

CD/SCF/DMS/DMD

Enstore SFA Migration Phase 2 Design Document

Alex Kulyavtsev

7/15/14
Version 1.1

Table of Contents

Enstore SFA Migration Phase 2 Design Document	1
Table of Contents	2
Introduction	3
Background information	3
Migration Operation	3
SFA Operation	4
Hardware	4
Proposed Design	4
Command Line Arguments	4
Migration of Tape	4
<i>Migration of Individual Packaged Files from Command line</i>	<i>5</i>
Keep Track of Space usage on the Local Disk.....	5
Writing Files to Tape.....	5
Metadata Swapping	6
<i>After the Write to Tape.....</i>	<i>6</i>
<i>File Restore</i>	<i>6</i>
Helper functions.....	6
Final Scan	6
Notes	7
Changes to SFA	7
Change Name for Migrated Package in SFA	7
Flush File Family from SFA Write Cache.	7
Changes in Tape Migration Procedures	7
Setup Proper File Aggregation Policy	7
Migration Validation.....	7
Acronyms	8
References.....	8
Revision History.....	8

Introduction

Small File Aggregation feature (SFA) of Enstore tape system provides transparent aggregation of small files into larger containers (packages) written to tapes. Migration and Duplication features of Enstore provide capabilities to migrate data from old generation of tapes to the newer technology tapes or create duplicate (mirror) copy of the files in different tape library for better reliability.

The migration and duplication in the initial implementation of SFA is performed by migration and duplication of physical files on tape 'as is' and proper modification of file metadata in enstore DB. This allows migrating file packages to new tape technology generation. We have large amounts of data written to tapes (LTO4) before SFA was introduced. Some datasets have "small" files yet unpackaged that need to be *packaged*. For the files written with and packaged by SFA the new tape technologies may offer better performance if the physical file (package) written to the tape is larger. To enable file *repackaging* in SFA migration process shall unwind packages and write them to the new tape using new policy.

The phase two of SFA Migration and Duplication will allow to *package* small files and/or to *repackage* packaged SFA files according policy during migration process.

We do not change duplication to repackage files: duplicated files stay packaged the same way both on original and copy tape.

Background information

Migration Operation

In current implementation Migration process reads files from source tape(s) and accumulates files on migration spool area on local disk. There are several (typically three) reader threads to provide parallelism in accessing and checking file metadata while some other process is reading data from tape. The other few (usually three) threads write data to tape from disk spool area, one actually writes and the other two performing metadata operations or waiting for IO. The threads communicate through python queues. All migration activities run on the same node (migration station). Migration stations operate in parallel working on different set of tapes.

The migration process operates 'physical' files on tape: regular files and packages. Migration process automatically identifies if the file is a package. In this case it creates proper metadata in enstore DB tables for packaged files on new tape. File Clerk performs metadata swap operation at once for all files in the package. This metadata 'swapping' happens after new file is successfully written to the tape. The 'restore' operation on migrated volume by administrator command reverses the result of migration by undeleting migrated files and by swapping bfid's referenced in

pnfs back to migrated files. The log of ongoing migration operation and history are kept in DB.

SFA Operation

Files are written to SFA using encp as usual. If proper conditions are met (“enable-redirection” flag set, file satisfies SFA policy), the file will be written by disk mover to the disk on SFA system, then packaged to tar files and tar files are written to the tape. When reading packaged files with encp, the container file is read to SFA disk, unpackaged and the small file is delivered by disk mover when encp is called.

Hardware

Few most modern migration stations have Intel X3360 @2.83GHz 4 core CPU, 8GB memory, 10GbE connection. Migration spool area resides on Nexsan appliance formatted with two 4.5 TB arrays connected through two 8Gbit FC. There is also 367GB SSD currently used as migration spool area.

The current generation of tapes allows transferring data at speeds 240 MB/sec (T10KC, ~5 TB) or 252 MB/sec (T10KD, 8 TB). We use IO rate=70 MB/sec to estimate “typical” IO rate in enstore to/from LTO-4 800 GB tape.

Proposed Design

Considering our current hardware configuration we suggest to unpack files to the local migration spool disk then write files to enstore through SFA facility.

Command Line Arguments

Add command line arguments:

--sfa-repackage enable file repackaging.
 Default: migration by package (no repackaging).

Migration of Tape

The files are read as usual from unpackaged tapes to the local migration spool area. Changes of the code are required when reading tapes containing packaged files when doing migration with repackaging.

During the tape migration of SFA files the function `migrate_volume()` will access the full list of files on tape to include packaged files (instead of just physical files) and read files through SFA which will unpack files and deliver them through encp. This is the simplest change but unpacking full package files on SFA servers can create unneeded load on SFA servers, flash file from SFA caches and it can be slow, as the chain of communication happens to pull small files through SFA.

To provide read speed optimization and reduce per-file delays we may unwind packages on local disk on migration station. In this case `migrate_volume()` will provide list of physical files on tape as it does now. After the file is read in by `copy_files()` and is identified as a package file, then we will unwind (e.g. `untar`) the package to the local migration spool disk. To increase tape IO speed, package files can be read to local SSD drive if Migration Spool is configured

so. The package files will be read in separate subdirectory in Migration Spool, and symbolic link can be set to the directory on SSD.

The further optimization will be to create named pipe and read SFA package with `encp` to the named pipe. The named pipe is needed to trick `encp` as it requires destination to be a file. The unpacker (`untar`) will read tar file from the named pipe and unpack archive. This eliminates two disk IO operations out of three: all unpacking done on flight in memory and only resultant small files are written to the disk. It is possible to select only undeleted files by generating list of undeleted files in package `file.list` and then pointing to it by specifying tar argument “`--files-from file.list`”. When command line option “`—with-undeleted`” specified all files need to be unpackaged. As deleted files are written to the separate stream. The `encp` need to be modified to do extra check whether the destination file is a named pipe. Now `encp` fails if the destination file exists.

Migration of Individual Packaged Files from Command line

It is assumed that migration of the single file by specifying BFID or file name on command line or the file with the list of files is used to “fix” individual files and is not used for large dataset migrations. The reading (and unpacking) of individual packaged files to the local disk is performed through SFA facility to migration spool area.

Keep Track of Space usage on the Local Disk

The present implementation of migration uses few (default three) read threads and few (default three) write threads to overlap metadata checks and data IO. There is a ‘feature’ in migration implementation that migration code does not check for available disk space on migration spool. We observed situation when files were not removed from spool after the crash and the reader started to fail due to the lack of available space. The individual files were failed with error and reader rapidly went through the full list.

This issue will be exaggerated in migration with unpackaging as we may want to accumulate some amount of files before writing them into SFA to optimize writes and minimize tape mounts.

The function `copy_files()` will be modified to check if space is available on local disk and wait if needed before scheduling read `encp` transfer. When space is released in migration spool after files written to tape and deleted, the suspended read transfers will resume.

Writing Files to Tape

Add flag ‘`—enable-redirection`’ into arguments of `encp` call in function `write_file()` to write data through SFA when migrating files with repackaging.

Metadata Swapping

After the Write to Tape

The function `swap_metadata()` is called after file written to the destination tape to reassign properly cross references in the `pnfs` namespace and `enstore` File Clerk DB so `pnfs` points to the new `bfid` and `enstore` record points back to the original `pnfs` record. Currently we use optimized call `swap_package()` to File Clerk to reassign all constituent files in the package to the new package at once. This will not work for migration with packaging/repackaging when original files are not packaged initially, or packaged source files can be repackaged to separate packages. The migration process with packaging/repackaging will swap metadata for each constituent file individually as we do in `make_failed_copies()`.

We do not swap package metadata anymore for packages when repackaging files.

We may swap file metadata as soon as file written to SFA. We assume that SFA will take care of delivering file to tape. It is preferable to run `scan` as we do now as a separate step. The validation script (not part of the migration) can be amended to check the file was actually written to tape before declaring migration finished.

File Restore

The Restore operation reverses the result of metadata swap thus it operates in opposite way. Migration will swap metadata in function `restore_file()` for constituent files individually instead of block operation in File Clerk for all files in the package.

Helper functions

Helper function `_is_migrated_state()` need to be modified to make functions calling it correctly report status of constituent file (`is_migrated()`, `is_checked()`, etc.). Specifically, it shall report correctly status of packaged file migrated with packaged based migration (dereference information for the parent package).

Final Scan

The final scan for the volume calls function `scan_file()` to read file from tape to `/dev/null`. At present we read physical files only (unpackaged and package files) but we do not attempt to unpack packages into individual files.

Option a) Read package file only to `/dev/null` as we do now. For migration with repackaging we can miss situation when we cannot extract some file from package.

Option b) Read each constituent file through SFA to `/dev/null`. This will put substantial load on SFA, but this will ensure the file can be unpackaged.

Option c) Read package file on migration station and do listing of the package file with “`tar tvf`” to make sure all files are present in the package. Tar “`seek`” option shall not be used. The function `scan_file()` need to be modified to create named pipe, run `tar` command reading package from the named pipe and marking constituent files on destination package as scanned.

When files are written to the tape with SFA we need to make sure:

- All files are delivered to the tape before attempting the scan. This will need to implement SFA feature to flush file family from write cache.
- Files are read from the tape but not from some cache. This may require flushing files from read SFA cache.

Option (c) seems to be simpler reasonable compromise between (a) and (b).

Notes

When writing files with SFA some files may be packaged and some files are not. Accumulating files in SFA cache takes some time and thus the order of the files on tape will change substantially for some files.

Migration code is overdue for refactoring.

Changes to SFA

Change Name for Migrated Package in SFA

SFA uses file family name as component of path for the package in /pnfs . Migrated files are written using file family name for destination "*file_family*-MIGRATION" thus the word "MIGRATION" is present in the file path. SFA shall trim the word MIGRATION at the end of file family name when creating the package.

Flush File Family from SFA Write Cache.

SFA may need to implement 'flush files' request to stimulate pushing files for specific file family to the tape without waiting for timeout expiration. We want to be sure before the final validation step that all files have reached the tape. When writing one or several tapes, the last batch of small files submitted for aggregation typically does not immediately trigger SFA policy to aggregate and write data to a tape by exceeding threshold for the file count or package size. Files stay in cache while SFA waits until timeout in policy expiries, for example 24 hours. When we do migration we know when we done with particular file family and we can trigger writing to the tape the last portion of files right away to avoid extra wait before validation step.

Changes in Tape Migration Procedures

Setup Proper File Aggregation Policy

Add to migration instructions: the proper SFA Policy Engine (PE) policy shall be set by administrators before starting migration to define extra policy for the file family *file_family* -MIGRATION when migrating files in enstore file family *file_family*.

Migration Validation

SSA performs migration validation using custom scripts. The modification of these scripts is out of scope of this document.

Acronyms

- SFA Small File Aggregation. Enstore feature to write/read small files to tape with transparent packaging/unpackaging.
- PE SFA Policy Engine

References

- [1] Enstore Technical Design Document,
<http://www-ccf.fnal.gov/enstore/design.html>
- [2] Fermilab DocDB, CS Document 5035-v2, Enstore Administrator's Guide
- [3] Enstore Small File Aggregation HLD,
<https://cdcv.s.fnal.gov/redmine/documents/58>
- [4] Fermilab DocDB, CS--doc--4698, Enstore Small File Aggregation Feature User Documentation

Revision History

Document ID	Version	Date	Author	Comments
	1.0	6/25/14	Alex Kulyavtsev	Initial revision
	1.1	7/15/14	Alex Kulyavtsev	Per comments from JH