SDSC Enhancements to NSL UniTree

Wayne Schroeder, San Diego Supercomputer Center, *San Diego, California*

ABSTRACT: *SDSC has developed enhancements to NSL UniTree 2.0 to improve reliability, recoverability, and functionality and is now running it as our production archival storage system. These enhancements include the integration of a new NameServer (1.8, from Livermore Computing), the development of a transaction journaling system (a library, NameServer transaction logging, and NameServer database reconstruction utility), the extension of a DataTree to UniTree index conversion utility, and the integration and enhancement of a passwordless FTP user interface system. This paper will describe these developments and our conversion from DataTree to UniTree.*

Introduction

SDSC has been very actively involved in archival storage systems since the creation of the center in 1985. An SDSC spin-off, DISCOS, a division of our parent company General Atomics, marketed the Los Alamos National Laboratory archival storage system Central File System (CFS) as "DataTree" and later the archival storage system from the Lawrence Livermore National Laboratory as "UniTree". The rights to UniTree were sold to OpenVision Technologies about two years ago and has recently (late 1994) been resold to UniTree Incorporated, a consortium of five companies, four of which are European.

We have been interested in running the National Storage Lab (NSL) version of UniTree for several years now. Until recently though, we felt that due to the experimental nature of NSL and its rapid development, it was not stable or reliable enough for our environment.

A little more than a year ago, we decided that using NSL 2.0 with SDSC extensions and integrating a new NameServer would meet our current needs and mesh well with our future plans.

The core development team was lead by Joe Lopez and consisted of four people. The team members were drawn from other groups and had additional responsibilities on other projects. Larry Diegel selected and integrated the hardware, developed the journaling library, and installed and enhanced the user interface software (uftp retry logic and local uftp inhibition under operator control). Tom Sherwin installed and enhanced NSL UniTree, helped integrate the NameServer, fixed various UniTree and AIX problems, enhanced the DataTree to UniTree index conversion software, helped with uftp inhibit, and developed the tape positioning extensions. This author developed new communication software to help integrate the new NameServer, added journaling calls to the NameServer, and developed the reconstruction, UniTree backup systems and intradatabase capability comparison software. Antoinette Long fixed various UniTree problems and assisted with AIX and UniTree troubleshooting. For a few months a student intern, Sharon Fenner, assisted with various tasks. George Kremenek assisted for a few weeks by implementing exit statuses in uftp and the single execute line extensions.

SDSC's migration to NSL UniTree from DataTree benefits our user community in a number of ways. These include:

Performance
: UniTree can store and retrieve files at many times the speed of our DataTree system. Future hardware upgrades (available for UniTree and not DataTree) will improve performance even more, perhaps an order of magnitude (HIPPI-attached peripherals). Support for third-party file transfers will allow us to achieve much higher transfer rates to archival storage.

Reliability
: The new UniTree RS6000 host platform, RAID disk, STK 4490 tape systems, and UniTree software (especially with local SDSC enhancements) should be an extremely reliable data archive. Our DataTree hardware was reaching end-of-life; several disk head crashes have occurred during the last couple of years.

Unix
: The interface to UniTree is standard FTP (with extensions) with UNIX-like commands. This interface is much easier for our new users to learn than was DataTree. Also the UniTree Server UNIX system is easier and less expensive for SDSC to operate.

Commonality
: UniTree is becoming a widely used standard at many supercomputing centers. This standard interface to archival systems will improve ease-of-use and collaboration with other centers.

Future growth

Perhaps most importantly, UniTree allows us to make some exciting and valuable future expansions in the size and functionality of our archival system. High performance data storage, access, and management is a major resource to our user community. Additionally, the system architecture of UniTree will allow us to produce new tools to access and analyze data in interesting new ways.

SDSC Enhancements

New NameServer Integration

A new NameServer was developed by Donna Mecozzi and Jim Minton of the Lawrence Livermore National Laboratory in 1993 for their version of UniTree. The NameServer records and manages user-defined names for files and directories. It converts user names to UniTree identifiers (Capabilities), maintains directory attributes, and checks access permissions.

The former NameServer used a btree structure to manage the user-defined name space, which was efficient in accessing information, but required substantial data reorganization when new information was added. It was felt that a new NameServer could provide greater reliability.

The new NameServer was developed for Livermore's storage system (a parent version of UniTree) and communicated via UDP Remote Procedure Calls (RPCs) as defined for UniTree clients in *libunix*. NSL UniTree uses TCP RPCs as defined in *libnsl*. It was felt that the cleanest way to integrate the new NameServer into NSL UniTree would be to change it to use TCP RPCs.

This required the development of a new set of communication routines for the NameServer which was essentially a blend of the UDP RPC code from the new NameServer and NSL TCP RPCs from the old. This involved an analysis of the arguments passed at various levels and their use, and the development, testing and debugging of the new blended code.

Since the lower-level routines in the new NameServer checks permissions in a different way than the old NameServer, we made changes to handle group permissions correctly. We added logic in the new interface routines to retry with other group ids from the list of groups the user belongs to, if a permissions error occurs.

The new NameServer does not keep names in alphabetical order like the old one did, so we added sorting logic. We did this not in the NameServer, but in the utilities that produce lists for users (utilities run for users for 'ls' and 'dir' commands).

Our version of the NameServer has both TCP RPC and UDP RPC communications (the latter kept for system administration tools). There were a number of problems that we resolved in the underlying support routines to allow these to operate at the same time.

Transaction Journaling

Transaction Journaling is an important feature of DataTree that was lacking in UniTree. In DataTree, for every transaction that causes an update to the directory[1] a transaction record is written that completely describes the action. If need be, these transactions can be reapplied to a directory backup to recreate its contents.

A backup of the directory is made periodically. If a hardware or software problem causes the directory to become corrupted, the backup image can be copied back in and the transactions reapplied to bring the directory back into agreement with the files stored on tape and disk.

A few times in our history, SDSC has used the transaction journaling recovery feature of DataTree to eliminate or reduce the number of files lost following a serious hardware or software problem.

Our transaction journaling project divided into three sub-projects: developing a Transaction Journaling Library, using this library in the NameServer, and developing a reconstruction utility.

We chose to journal NameServer transactions since some sites had experienced problems in the old NameServer database. Transaction journaling would have allowed them to recover gracefully.

Transaction Journaling Library

We decided to make the transaction journaling library generic so that it could be easily reused.

The transaction journaling library was developed with the following goals and assumptions:

- The interface should be generic enough to be used anywhere within UniTree, in any server, and at any level.
- Multiple journals may be generated within a server. For example, the merged tape/disk mover may generate a separate journal for the tape mover and the disk mover.
- Journal file names will include a server specified path and file name including a date/time stamp of when the journal was created: <dirpath>/<name>.<YYMMDD>_<hhmmss>_<TZ>
- In order to minimize the disk I/O overhead, each journal entry is written as a multiple of 512 byte blocks. The smallest journal entry requires 512 bytes to log.
- A journal entry is written in one write. Possible problems occur when multiple writes have to be executed as one atomic event. It's much simpler and more foolproof to do it all in one write.
- Since it will be common to journal various input and output arguments, the journal routine has an argument to specify an array of **iovec** structures pointing to multiple source buffers. This is a speed optimization to prevent the unnecessary copying of the data before calling the journaler.

1. "Directory" refers to the information that describes the files stored in the archive: the names, locations, etc. This is also commonly called the "index" or "meta-data."

The contents of the input and output arguments are totally the responsibility of the server. The journaler simply records the bytes as requested; it has no concept of their meaning, structure or value. It is the responsibility of the UniTree server, not the journaler, to include enough information during logging to make sense of the journal entries when they are read back during a database reconstruction.

The interrecord information is recorded by the journaling library, including time stamp, record number, record size, etc.

Additional features are planned but are currently unimplemented. These include:

- The journaler should automatically break journals into reasonable sized chunks to facilitate handling based on maximum size, number of transactions, and the age of the journal. Instead, we restart the NameServer each night during the UniTree directory backup and periodically **rcp** the current file to another host.
- UniTree asynchronous disk I/O and multiple journal entry buffers will be used to reduce the impact of the journaler's disk I/O on UniTree. This disk I/O should avoid all system caches. This may be a performance improvement in the future, but for now synchronous I/O seems adequate.
- A second journal area will be specified for use in case the primary area fills up.

Using the Transaction Journaling Library in the NameServer

We added calls from the NameServer to the transaction journaling library to record information for every successful TCP RPC call that updates the NameServer directory. This includes FuncChange (change entry attributes), FuncSymLink (create a symbolic link), FuncInsert (insert a file or directory into a directory), FuncReplace (replace a directory), FuncDelete (delete an entry), FuncMkdir (make directory), FuncMvdir (move directory), FuncDTInsert (insert a DataTree file, SDSC extension). The information recorded includes the various common input arguments, function-specific input arguments, and results. Six to eight separate values are recorded via the I/O vector. Each of these fits in the basic 512 byte transaction record.

These calls were added in the TCP RPC communication routines. Although the UDP RPCs may also update the database, they are only used by UniTree administrators and so can be reproduced by hand.

The RPC result structure is recorded since it includes the created capabilities which are needed for reconstruction.

Reconstruction Utility

The reconstruction utility applies journal entries to a NameServer directory. Using a backup of the NameServer directory, we can apply journaled transactions and recreate the directory.

It uses most of the NameServer routines to handle reading, writing and caching the directory, and processing function calls. But instead of registering for RPC communications, it reads transactions from a set of transaction journal files and (re)calls the underlying NameServer routines to perform the function.

There are options to list various attributes of the transactions and/or perform the actions, and/or process only a portion of the transaction journal file.

A difficulty exists because each time a Capability is created, it is unique (using timestamps as part of the seed). This means that as we reapply functions that create Capabilities, the new Capabilities will not be the same as the ones created last time. Since many functions supply a Capability to identify an object to operate on, a subsequent function on a Capability created via the journal would fail as it would not be found.

To solve this problem, the reconstruction utility keeps a table of old and new Capabilities. For make_dir, for example, the journal records the Capability created when this function was first performed. The reconstruction utility keeps track of the Capability created the first time and the one created this time. For subsequent transactions, the old Capability is replaced with the new before the NameServer function is performed. Thus a make_dir, followed by a change (e.g. change group), will operate on the correct object.

We have tested this fairly extensively and believe it is operating correctly. We have reproduced a NameServer directory from backup and about a day's worth of journals and it compares correctly. To do this, we wrote a compare utility that could ignore irrelevant fields (such as timestamps). We have not, however, had to use the reconstruction utility to correct the Directory (fortunately).

UniTree Backup

We take UniTree down nightly to create a full backup. This is done by a root cron job.

The backup script first checks that UniTree is running normally. If not, the backup is skipped for the night. It then installs a banner file that is checked by our uftpd routines, which displays a notice and prevents new ftp sessions from starting. The uftp script (on the CRAY for example), for batch command-line invocation, will periodically retry in this situation. The backup script then waits for the running UniTree sessions (uftpd) to complete, giving them up to 15 minutes, after which it kills them. This effectively quiesces UniTree. It then runs 'backup' to cause the TapeServer to backup its Directory to tape, and copies the NameServer Directory to a file. After the TapeServer backup completes (usually 30 to 45 minutes), it removes the banner file to allow UniTree access to resume.

It then stores the copy of the NameServer directory into UniTree, creates a listing of this UniTree directory (a new directory for each month), and sends electronic mail to the administrator. This listing is created with an added option that lists file names with their Capabilities as hex strings (Capabilities are 4 64-bit words long). As long as the TapeServer Directory is intact, we can retrieve these files via their Capability, with little else of UniTree running. To our knowledge, the TapeServer has been very reliable.

Once a week, we also backup the LocationServer directory and the TapeServer headers into UniTree. The LocationServer

directory is less critical since if it is somewhat out-of-sync it will be corrected during normal operations (when people access files). The TapeServer headers are stored in case we wish to analyze the information over a long period. The normal TapeServer backups are saved round-robin within a set of tapes and so do not extend far into the past.

NameServer journal files are also stored into UniTree. A new journal file is created each time the NameServer restarts (we restart it during backup so this is normally at least once a day). A cron job runs every few minutes and if there is more than one journal file, it stores the previous one into UniTree, moves it temporarily to another directory and electronically mails the Capability off-system.

We also have a cron job that periodically **rcp**'s the current journal file to another system (the CRAY). This way, even with a bad disk problem, we should have a journal that is nearly up to date.

DataTree to UniTree Index Conversion

The conversion tools, initially created by NCSA and rewritten by SDSC, convert a DataTree database into a form that can be inserted into the UniTree database. SDSC rewrote the routines to deal with our (more current) version of DataTree (multivolume files, mixed case names, file sizes in bytes, etc.) and with NSL UniTree (communicating via *libnsl*), adding logging, restartability, more error checking, configurability, and the ability to access all fields in the DataTree structure.

There are several sections of the conversion tools: the directory parser (dtparse), the directory/file creator (dtcreate), and special library and server codes to allow DataTree files to be processed inside of UniTree.

DTParse is the main decision maker of the conversion process. This program reads the DataTree file database and converts the information into a file usable by the **dtcreate** program. DTParse is responsible for determining who owns a given DataTree file, determining the location in the UniTree file database where the file belongs, and flagging any peculiar entries that may need more attention. Files may need more attention when they have Access Control Lists (ACL's) or when passwords have been assigned, or when the user does not exist in the input password file. Once DTParse has completed, the output file can be fed into the DTCreate tool to have the entries actually created.

DTCreate reads the output generated with the DTParse tool and sends requests to the TapeServer and the NameServer to create files and directories.

Modifications were made to the TapeServer to create a back door for the DataTree file information to be sent directly into the UniTree tape server. Modifications were also done to enable UniTree to read the DataTree tape formats.

We ran the conversion over the Thanksgiving holiday, over a period of six days. During this time both DataTree and UniTree were unavailable to users. Users were prepared for this period and kept the CRAY and Intel supercomputers fairly well utilized with less data-intensive applications.

Passwordless FTP

We chose 'dftp', from DISCOS/Livermore, as a basis for a high-performance, reliable and secure interface. FTP provides higher-performance and greater reliability than NFS access to UniTree. 'dftp' has a passwordless interface which improves security by allowing users to access UniTree from trusted hosts without putting passwords into scripts or .netrc files. We made a series of extensions to 'dftp' and installed it as 'uftp.exe' (unitree ftp).

One extension allows users to stack commands on the execute line, to provide a single command line interface. Commands that are separated by commas on the execute line are executed in sequence. When all are completed, uftp exits. 'uftp' with no execute command line goes into interactive mode.

We also made changes such that the exit status from uftp returns success or failure and the type of failure. Thus with a single execute line invocation, the user script can easily determine if a transfer completed successfully.

We added a 'uftp' script between the user and the uftp program that users run to access uftp.exe. It checks the exit status and for transitory errors on single execute-line invocations, will retry the operation periodically (less frequently as time goes by).

uftp.exe and uftpd also check for banner files and if they exist they are displayed and the script sleeps. Banner files are used to hold off access during UniTree directory backups or other periods of downtime, either locally (from one host), or globally.

The script also checks for many environment variables that control various options. These include the default server host and port, verbose and debug mode, store and get unique (i.e. create a new name if there is a name collision), hash size, CRAY time-limit, etc.

To make UniTree somewhat more secure, we extended uftpd to access different passwd files for connections from different hosts. We maintain separate user/passwd lists for the CRAY, Paragon, workstations, and Macintoshes. This restricts the set of allowable user logins to those defined on each host and restricts access to the defined hosts. User accounts are set up on the various platforms and periodically installed on UniTree.

To speed up 'dir' commands, we implemented a password caching system. Since we have a fairly large number of UniTree users (7,200+), the searches through the passwd file for user information can take a while (on the order of a second). When users do long directory listings (dir commands), uftpd converts each User ID to a name. So 'dir' commands on long directories, especially for users near the end of the file, could take a long time. To correct this, password information is cached. Since most accesses are usually the same one over and over, we very significantly improved the performance. This is done via a shared-memory password file indexed by UID. The conversion of group ids to group names is also done this way.

We have also modified uftpd to record status information on top of the argument information accessed by **ps**. This way, we can easily determine the state of various uftpd sessions:

```
hal-6000> ps -ef | grep uftpd
  [...] uftpd (u13431) Transfer (retrieve)
    ok.
  [...] uftpd (u9212) In retrvfile
  [...] uftpd (kremenek) In store
```

Hardware Configuration

We are running NSL UniTree on a IBM RS/6000 Model 99J with 512 MB of memory. To this we have attached approximately 150 GB of disk, and two STK 4791 Tape Controllers (with ICRC Compression Firmware) which attach to two StorageTek Silos (each containing 6,000 tapes). The disk consists of 110 GB of RAID disk (five RAID-3 disk arrays) attached via 20 MB/s differential SCSI, and 41.8 GB of disk attached via single-ended SCSI (10 MB/s). The 110 GB RAID is used as UniTree disk cache and for UniTree Directories (UniTree Servers maintain dual copies of the Directories and we keep one on RAID). Each of the two STK silos contains four STK 4790 tape drives and handle the extended length 36 track 3490 tapes. This gives us a capacity of about 1.6 GB per tape and about 9.6 TB per silo.

Our current archive consists of approximately 8 TB in 1.45 million files contained in 38,000 3480 DataTree tapes and 4,000 3490 UniTree tapes.

Current Status/Transition

As one would expect with a software system of this size and with the changes we made, there were a number of small and larger problems to deal with as we moved into production. Some were remaining problems with our NameServer integration, some were NSL UniTree bugs, and some AIX problems. We fixed most of these fairly quickly and over several months created additional integrity checking software, making the system more reliable.

Our most serious problem to date was a situation in which tape access would abort (timeout) when we had many tapes running. IBM and StorageTek staff worked diligently and effectively with us to resolve this problem. The main problem was that tape reads were taking a long time due to the controller having to decompress data as it was searching for a tape mark for a particular file. Although the controller is multiplexed, the decompression is a compute-intensive task and only one read or write can occur at the same time. 'Locate' functions can occur simultaneously on each drive.

The immediate work around was to add a second controller and increase the timeout values. We then modified UniTree so that we can 'locate' to a tape position before forwarding to a tape mark. This does not require data decompression, multiplexes very well, and very significantly speeds up our tape access.

We also wrote additional software to compare the file Capabilities known to the NameServer, DiskServer, and TapeServer. This included commands to produce text lists of the Capabilities, and a simple database utility to efficiently add and remove these text strings to an internal database so that we could compare 1.4+ million entries. It hashes the beginning of the Capabilities and builds a link list. Using this software, we discovered additional files that were known to the NameServer but were not present in the Disk or Tape Server databases. These appear to have occurred in our early production period during initial instability.

Future

Reliable, high performance archival storage is a key component of SDSC's future. As always, there is a great need to store and retrieve massive amounts of data from the simulations running on supercomputers such as the CRAY and Intel Paragon.

There are also new efforts under way to provide database access to archival storage information, to access information via queries instead of file and path names, to access portions of files, and in other ways improve the data and information management systems available to the scientist.

With NSL UniTree, we expect to be able to continue to improve performance and reliability and collaborate in the development of some of these new tools.

We are in the process of 'kerberizing' UniTree as part of a general Kerberos project at SDSC. Using the MIT Kerberos 5 Beta 3 release we plan to Kerberize our CRAY, Paragon, workstation, Macintosh and archival storage environments.

We hope to be able to acquire HIPPI-attached RAID disk this year and, by using this key feature of NSL UniTree, greatly increase the host to storage system transfer rate. We may need to restrict its use in some way, though, as it vastly improves the CRAY to/from archive disk transfer speeds without correspondingly increasing the archive disk to/from tape speeds.

For the longer term, we intend to migrate to HPSS, the High Performance Storage System under development by Lawrence Livermore National Laboratory, IBM Federal, National Energy Research Supercomputer Center, Los Alamos National Laboratory, Oak Ridge National Laboratory, Sandia National Laboratory, and others. We will install a test version of HPSS at SDSC and are collaborating on migration from UniTree to HPSS, and the development of Database interfaces to HPSS.

Over the next few years, we hope to provide distributed TeraFlop supercomputing systems (via distributed systems with other centers) and PetaByte archival storage systems to researchers who are tackling critical grand-challenge problems of national and global interest.

Acknowledgments

This work was funded in part by National Science Foundation Cooperative Agreement ASC-8902825.