XFS Practical Exercises

10 - DMAPI

Overview

HSM is a data storage application that automatically moves data between high speed/high cost and low speed/low cost storage media. To make the HSM application independent of the underlying filesystem implementation, the services of DMAPI (Data Management API) are used. DMAPI is an implementation layer siting between the HSM and the filesystem.

The HSM lab demonstrates a sample HSM implementation over a DMAPI XFS filesystem. The source code and the binary programs of the sample HSM implementation are located in

```
xfs-cmds/xfstests/dmapi/src/sample_hsm
```

The sample HSM:

1. Migrates selected files from a DMAPI filesystem to a tertiary storage. The file selection criteria is file size bigger than a given threshold. The tertiary storage is a staging directory on a DMAPI filesystem.
2. Sets up DMAPI managed regions on the files that have been migrated.
3. Releases (frees) the data blocks on the files that have been migrated.
4. Creates a DMAPI session and sets read/write/truncate event disposition of the managed region events.
5. Waits for events on a migrated file. On read event restores the file. On write/truncate event invalidates the file (assumes the whole file contents will be changed/renewed).

migfind, migout, migin and wbee are the programs implementing the sample HSM. mrmean and mls are utility tools.

Goals

The goal of these exercises is to understand the principles of operation of a HSM and its interactions with DMAPI. This knowledge could be used for new custom HSM development or analysing and triaging DMAPI problems with third party HSM implementations.

Prerequisites

Access to DMAPI standard for reference to the functions used in the sample HSM.

Setup

- Recent Linux kernel with XFS
  - SLES10
  - 2.6.18 or later kernel using http://oss.sgi.com/cgi-bin/cvsweb.cgi/linux-2.6-xfs/
- In addition to a standard build environment, the following packages are installed
  - xfsprogs-devel 2.7.11-18 or later
  - dmapi-devel 2.2.3-12 or later
- Build the dmapi tools

```
# cd xfs-cmds/dmapi
# autoconf
# ./configure
# make
```

- Build the dmapi test tools

```
# cd xfs-cmfs/xfstests/dmapi
# autoconf
# ./configure
# make
```

- SGI XFS user space commands xfs-cmds version 2.8.14 or later. Note this is a more recent version of xfs-cmds than is included in SLES10.
- Empty XFS DMAPI filesystem on the machine used by the sample HSM (no important data should be kept on this filesystem).
- Empty staging directory on the DMAPI filesystem to store copies of the files migrated by HSM.
- Create two text files on the dmapi filesystem

```
# dmesg | dd of=/mnt/dmapi_filesystem/small_file.txt ibs=4k count=1
# dmesg | dd of=/mnt/dmapi_filesystem/big_file.txt ibs=4k count=3
```

- Build the dmapi test programs

```
# cd xfs-cmds/xfstests/dmapi
# ./configure
# make
```

For the purpose of this lab all programs should be run as root.

The following paths are used in the exercises:

Path	**Description**
`/mnt/dmapi_filesystem`	Name of DMAPI filesystem
`/mnt/dmapi_filesystem/stagedir`	Staging directory
`/mnt/dmapi_filesystem/small_file.txt`	Small test file
`/mnt/dmapi_filesystem/big_file.txt`	Large test file
`xfs-cmds/xfstests/dmapi/src/sample_hsm`	Location of sample HSM

Exercises

Exercise 1

List the implemented DMAPI events on this filesystem:

```
# cd xfs-cmds/xfstests/dmapi/src/suite1/cmd
# ./path_to_fshandle /mnt/dmapi_filesystem
fc88828964790000
# ./get_config_events fc88828964790000
```

Exercise 2

Create a list of files to be migrated by the HSM. A real HSM selects the files to be migrated using criteria, like a large file that hasn't been accessed recently.

In this exercise the sample HSM program migfind creates a list of files with size bigger than a threshold value. The file list contains one line (with three fields) per DMAPI file. The three fields are:

- DMAPI handle length
- DMAPI handle converted to an ASCII string
- file size

In the next exercise the file list will be used by migout program.

To create the list of files bigger than 10Kb run:

```
# cd xfs-cmds/xfstests/dmapi/src/sample_hsm
# ./migfind -s 10k /mnt/dmapi_filesystem >& /tmp/cand_file
```

Check the contents of the list file cand_file

```
# cat /tmp/cand_file
24      5d1111a90e4800000e00000000000000103c00100000000        12288
```

It contains one line with three fields. These three fields are the handle length, handle and the file size of `/mnt/dmapi_filesystem/big_file.txt`.

Exercise 3

Migrate files to the stage directory. The migration performs the following tasks:

1. Creates and opens files in the staging directory. The file names are the ASCII representation of the original file DMAPI handle.
2. Copies the original file data to the corresponding file in the staging directory. It uses dm_read_invis() to preserve the a_time of the original file.
3. Saves the location of the staging file for future file restoration. The location is saved (in the form of DMAPI attributes) in the original file via dm_set_dmattr() interface. The location is stored as two attributes - "DMAPI handle" and "length of handle" instead of using a full path name to the staging file.
4. Sets up managed regions on the original file via dm_set_region() interface.
5. Releases/removes all but the first 8Kb of the original file data via dm_punch_hole() interface.

The list of files to be migrated (cand_file) was created in Exercise 1. Check that we do not have a DMAPI event mask for this file:

```
# cd xfs-cmds/xfstests/dmapi/src/suite1/cmd
# ./get_eventlist /mnt/dmapi_filesystem/big_file.txt
```

To migrate the files in the list run:

```
# cd xfs-cmds/xfstests/dmapi/sample_hsm
# ./migout /mnt/dmapi_filesystem/stagedir < /tmp/cand_file
```

Check a copy of big_file.txt has been created:

```
# ls -al /mnt/dmapi_filesystem/stagedir
# head
/mnt/dmapi_filesystem/stagedir/5d1111a90e4800000e000000000000000103c0010000000
0
```

Do not try to read from or write to `/mnt/dmapi_filesystem/big_file.txt` at this time, but we can see that a DMAPI event mask is not set on this file:

```
# cd xfs-cmds/xfstests/dmapi/src/suite1/cmd
# ./get_eventlist /mnt/dmapi_filesystem/big_file.txt
```

Exercise 4

Restore a migrated file by opening it in vi text editor. The file restoration triggered by vi is actually done by the HSM program migin running as a daemon and the helper program wbee.

The functions migin performs are:

1. Creates a DMAPI session.
2. Sets read/write/truncate event disposition for the DMAPI filesystem via dm_set_disp() interface.
3. Waits for DMAPI events via dm_get_events() interface.
4. On DMAPI event spawns a child process with arguments event type, event token and the DMAPI session id. The span process is the wbee program described next.

The functions wbee performs are:

It is started by the migin program (described above) when a DMAPI event is

generated.

1. Retrieves the DMAPI event message via dm_find_eventmsg() interface.
2. On read DMAPI event:
    a. Retrieves the DMAPI handle and the handle length of the original file from the event message.
    b. Retrieves the DMAPI handle and handle length of the staging file via dm_get_fileattr() interface using the handle from 2a.
    c. Reads chunks of data from the staging file via dm_read_invis() interface using the handle from 2b.
    d. Writes the chunks of data to the original file via dm_write_invis() interface using the handle from 2a.
    e. Removes the original file DMAPI managed regions via dm_get_region()/dm_set_region() interfaces.
    f. Responds to the DMAPI event via dm_respond_event() interface. At this point all data has been restored to the original file. The application that generated the DMAPI read event will be unblocked and finish its read.
3. On write/truncate DMAPI event invalidates the DMAPI file data by:

a. Removing the original file DMAPI managed regions via dm_get_region()/dm_set_region() interfaces.

b. Responding to the DMAPI event via dm_respond_event() interface. At this point the application that generated the DMAPI write/truncate event will be unblocked and finish its operation.

To restore the /mnt/dmapi_filesystem/big_file.txt perform the following steps:

Check the file map shows only 8Kb resident data:

```
# ./mls –aehm /mnt/dmapi_filesystem/big_file.txt
```

Note that `/mnt/dmapi_filesystem/big_file.txt` has only one resident extent.

In another window open the file with vi:

```
# vi /mnt/dmapi_filesystem/big_file.txt
```

At this point vi hangs and no text is displayed because there is no DMAPI session to handle the read DMAPI event.

In another terminal window run:

```
# ./migin -v /mnt/dmapi_filesystem
Received read, token 1453
execl(wbee, wbee, -r, -s, 37, -t, 1453, 0)
```

migin creates a DMAPI session (37), receives notification of the read DMAPI event (token 1453) and uses wbee to process the event.

In the first terminal window, vi is unblocked and displays the `/mnt/dmapi_filesystem/big_file.txt` contents. Now exit vi and run:

```
# ./mls -aehm /mnt/dmapi_filesystem/big_file.txt
```

A proper HSM will have a process waiting for event notifications at all times so the only delay the user will observe is the time it takes for the HSM to migrate the data back to the filesystem.

Questions

1. What is the primary goal of a HSM application?
2. What is the primary goal of DMAPI?
3. Why does wbee program use dm_write_invis() instead of write() to restore the original file?

Answers

1. To migrate less active data off primary storage and move it to less expensive storage, while keeping it available to users.
2. Providing a consistent, platform and OS independent interface for development of HSM applications.
3. Three reasons.
   a. dm_write_invis() preserves the original file timestamps.
   b. dm_write_invis() does not generate DMAPI write events.
   c. The sample HSM stores and uses the DMAPI handle instated of the path to the staged file.