Improving Recoverability by Utilizing DMF

Nicholas P. Cardo, Sterling Software, Inc., Numerical Aerodynamic Simulation Facility, NASA Ames Research Center, Moffett Field, CA

ABSTRACT: *The Cray Data Migration Facility (DMF) can be utilized to improve system recoverability after fatal failures. The High Speed Processors (HSP) group at the Numerical Aerodynamics Simulation (NAS) Facility at NASA Ames Research Center have developed utilities to take advantage of DMF to reduce recovery time from a database or filesystem loss. By taking daily snapshots of the databases and archiving them without stopping DMF, recovery time from a database loss is reduced. Throughout the day, files are forced to become dual resident. Since the data resides on DMF managed tapes, it is only necessary to backup the inode, not the data. On our 150GB filesystem, this caused a reduction in backup time from 8 to 12 hours down to under 2 hours and a reduction in tapes from over 50 down to under 10. This allows us to have a more accurate representation of the filesystem on tape. Also, these short backups allow for faster recovery time when filesystems need to be remade. This paper will discuss how these utilities work and how they are utilized to reduce our down time during a fatal catastrophe.*

Cray Research's Data Migration Facility has become an integral part of many high performance computing centers. An area of concern at the NAS facility is the ability to recover the DMF databases and the ability to perform an accurate restore of the large migratable filesystems. In addition to accuracy, the mean time to recovery was an important driving factor.

1 Database Backups

Over the course of a day, thousands of files can be altered. Some files may have been deleted while others updated. In either case, a database update is made to reflect the change in the file. Additionally, the transaction of updating the database is appended to a journal file.

1.1 Standard Recovery

The journal file can be applied to an old copy of the database in an attempt to bring it up to date. The process to complete this is:

```
# dmdump dmdbase.dat > oldtxt
# dmdjournal -t oldtxt -j oldjrnl -o newtxt
# dmdbase -c newtxt
```

However, if the last database backup is a month old, there are a lot of journal entries to apply to the database. This is a time consuming process which increases the down time. A better approach would be to make database backups on regular intervals, thereby keeping the journal files small.

1.1.1 DMF 2.0 Solution

At this and earlier releases, the only way to start a new journal file is to restart DMF. With a large database, this was taking almost 25 minutes to complete. This outage was not acceptable and a new solution sought.

A new utility, `dmaudit`, was introduced at this release. The `dmdaemon` was updated to support the functions of this utility. `dmaudit` works by sending a request to the `dmdaemon` to make an accurate copy of the databases. A small local modification was made so that another local command, `dmsnap`, could send a request to the `dmdaemon` to snapshot the databases as well as start a new journal file.

The local modification of 27 lines, provided the capability of obtaining an accurate copy of the databases as well as start a new journal file without having to restart DMF. Eventually, design SPR 78731 was opened with CRI.

A problem arose where every so often the `dmdaemon` would hang. The problem was traced to some signal processing code within the daemon. By adding the local modification, the existing DMF problem became visible. SPR 66232 was opened against this problem and a fix provided by CRI.

1.1.2 DMF 2.1 Solution

The concepts presented to CRI in SPR 78731 were beginning to be implemented. At this release, a new journal file is automatically started for each day. This solved the problem of having to restart DMF in order to start a new journal file. However, there still remained the need for a command to initiate the safe duplication of the DMF databases. The locally developed command `dmsnap` is still needed at this release.

1.1.3 DMF 2.2 Solution

All the components for making a safe backup of the databases as well as the restarting of journal files are now in place and delivered with DMF. A new command, `dmsnap`, was introduced at this release and will cause the `dmdaemon` to safely snapshot the databases. The journal file is automatically restarted at midnight. This provides a journal file that represents the activities of a single day. All this can be accomplished without interrupting production operation.

1.2 *Performing the Database Backup*

Ideally, the database backup and the start of the new journal file should coincide with each other. Since the new journal file starts at 00:00 each day, the database backup should also be performed at 00:00 each day. This provides an accurate copy of the database with a relatively small journal file to apply in the event of a catastrophe.

A script was written that executes the `dmsnap` command and copies the databases to tapes stored in our robots.

1.3 *Summary*

This solution gives us the luxury of having an accurate copy of the DMF databases on tape while keeping the size of the journal files to recover from small. The process greatly minimizes the down time due to a catastrophic database loss. At worst case, the copy of the database used for recovery would be 24 hours. As an added safety precaution, 7 days worth of databases are kept in robotic storage and are automatically cycled through.

2 Reducing Filesystem Backups

Performing filesystem backups are an important aspect of recoverability. The ability to accurately restore the filesystem is the key to recoverability. As filesystems grow, so does the length of time to perform a filesystem backup. Since it is not practical to shutdown to single user mode to perform system backups, a way of keeping the filesystem backups short is needed.

2.1 *Short Backups*

Short backups can be defined with two characteristics. The first is the length of wall clock time to produce the backup and the second is the amount of physical secondary media used for the backup.

Our 150 gigabyte filesystem has approximately 1 terabyte of data managed by DMF. It would require 50 - 60 3490E tapes and 8 - 12 hours to complete the backup. Due to the long duration of performing the backup, upon completion the backup was no longer an accurate representation of the filesystem. Over the course of 8 - 12 hours, the filesystem can drastically change. Also, the use of 50 - 60 tapes for one filesystem introduces potential logistical problems with the management of the tapes. Additionally, the length of time to perform a restore is dependant on two factors, the number of files and the number of tapes to read. The length of time to restore from 50 - 60 tapes was just too long, increasing down time.

2.2 *Goals*

A task was created to address the problem of performing backups on large filesystems. For an accurate backup of a large volatile filesystem, it was determined that the backup should complete within 2 hours. Also, for improved recoverability, the number of tapes would need to be reduced. Assuming 7 minutes per tape in streaming mode, this would limit the dump to under 18 tapes.

The task focused on a way to off-load the large amounts of data normally backed up manually on a nightly basis, onto DMF. Three requirements were formalized

1. The backup should complete in under 2 hours.
2. On-line resident files would still be on-line.
3. Not affect normal production processing.

The general idea is that to the users, there would appear no change in the residency of their files. This meant that force migrating the filesystem and using dmfill was not acceptable.

2.3 *Design*

The conceptual design was a straightforward process of creating a list of files to migrate, migrate the data to tape, then unmigrate the same files.

2.3.1 Creating Migration List

Some filtering was necessary for the creation of the migration list. The program `dmmighit` was designed to scan the inodes of a filesystem and produce a list of files, based on inode information, to be migrated. Since an inode scan is used, the list could be produced within 4 minutes. Because normal production could not be impacted, any file currently being used could not be migrated. Also, since the file would end up on-line, evaluating `.keep` files would not be necessary. As a result, the following guidelines are used for the selection of files to migrate. Any file meeting all the rules will be migrated. The selection rules are:

1. The file must be `S_IFREG`.
2. The size of the file must be greater than or equal to `MIN_DM_SIZE`.
3. The file must not have a handle.
4. The file must not have been accessed in the last hour.
5. The file must not have been modified in the last hour.
6. The owner must be a valid user.

Enough information needed to be written to the migration list to allow for proper migration and unmigration of the file. The entry in the migration list is:

```
uid|gid|device|inode|generation|size
```

The information produced from `dmmighit` can be passed on to the next step, migration.

2.3.2 Migrating the Data

The migration process is actually two steps. The first step is to verify that the entry read from the migration list is still valid. The second step is to send an actual migration request to the `dmdaemon`.

The system call `dmofrq` works two sets of command arguments. If the command argument is specified in lower case letters, then it is expecting to have a pathname supplied as the file pointer argument. If the command is specified in upper case letters, then the file pointer argument is the structure `dm_dvino` which contains the device and inode number for the file. By using the command "S" (status) with `dmofrq`, it is possible to obtain the `stat` structure for the device and inode specified. Both the device and inode numbers are available from the migration list. An example of using dmofrq in this manner would be:

```
struct dm_dvino dvi;
struct stat stb;

        dvi.dm_dev = device_number;
        dvi.dm_ino = inode_number;
        dmofrq(&dvi, 'S', &stb, NULL, 0);
```

This will return the `stat` structure for the inode specified by `inode_number` on the device `device_number`. This effectively executes the `stat` system call on a file without knowing the pathname to the file, only the inode number. This can be useful in other areas as well, not just for DMF.

Having the `stat` structure, some preliminary checks can be performed before building the daemon request. Two simple checks are to see if the size of the file has changed or if the owner of the file has changed. If either of these have changed, it could indicate that the file is being updated or that the inode has been reassigned to another user. No action is taken against any file suspected of being active or changed. The data in the inode must match the data in the migration list in order to be considered for migration.

After verifying that the file should still be migrated, a daemon request can be formulated. The first step is to get the Media Specific Processes (MSP's) to use for the backup request for the `dmdaemon`. To obtain this information, the `dmcom` library function `dmmfunc` is used. This information is necessary in order to send the request to the `dmdaemon`. The `dmmfunc` call is as follows:

```
int nc; /* number of copies */
int pmsp, smsp; /* primary/secondary msp */
int archmed; /* archive media from udb */

    dmmfunc(&stb,archmed,nc,&pmsp,&smsp);
```

The `stb stat` structure was filled in from the call to `dmofrq` to stat the inode as previously discussed.

It is necessary to perform some preliminary processing prior to sending the request. Having gathered all the information, it is necessary to construct the migration request. The following code builds the migration request packet, `mptr`, sends the request to the daemon and reads replies.

```
struct migreq *mptr = NULL;
    mptr->path[0] = '\0';
    mptr->devno = dvi.dm_dev = devno;
    mptr->inode = dvi.dm_ino = inode;
    mptr->gen = gen;
    mptr->msp_bitmask = 1 << (pmsp - 1);
    if (smsp)
        mptr->msp_bitmask |= 1 << (smsp - 1);
    mptr->flags = 0;
    mptr->pathlen = strlen(mptr->path);
    size = (size + BSIZE - 1) / BSIZE;
    send_request(DRQ_BACK, mptr,
        LMIGREQ(mptr), size, port);
    do_some_replies();
```

When enough data has been sent for migration to fill a tape, the MSP's will begin mounting tapes and writing the data to tape. The process of migrating the files in the migration list is the sole purpose of the program for the second step, `dmmigput`.

2.3.3 Unmigrating the Data

The process of unmigrating a file is much simpler than the process of migrating a file. A recall request is constructed and sent to the daemon for processing. All components for constructing the recall request are available in the migration list. The following code segment constructs the recall request and sends it to the daemon for processing.

```
static struct dmn_req {
    struct recreq recreq;
    char buffer[(2 * FILENAME_MAX) + 1];
} dmn_req;

dmn_req.recreq.devno = devno;
dmn_req.recreq.inode = inode;
dmn_req.recreq.gen = gen;
dmn_req.recreq.msp_bitmask = 0;
dmn_req.recreq.flags = 0;
dmn_req.recreq.pathlen = 0;
dmn_req.recreq.path[0] = '\0';
send_request(DRQ_RECL,&dmn_req,
    LRECREQ(&dmn_req.recreq),0,S_DMP_DMF0);
do_some_replies();
```

This functionality of recalling migrated data based on the migration list was built into the utility `dmmighit`. A recall request is issued for each file in the migration list.

2.3.4 dmmig

A controlling script, `dmmig`, was written to execute the three steps. The first step is to execute `dmmighit` to produce a migration list.

The second step is to execute `dmmigput` to send backup requests to the `dmdaemon` for each file in the migration list. Since `dmmigput` is only sending requests to the `dmdaemon`, it will complete before the MSP's complete the migration of data to tape. Because of this, it is necessary to check for when the request pipe, `dmd.req.pipe`, is empty as well as when all `dmtpput` processes have completed. Recalling data prior to this will cause backup requests to be cancelled.

Once the request pipe, `dmd.req.pipe`, is empty and all `dmtpput` processes have completed, `dmmigget` can be executed, which sends recall requests to the `dmdaemon`. Since the data should still reside in the `.dmpre` directory, the data blocks are just reassigned back to the original inode; no tapes are mounted. At this point the data blocks reside on both tape and disk, making the file dualstate. By using the -a parameter on the dump command, only inode information is dumped for dualstate files. The end result is that to the user, there appears no change in the files residency.

2.4 Summary

By utilizing `dmmig`, backups have been successfully reduced. A full backup of the filesystem typically completes in under 90 minutes and requires 6 tapes. The shortest backup has been 45 minutes and 3 tapes. Another benefit was achieved by utilization of this utility. It is estimated that the control room staff have gained back 198 man hours each month.

3 Improved Recoverability

The driving force behind both utilities was to improve recoverability in the event of a catastrophe as well as reduce recovery time.

The first utility `dmsnap`, reduces our recovery time by making daily snapshots of the databases and storing them on tape. This not only insured that a current copy of the databases are available but that they can be restored to full production in a short period of time. This is due to the fact that only one journal file needs to be processed. Additionally, this can be accomplished without impacting regular production. We have increased our ability to recover from a disaster by keeping seven days worth of databases on tape. Although it is hoped that the copies made by this utility are never needed, it is a comfort and luxury that are easily afforded.

The second utility, `dmmig`, has proven to be a valuable tool. Our mean time to recovery has been drastically improved as well as reducing the impact on staff. It is estimated that the recoverability of this filesystem was increased by 91%. With a time savings of 198 man hours per month, the equivalence of 1 person has been gained back over a year. The logistical problems with maintaining vast amounts of backup tapes has also been reduced. Since our site makes 2 copies of the DMF data, an additional margin of safety has been achieved.

The safeguarding of research data is a high priority. The combination of both utilities provides us with an easy and quick recovery path. These utilities have strengthened our commitment to providing safe data storage for our user community.

4 Lessons Learned

Although we are safely and effectively running these utilities in production today, there were obstacles that needed to overcome.

The `dmsnap` utility, available with DMF 2.2, raised the question of when to take the snapshot. The journal files are automatically restarted at 00:00 each day. By running dmsnap at 00:01 each day, any updates to the database are confined to 1 journal file. Under heavy DMF activity when the dmsnap is run, it is possible that the snapshot could be internally delayed by the `dmdaemon`. Under this condition, it may be necessary to use 2 journal files to perform proper recovery. But recovering from 2 days worth of journal files is much more beneficial than recovering from a whole months, or more, of journal entries.

Although the design of the `dmmig` tools is straight forward, there was a side effect that required resolution. DMF utilizes a single pipe to communicate to the `dmdaemon`. As with `dmmctl`, it is possible with `dmmigput` and `dmmigget` to fill the pipe causing delays in normal user activity. These delays can be as long as an hour, making for dissatisfied users. Three steps were taken to overcome this problem. The utility `dmmighit` accepts as a command line parameter, the number of files to migrate. This reduces the load on the `dmdaemon`. Additionally, running `dmmig` multiple times throughout the day also reduces the number of requests to process at once. An analysis of file sizes provided us the means to set MIN_DM_SIZE effectively. The combination of these steps reduced the impact of running `dmmig`. This utility is in production on two systems. One system migrates approximately 3000 files each day. The second system migrates approximately 1500 files each day. Both system run `dmmig` 3 times a day.

5 What Else is Needed

The integration of the `dmsnap` command gives administrators the ability to obtain an accurate and safe copy of the DMF databases without stopping DMF. The internal workings of performing the snapshot will idle DMF activity before actually copying the databases. Meanwhile, requests can queue up. This functionality can be harnessed for another application, database compressions.

If a command were available that worked similar to `dmsnap`, database compressions could be performed without stopping the dmdaemon. The main advantage of this is that normal production would not have to completely stop. The basic design would incorporate 4 steps:

1. Idle DMF activity.
2. Compress the databases.
3. Install the compressed databases.
4. Resume normal processing.

This is currently under investigation.

6 Conclusion

Disaster recovery capabilities are never fully appreciated until disaster strikes. With a good recovery plan and well planned steps for recovery, the pain and damage can be minimized. With high performance computing, every hour of down time is an hour per processor of lost computing. Also, lost data means lost research. Every computing center needs to plan for a disaster, and hopes it never comes...