```

```
> mount
...
/dev/sdb1 on /mnt/scratch1 type xfs (rw,dmapi,mtpt=/mnt/scratch1)
...
```

# What if there is no DMAPI module?

```
# mount -o logdev=/dev/sdb,dmapi,mtpt=/mnt/data /dev/sdd /mnt/data
mount: wrong fs type, bad option, bad superblock on /dev/sdd,
       missing codepage or other error
       In some cases useful info is found in syslog - try
       dmesg | tail  or so
# dmesg | tail | grep dmapi
XFS: unknown mount option [dmapi].
```

- Check that DMAPI is probed on boot and loaded
  - SLES10 set DMAPI\_PROBE to yes in /etc/sysconfig/sysctl
- Without rebooting XFS with load dmapi with

```
# echo 1 > /proc/sys/fs/xfs/probe_dmapi
# lsmod | grep dmapi
xfs_dmapi                22936  1
dmapi                    39728  1 xfs_dmapi,[permanent]
xfs                      479796  4 xfs_dmapi,xfs_quota
```

# DMAPI Implementation

- Not all DMAPI interfaces used by SGI's HSM product
  - Many of the bugs recently fixed in DMAPI did not impact SGI's HSM
- Some DMAPI interfaces are not implemented by XFS
- Maintaining 2.4/2.6 compatible code makes it difficult to read
  - Plan is to remove 2.4 support from this code base
- DMAPI requires two kernel changes that are not in mainline
  - mprotect
  - open\_exec

# DMAPI Libraries

```
> rpm -qpl dmapi-2.2.3-12.2.ia64.rpm  
/lib/libdm.so.0  
/lib/libdm.so.0.0.4  
/usr/share/doc/packages/dmapi  
/usr/share/doc/packages/dmapi/CHANGES.gz  
/usr/share/doc/packages/dmapi/COPYING  
/usr/share/doc/packages/dmapi/PORTING  
/usr/share/doc/packages/dmapi/README
```

# DMAPI Development Package

```
> rpm -qp1 dmapi-devel-2.2.3-12.2.ia64.rpm  
/lib/libdm.so  
/usr/include/xfs/dmapi.h  
/usr/lib/libdm.a  
/usr/lib/libdm.la  
/usr/lib/libdm.so  
/usr/share/man/man3/dmapi.3.gz
```

# DMAPI Kernel Modules

```
> rpm -qpl kernel-default-2.6.16.21-0.8.ia64.rpm | grep dmapi
/lib/modules/2.6.16.21-0.8-default/kernel/fs/dmapi
/lib/modules/2.6.16.21-0.8-default/kernel/fs/dmapi/dmapi.ko
  - Filesystem independent module
/lib/modules/2.6.16.21-0.8-default/kernel/fs/xfs/dmapi
/lib/modules/2.6.16.21-0.8-default/kernel/fs/xfs/dmapi/xfs_dmapi.ko
  - XFS dependent module
```

# DMAPI Triage

**sgi<sup>®</sup>**