HPTFS: A High Performance Tape File System

Xianbo Zhang, David Du
Digital Technology Center Intelligent Storage Consortium (DISC)
University of Minnesota, MN, USA
{*xzhang,du*}*@cs.umn.edu*

Jim Hughes, Ravi Kavuri
Sun Microsystems
One StorageTek Drive
Louisville, CO, USA
{*James.Hughes, Ravi.Kavuri*}*@Sun.COM*

Abstract

The exponential data growth and new data retention regulations have created a higher demand for tape storage systems since the tape technology provides long lifetime archival with low cost per storage unit [1, 2, 3]. However, tape storage is hard to use due to its SCSI interface and sequential access nature. The purpose of this project is to provide programmers and end users an easy to use interface while taking the full advantage of high capacity and high streaming speed from modern tape technologies. We have built a software, called HPTFS (High Performance Tape File System), that mounts a tape as a file system and enables tape write sharing among multiple users by transparently interleaving user data streams to tape. This systems is simple enough that it could be directly built into a tape drive thus completely changing the paradigm on how tape is accessed for the fist time since tape itself was introduced. As a secondary feature, the software can also support the concept of continuous data protection by simultaneously writing data to both disk and tape providing a seamless support of data mirroring for any user application. We have evaluated the basic system performance of HPTFS and presented some ideas of further improvements.

1. Introduction

The demand for data storage has been dramatically increased due to the explosive growth of data [11, 12], especially the growth of fixed-content data [13] driven by businesses of all sizes along with government mandates for data retention [7, 8, 9, 10]. Fixed-content data such as check images, CAT scans, video/audio files and email messages, do not change over time and have to be retained for a long period of time. These data may not be actively accessed, but should be available when needed. How to protect these data along with other mission-critical data for a long period of time is vital for the enterprise business continuance and may even be required by regulations [7, 8, 9, 10]. The main concerns for this explosive amount of data are storage cost and data protection. With current data storage technology, tape is still a good candidate or even the only choice for storing such huge amount of data with economy and proven long lifetime in mind [14, 21, 22]. Tape media provides long lifetime archival and convenient portability for offsite protection. Valuable backup data is required to be sent offsite for certain distances by regulations depending on the classification of the data.

Modern tape technology featuring growing storage capacity and growing streaming rate makes data backup and long-term data archival economical as well as efficient. Tape capacity is doubling every two years [14]. One S-AIT tape from Sony can store 500 GB native data [4]. With conservative compression rate of 2:1, today one tape can hold 1 TB data already. It is projected that tape media holding 15 TB native data is just several years away [14]. Tape drive native transfer speed has been increased from several MB per second to even 80MB per second [5] or 100MB per second [6] during last several years. With compression turned on, the tape drive write speed can outperform the hard disk write speed. As power and floor space becoming more of a premium, we must appreciate that a tape cartridge sitting in a tape library not being accessed consumes no energy and occupies less space than a disk in a RAID (Redundant Array of Independent Disks) or a MAID (Massive Arrays of Idle Disks) [24].

However, tape technology faces its own problems. The tape drive can be slower than expected if it is not fed with enough data due to the overhead of tape drive repositioning. A tape drive cannot be easily shared among users. The tape drive is not programmer-friendly due to its SCSI interface. Currently tapes are accessed by backup software that is commonly managed by system administrators with special knowledge of tapes and backup/restore/archive techniques. Application programmers are scared away from

tapes due to their totally different user interfaces from the traditional file systems.

Making the use of tape as easy as possible while taking the full advantage of its high speed and high capacity is the main goal of this tape file system project initiated by Sun Microsystems and DTC (Digital Technology Center) Intelligent Storage Consortium (DISC) at the University of Minnesota. We believe a generic file system interface to tape storage would be a significant improvement to the usability of tape as a shared media. A file system paradigm will be valuable for the broader use of tape storage since more users are familiar with file systems and many applications write and read data through standard file system APIs. Such a file system can hide the difference between accessing a tape and accessing a disk. With the same convenience, users/user applications can easily put data on disk or tape based on data access patterns without knowing tape SCSI interfaces. This provides conveniences to achieve a better usage of tape storage and disk storage. This new tape file system paradigm brings convenience to personal users, simplifies data backup software development, and enables new features for tape-resident data: indexing, content based searching, data sharing with better request scheduling, easy remote access, file system based data protection, and faster disaster recovery.

For the rest of the paper, the related work is presented in Section 2. The system design and implementation are introduced in Section 3. Section 4 presents the tape data layout that was chosen for the system. Other design issues are further discussed in Section 5. System performance evaluation and some conclusions are presented in Section 6 and Section 7 respectively.

2. Related work

Robotic tape libraries are commonly used by applications like data warehousing, digital library, digital image, multimedia and scientific computing due to their requirement of huge storage capacity. The rapidly growing demand for storing fixed-content data opens a greater opportunity window for high capacity and high performance tape drives. The issue of how to federate tertiary storage (mainly tape storage) into a complete storage system attracts many researchers. Cost-effectiveness, performance-efficiency and storage usability are the main goals for this federation. In this section, we present a number of existing ways of using tape library that include data backup software, Hierarchical Storage Management system (HSM), Virtual Tape Library (VTL), log-structured organization, and data buffering and interleaving.

2.1. Data backup software

There are many backup software programs that exist in the world [15, 30]. Designed and used properly a backup software can move data back and forth between disk and tape providing the functionality of data backup, archive and restore. There exist a set of basic software for data backup. With basic software, users need to write scripts to automate the backup process based on desired backup/restore policy. There are also a set of commercially available software for data backup. With commercial software, many backup functions are available with license and support fees. Commercial software usually provides automatic device discovery, volume configuration and automated backup/archive/restore for handling a number of complex situations. The following sub-sections give a brief introduction to both basic and commercial backup software. The main purpose is to give the user an idea how things are working in the backup application.

2.1.1. Basic software The most frequently used free software to manipulate tapes is *mt, dd, tar/gnutar, cpio/gnucpio* [30] and *KDat* [36]. *mt* is used to move tape forward or backward so that a tape drive can be positioned to a specific location. The others are used to transfer data from disks to tapes or tapes to disks with limited data management. *dd* reads/writes a single file from/to tapes, stripping the file of its name, directory location, date of creation, etc, and the user is required to maintain documentation describing how files are written to a tape. *tar/gnutar* reads/writes multiple files and directories while preserving the directory structure, file names, etc. *tar* can easily archive files across machines. *cpio/gnucpio* is similar to *tar/gnutar* and reads/writes multiple files and directories and it also preserves the directory structure. *cpio* can handle backing up and restoring device files (those in */dev*) which *tar* usually can not. *dd* and *cpio/gnucpio* provide error recovery in the event of tape failure while *tar/gnutar* does not. *dump* and *restore* are two commands that provide user a simple backup/restore solution. *dump* examines files in a file system, determines which need to be backed up, and copies those files to a specific storage medium. Subsequent incremental backups can then be layered on top of the full backup. The *restore* command performs the inverse function of *dump*. It can restore a full backup of a file system, single files and directory subtrees. *KDat* is a tar based tape archiver which is designed to work with multiple archives on a single tape with a graphical user interface. With a complex index on disk, *KDat* can quickly locate an individual file in the archive without reading the whole archive. Based on this basic software, the system administrator can create complex scripts to perform many backup jobs but this requires a good understanding of vari-

ous options of the software and the corresponding scripting languages.

2.1.2. Commercial software With a special software component, backup software can copy data from disks to tapes. If the data source is kept at the original location, the operation is called backup, otherwise the operation is called archive. The special component instructs the robot for tape switching and writes/reads data to/from tape using SCSI commands. Backup software from Symantec, CA, EMC, Microsoft, etc. manages resources such as tapes, tape drives and robots, and schedules backup jobs according to user predefined policies and resource availability. A backup operation is scheduled to run within a certain backup window in a 24×7 time frame. The backup system usually maintains a catalog to track the data location on tape media and disaster recovery information. Full backup together with incremental backup is the way to handle exploding data capacity requirements along with a shrinking backup window. The complex backup process is managed by system administrators with special training. Within such a system, users and applications do not have direct access to tape storage, and tape is treated as a second-class-citizen storage. User data are written to disks and copied from disks to tapes for protection.

2.2. Hierarchical storage management system

Hierarchical Storage Management system (HSM) is a policy-based management system for file backup and archiving that transparently moves data back and forth between storage hierarchies to meet user data requests while keeping down the storage cost. Typically a storage hierarchy is composed of primary memory, high performance hard disks, low performance hard disks and tape storage. Within such a hierarchical storage system, more frequently or recently accessed data or data requiring low latency are maintained at high performance media while less frequently accessed or less critical data are stored at tape media. By leveraging the composition of the storage hierarchy, a specific cost range and performance requirement can be met with proper data migration/duplication policies. A proper data classification, placement and transfer model across the hierarchical media can help achieve the desired performance. When user data is moved from disk to tape, a stub is left at the disk pointing to the location of the real data on lower level storage media (tape). With the data stub, migrated data still appears to be on disk. When an application accesses a file, which actually resides on a lower tier storage, the HSM system fetches the real file from the lower tier storage and caches it on the higher tier storage, and then the cached data is accessed by the application. The application does not see any semantic difference between accessing data from the lower tier storage and accessing data from the higher tier storage except for experiencing slower response when the requested data resides on a lower tier storage. However, the resulted longer response time may disappoint certain applications causing access failure when the access cannot be met within a specific time frame. HSM products are offered by Symantec, CA, IBM, and Sun Microsystems, etc. They are providing the same set of basic functionalities with different features and various performances.

2.3. Virtual tape libraries (VTL)

Due to the explosive data growth and economy globalization requiring 24×7 hours business, traditional backup/restore operations do not work very well any more [19]. To respond to the exploding data with shrinking backup/restore window size, disk based virtual tape library (VTL) is introduced during the last several years [20]. VTL technology transforms disk into virtual tape to improve backup/restore times. Data is backed up to VTL and then migrated to physical tape library for offsite protection or off-shelf archival. Our implementation offers advantage over the VTL solution in that we neither require the cost of disk for capturing the data nor the complexity of migrating and managing virtual and real tape cartridges.

2.4. Log-structured organization for tertiary storage

The append-only nature of tape data writing encourages people to apply log-structured file system (LFS) to manage tape data [16, 17, 18]. The Highlight and LTS projects apply log-structured file system techniques to manage data blocks on tertiary storage. By integrating LFS with tertiary storage and treating tertiary storage as a backing store, Highlight allows automatic migration of LFS file segments (containing user data, index nodes, and directory files) between storage hierarchies. LTS provides a uniform, random access, block-oriented interface to hide the details of tertiary storage. Using the techniques from LTS, Paradise [23] implements a scalable database system capable of storing and querying raster image data sets residing on tape. Without fetching data to disk, the query operation can be served directly by tertiary storage. Our approach distinguishes from these projects by providing a generic file system interface for data access instead of an application specific interface and by performing transparent data buffering and interleaving for maximum write performance.

2.5. Data buffering and interleaving

Slow data transfer rate to tape makes a tape drive head reposition frequently and the corresponding back and forth

operations for repositioning, usually called "shoe-shining", increases the stress on the tape drive and wears the tape media. Among all the techniques for matching the speed discrepancy between a host and a tape drive, volume buffering and data interleaving are the two most effective ways to reduce tape drive repositioning operations and achieve maximum write performance. When physical memory is expensive, data is buffered at staging disks and moved to tape afterwards. Virtual Tape Appliances buffer data which is later moved from local disks to a local tape drive so that the tape drive can operate at full speed. The other method, data interleaving or data multiplexing, is also used to stream multiple users to a single tape drive. Multiple slow transfer rates can be interleaved to be a high aggregate transfer rate to match the tape drive speed making the tape drive work in a streaming mode. Streaming occurs when the aggregated data transfer rate to or from the hosts closely matches the tape drive's data transfer rate, allowing the drive to read or write data in a continuous stream with full speed. The effect of data interleaving comes in two ways: during writing, it helps to maximize tape write speed and during reading, it reduces the effective read speed for a particular host which is beneficial when the consumption rate of the host is slower than the tape drive. Backup software such as BrightStor ARCserve Backup and Symantec NetBackup has data interleaving capability. A user may suffer from data interleaving due to its lengthened read pass for a specific read request. However, this drawback may be mitigated by intelligent data interleaving and parallel data retrieval. With proper scheduling, a tape drive may shorten its read pass and also serves multiple read requests with one pass. The effectiveness of request scheduling depends on the actual data requests pattern.

3. Design and implementation

We have designed a High Performance Tape File System (HPTFS) which seamlessly integrates tape storage to the client operating system with a traditional file system access interface. Based on the design, we have built a prototype file system. HPTFS has a series of desired features as described below and opens a new paradigm for the use of tape storage.

3.1. System features

We are exploring a new way to federate tape storage into the current hierarchical storage system. The designed system bears the following features:

- Providing a generic file system interface. Applications based on file system interface can write and read tape data without any modification. Various file transferring protocols such as cp, scp, sftp and http will be automatically supported. However, good performance for random access is not a target for such a sequential storage media. This limitation does not hurt fixed-content data or data archive too much. For these types of data, it is important to make sure that they are stored correctly with high speed and safety.

- Containing user data and corresponding metadata (including directory data) on the same tape. Individual tapes can be transported to another location and read without any additional pre-existing information being needed. These tapes can be considered completely self-describing. When backup software is built over such software, the catalog can be easily recovered from individual tapes as needed.

- Sharing tape media among multiple users. The system supports concurrent writes for maximum write performance. As shown in Section 6, our tape storage system can provide a consistent concurrent write performance.

- Supporting read while write. If data is configured to write to tape and disk simultaneously, read request would be served by disk instead of by tape. Read while write is an expensive operation for tape since it requires the tape drive to move back and forth resulting longer time delay. This feature can easily be used to achieve the concept of continuous data protection (CDP).

- Writing to or reading from tape directly without involving disks along the data path. The implemented system buffers read/write data using a small amount of main memory instead of disks. Using main memory as a data buffer not only improves tape I/O performance, but also reduces system cost and complexity.

3.2. System implementation

HPTFS, built over FUSE [29], mounts a tape as normal file system based data storage and provides file system interfaces directly to the application. It interprets a file system call as proper tape operations and returns results to the calling process. FUSE is a user-space file system framework that traps file system calls and upcalls user-space daemon process to perform actual file operations.

3.2.1. System architecture The system architecture of HPTFS, which is composed of fuse kernel module and hptfs user daemon process, is shown in Figure 1. In essence, every file system call is translated to tape operation calls and data/metadata is written to or read from tape media. Thus, user applications have a direct access to tape media using the same file API's as to disk.

As shown in Figure 1, VFS (Virtual File System) is a kernel software layer providing a common interface to all different file systems. It receives file system call requests from user level applications (e.g., open, stat, read, write) and interacts with fuse module if file operations are for FUSE mount point. Via fuse kernel module, system call requests are passed from kernel space to user space daemon process hptfs. hptfs calls user mode functions registered with *struct fuse_operations* to access data and metadata from SCSI tape media through an SCSI subsystem. The SCSI subsystem has a 3-level architecture with the "upper" level (*st.o*) being the closest to the user/kernel interface while the "lowest" level (hba) is closest to the hardware as shown in the right bottom part in Figure 1. The requested data and operation status are passed back to calling process through hptfs, fuse and VFS.

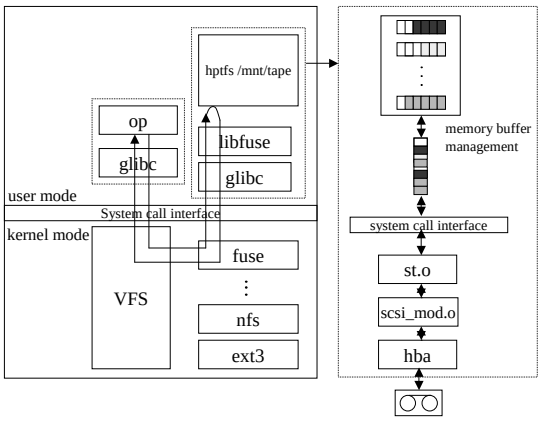

Figure 1. Architecture of HPTFS

In the following subsections, we briefly introduce the user mode system calls, tape device function calls, and data buffering and interleaving techniques implemented in HPTFS.

3.2.2. User mode file system calls Each user file on tape has a corresponding user data and metadata (will be discussed in Section 4) and operations on the user file data and metadata are completed by user mode file system calls. These user mode file system calls are transparent to user applications. As described in Section 3.2.1, a file system call made by a user application, such as open, read, write and close, is decomposed into a series of file system operations which are transferred to user mode and are completed by user mode file system calls. For example, an *open* operation call from a user application could be translated into *LOOKUP*, *MKNOD* and *OPEN* operations, of which *MKNOD* and *OPEN* are actually completed by $layer\_tfs\_mknod$ and $layer\_tfs\_open$ user mode system calls. Part of the implemented user mode system calls are listed in Table 1 for a better understanding of how the system works. The combination of these user mode system calls supports tape data read/write/append and corresponding metadata operations.

3.2.3. Tape device function calls Tape is a sequential access media and the data of a file is stored as blocks on tape. Block marks exist between blocks and file marks exist between files. At the end of the tape, there is an end mark and data cannot be written beyond the end mark. Data location information in terms of file number and block number has to be managed for data store and retrieve. Tape drive can be positioned to a specified block within a specified file through magnetic tape I/O interface. Tape drive can start to write or read from where it is positioned. The Tape interface under Linux environment is defined in /usr/src/linux/include/linux/mtio.h. Tape drive can be controlled to move forward or backward, read/write a data block, write block marks and file marks, set block size, lock/unlock tape drive, etc. Based on tape interface defined in mtio.h, the following function calls are developed for convenience.

- *block size: bool td_setblocksize(int blksize), int td_getblocksize()*
- status query: bool td_istapepresent(), bool td_pastEOF(), int td_getfile(), int td_getblock()
- *tape positioning: bool td_seek(int file, int block) which intelligently moves tape drive to the right block based on the information of tape drive current position and destination position.*
- data operations: int td_read(char* buf, int size), int td_write(const char* buf, int size), void td_flush().

To maximize the performance of tape data operations, data buffers are maintained for read and write respectively. Read buffer is used to facilitate large read while write buffer is used to facilitate large write. These data buffers can effectively reduce tape repositioning overhead and improve tape I/O performance.

3.2.4. Data buffering and interleaving for write operations To improve tape file system performance, it is crucial to reduce tape drive repositioning. The techniques of data buffering and interleaving are used to reduce the number of tape drive repositioning operations by streaming tape drive as long as possible with the same amount of data. Data for a write operation is buffered at main memory before committing to tape media. Once data is

Table 1. Main user mode system calls

System call	Note
int layer_tfs_getattr (const char *path, struct stat *stbuf)	Get tape file attributes.
int layer_tfs_getdir (const char *path, fuse_dirh_t h, fuse_dirfil_t filler)	Read the contents of a directory. This operation is the opendir(), readdir(), ..., closedir() sequence in one call. For each directory entry the filldir function should be called.
int layer_tfs_mknod (const char *path, mode_t mode, dev_t rdev)	Create a file node. There is no create() operation, mknod() will be called for creation of all non-directory, non-symlink nodes.
int layer_tfs_chmod (const char *path, mode_t mode)	Change the permission bits of a file.
int layer_tfs_chown (const char *path, uid_t uid, gid_t gid)	Change the owner and group of a file.
int layer_tfs_truncate (const char *path, off_t size)	Change the size of a file.
int layer_tfs_utime (const char *path, struct utimbuf *buf)	Change the access and/or modification times of a file.
int layer_tfs_open (const char *path, struct fuse_file_info *fi)	File open operation.
int layer_tfs_read (const char *path, char *buf, size_t size, off_t offset, struct fuse_file_info *fi)	Read data from an open file. Read returns exactly the number of bytes requested except on EOF or error.
int layer_tfs_write (const char *path, const char *buf, size_t size, off_t offset, struct fuse_file_info *fi)	Write data to an open file. Write should return exactly the number of bytes requested except on error.
int layer_tfs_release (const char *path, struct fuse_file_info *fi)	Release an open file. For every open() call there will be exactly one release() call with the same flags and file descriptor.

successfully buffered at main memory, HPTFS acknowledges the user application with a write success status thus speeding up the system write speed. The system has multiple worker threads receiving and buffering transmitted data from each individual data streams resulting in multiple data write buffers, which is essential for maximum tape write performance. When the buffered data from a file reaches a preset size or a release request is received, the buffered data is committed to tape as tape data blocks. A data block is filled with additional zeros if it does not have the size of a tape data block. To reduce the required memory size and have a better write concurrent support, data from different applications are interleaved at block level on tape media. It is possible that blocks belonging to a user file may not be contagious on the tape. ObjectIDs, composed of volume ID and object start position, are used to distinguish blocks. Each tape block carries an ObjectID as its block header. ObjectID uniquely identifies the owning object of a block. Multiple writes through multiple threads may cause data corruption and have to be serialized.

3.2.5. Data buffering and de-interleaving for read operations For a read operation, tape drive is positioned to the beginning of a requested object with fast forward/backward operation. Data blocks are read from tape, filtered with the requested ObjectID and the requested data offset, and finally assembled with block header removed. Filtering and assembling operations happen in the memory. To fill a given large read buffer, variable tape passes may occur due to changing interleaving status. The implemented read strategy is used to reduce the number of tape read operations and make tape work in streaming mode as often as possible. Depending on the requested data offset and size, a read request may not trigger a new tape read operation. The policy of First In and First Out (FIFO) is used for data buffer replacement. With the current implementation, read requests are serialized to be served one at a time. In the near future we plan to serve multiple requests concurrently if required data happen to be interleaved on tape. This will improve the aggregate read performance, and data can be read from tape as fast as they are written to tape.

4. Tape data layout

Slow data transfer rate to tape causes a lot of tape drive repositioning. Due to inadequate buffer or repositioning time the tape drive could write slower than the data transfer speed even the transfer speed is slower than the tape drive rated speed [25, 26, 27].To make tape drive work in streaming mode as frequently as possible and support concurrent writes on a sequential access media, multiple data streams are interleaved to tape at block level (more details in Section 5.3). Block level data interleaving help make tape drive work in streaming mode as often as possible when each individual data transfer rate to the tape drive is slow. Block level interleaving also help share expensive tape drive among users with good performance.

To make tape data self-describing, each tape needs maintain its own user data and metadata. Whenever us data is appended to tape, tape metadata has to be chang accordingly. Since tape is a one-dimentional storage, the are only three potential places to store metadata inform tion: the beginning of user data, within user data, and t end of user data. To store data at the beginning, tape me data has to be stored as a separate partition. With a pa titioned tape, data update in one partition will not affe data in other partitions. When a tape is not partitioned, t entire tape is dedicated to a single data set. If data is ov written at some place, any previously written data past t new end of data (EOD) mark on the tape becomes inacce sible. However, not all tape drives support tape partitioni (It is easier to support partitions with helical recording than linear recording.) Another difficulty of tape partitioning is that it is hard to decide how much space needs to be reserved for the metadata partition in advance. If metadata is mixed with user data, tape scanning is required to collect metadata information when the tape is first accessed. Scanning a high capacity tape takes a long time. In our designed system, metadata is stored at the end of user data. During write operation, metadata is maintained in main memory and written onto tape when tape is unmounted. After tape is re-mounted and metadata is read into main memory, user data can be appended to the end of last user data write.

For data compatibility and potential conversion, a tape header is needed at the beginning of user data. Thus each tape in our system has three segments maintaining tape header, user data and metadata. For easy location, there is a file mark at the end of each segment. Logically speaking, tape header, user data and tape metadata are stored at fixed locations respectively: at the tape beginning, after the first file mark and after the second file mark. The tape data layout is illustrated in Figure 2. The first segment contains just one-kilobyte data block recording the tape header which consists of HPTFS version number, used tape data block size, tape identifier, and other tape media information. Within the user data segment, data blocks from multiple user files are interleaved together. Data blocks of a user file may not be next to each other, which are affected by the degree of write concurrency and the relative speed of each individual data stream. With a higher data stream rate for a write, the total length of the object scattered over the tape will be shorter with data blocks closer to each other. A block header, which is composed of objectID and shown in Figure 3(a), is required to distinguish a block of file A from a block of file B since we can not have a file mark to separate file A from file B. The last segment on the tape maintains the metadata describing all the user files on the tape. Each user file on tape has one metadata entry stored as an element of an array. In addition to what a UNIX file *struct stat* contains, a metadata entry, as shown in Figure 3(b),

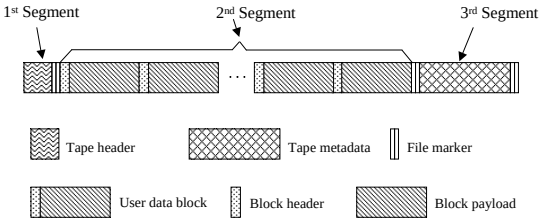

Figure 2. Tape data layout

```
struct objid
{
   int vol;
   int f_no;
   int b_sp;
   int seq;
};
```

(a) Object id

```
struct tapemeta
{
   char name[1024];
   int f_no;
   int b_sp;
   int b_ep;
   struct objid id;
   struct stat stbuf;
   struct tapemeta *next;
};
```

(b) Metadata entry

Figure 3. Data structure

also bears the object ID and the start position of the object. For the purpose of redundancy, intelligent read scheduling and read backward, the ending position of the object and the segment number are also stored in the metadata entry. Segment number starting with 0. In the near future, metadata can be move to the memory chip embedded with tape cartridge or cassette for improved performance. Even if the metadata is too large to fit in the chip, pointers to the metadata in the chip will also enhance performance.

In our current implementation, after a tape is mounted tape header is read and examined according to the expected format. If the tape is blank, a proper header can be created and the tape is marked by the system with empty metadata. For a non-empty tape, a data request will incur a read operation for the metadata segment. Once the metadata segment is read into memory, it is converted into a double linked list with each entry corresponding to one user file. In the future implementation the file directory in memory will be represented as a rooted tree with unbounded branching [28] and still be physically stored as an array on tape. All the data and metadata are managed by the user daemon process *hptfs*. *hptfs* is also responsible for data format translation between data on tape and data in VFS format.

5. Other design issues

There are some other design issues worth mentioning. We will discuss them in this section.

5.1. Kernel space versus user space

While implementing the HPTFS, the first major decisic that we have to make was whether this file system should b implemented in the kernel or the user space. The trade-of lie in file system speed, ease of implementation and co of future maintenance. Normally a file system is impl mented in operating system kernel for faster speed. A fi system implemented in user space has the advantages c user-friendly debug environment, shortened developme cycle, and lower maintenance cost. If we think deep abo tape operation, the millisecond difference of file access b tween kernel space implementation and user space impl mentation is nothing comparing to the tape data seek l tency in terms of seconds or even minutes. With this ob servation, our system is built upon FUSE, a user space file system under Linux environment. Inheriting security features from FUSE, HPTFS ensures a secure, non-privileged mount. That is, the mount user cannot get elevated privileges with the help of the mounted file system. The mount user cannot illegitimately access information from and induce undesired behavior in other users' and the super user's processes [29].

5.2. Write buffer management

For concurrent writing, how to buffer write data affects the system write performance. Among all possibilities, we chose to have a separate receiving memory buffer for each user file opened for write. When it receives the first write request of a data stream, HPTFS dynamically allocates a fixed size memory for the data stream and stick it with *struct fuse_file_info* associated with each opened file. This allocated buffer is released when the file is closed. For the maximum memory usage, we should dynamically change each receiving buffer size according to total system memory usage and individual file write speed. However, this buffering strategy is complex and error-prone. We chose simplicity over efficiency at this moment to use fixed buffer size for each file. Whenever a buffer is filled or a file close signal is received, the buffered is divided into blocks and sent to tape with block header added if the thread handling the write can get the exclusive lock to the open handle for tape drive. Multiple threads writing to the same tape naturally creates data interleaving on tape.

5.3. Choice of data interleaving

Data interleaving best suits the requirement of concurrent writes on a sequentially accessed tape which requires streaming mode for maximum I/O performance. Data can be interleaved at file level or block level as shown in Figure 4. File level interleaving puts blocks of a file continuously

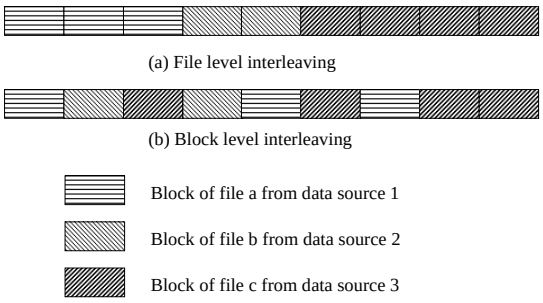

Figure 4. Interleaving comparison

on the tape while block level interleaving does not guarantee that. File level interleaving benefits the retrieval of any individual file if read data can be buffered or sent to clients fast enough since file level interleaving requires minimum tape drive read pass for each individual file, but it requires the data write buffer big enough to hold the whole file before it is committed to tape, which is prohibited for big files. We chose block level interleaving to achieve maximum write performance with less memory requirement.

5.4. Continuous data protection(CDP)

To effectively protect data from various data losses, it is important to have multiple data replicas, especially with replicas stored on disk and tape simultaneously. Keeping a replica on tape would be beneficiary to fight against possible data losses since tape is append-only media and easy to be sent to offsite for better data protection. Writing to disk and tape simultaneously fulfills the concept of continuous data protection [37]. Any newly created or modified data goes to disk and tape at the same time. The corresponding system architecture is shown in Figure 5.

With minor modification, our system can support writing to disk and tape simultaneously. This implementation has the side effect of supporting Read While Write. When tape drive is writing, read operation requires tape drive repositioning which may result in long latency due to costly data seek. When our system is configured to write to disk and tape simultaneously, read request could be served by disk instead of tape. Within our system, a user has two options: write to tape or write to disk and tape. If the data has the characteristic of write once read rare, data can be directly stored on tape without involving any disk operation. If the data has the characteristic of write once read many, data can be stored at tape storage and disk storage concurrently such that any read request will be served by the disk. For both situations, data is stored at tape at the first place completely eliminating data backup window.

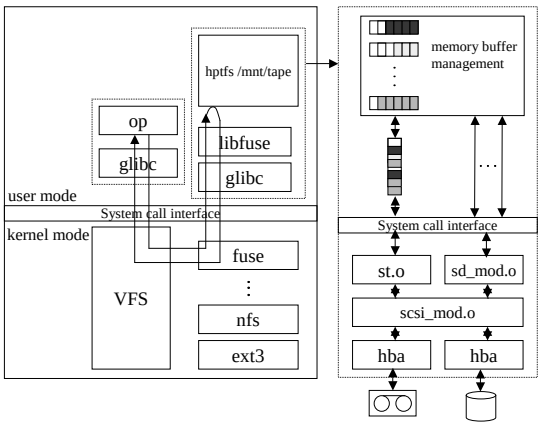

Figure 5. Continuous data protection

6. Performance evaluation

A variety of performance tests were run on HPTFS to evaluate its performance under different workloads. For a tape file system, it is important to test its I/O performance to make sure the tape drive can work in streaming mode; for our designed file system (HPTFS), we test its concurrent write feature to make sure it can support write sharing among applications. These tests are carried out by writing big chunk of data to tape through HPTFS using application *dd*. As mentioned in 2.1.1, *dd* is a commonly used application which can be used to transfer data to tape with a changeable I/O block size.

We are not expecting that HPTFS will be good at random read due to the sequential nature of tape media, but for the sole purpose of functionality test, we used PostMark[31] to test the sequential write and random read performances of HPTFS under a given range of file sizes since the main targeted operations for HPTFS are file create, write and read. PostMark is a file system benchmark which simulates the operation of electronic mail and news servers by performing a series of file operations on short-lived files with changing sizes within a given range.

And finally we test the correctness of HPTFS write/read operations. We test if data can be correctly written to and read from tape using an operating system command *cp*. This also tests if HPTFS really provides a general file system interface for existing applications to write to/read from tape without any modification.

6.1. System settings

To evaluate if HPTFS scales with hardware upgrades, we have two test systems set up as follows:

- *Setting A: Linux kernel 2.6.9; A PC with one Intel CPU (600MHz), 256MB memory and Maxtor IDE hard drive (54098H8, 7,200RPM); A StorageTek T9840A tape drive with 10MB/s native data transfer rate.*

- Setting B: Linux kernel 2.4.26; A PC with four Intel(R) XEON(TM) CPU (2.40GHz), 3GB memory and Hitachi Deskstar IDE hard drive(IC35L090AVV207-0, 7,200RPM); A StorageTek T9940A tape drive with 10MB/s native data transfer rate.

Depending on the configuration, data is written to tape and disk simultaneously or write to tape only. With compression turned on we can get higher tape data transfer rate. The actual data compression rate mainly depends on the compressibility of the data. To emulate high performance tape drive, data with high compressibility, such as zeros, is sent to tape drive. Tables in the following subsection show the mean performance value and the corresponding standard deviation of each test case under 20 runs.

6.2. System performance results from *dd*

Command *dd* is used to rate system performances with different configurations. A series of commands similar to the following are used to rate the tape write performance.

time dd if=/dev/zero of=/dev/nst0 bs=256k count=8000

Where the special file */dev/zero* is a source of zeroed memory with infinite length. Reads from it always return a buffer full of zeros; Under Setting A, the write speed of tape drive STk9940A is rated as 29.759MB/s with standard deviation of 0.020MB/s. Under Setting B, the write speed of tape drive STK9940A is rated as 37.604MB/s with standard deviation of 0.087MB/s.

The commands similar to the following are used to rate tape read performance.

time dd if=/dev/nst0 of=/dev/null bs=256k count=8000

Where the special file */dev/null* is a file to which any data written gets discarded and file */dev/null* always remains empty. Under Setting A, the read speed of tape drive STk9940A is rated as 26.562MB/s with standard deviation of 0.007MB/s. Under Setting B, the read speed of tape drive STK9940A is rated as 35.126MB/s with standard deviation of 0.005MB/s.

To test the concurrent write performance of HPTFS, *n* commands similar to the following are issued under different settings. The transfer size is the aggregate data size of all issued commands and the transfer time is counted as the longest write completion time.

time dd if=/dev/zero of=/mnt/tape/1 bs=blocksize count=blocks & time dd if=/dev/zero of=/mnt/tape/2 bs=blocksize count=blocks & ... time dd if=/dev/zero of=/mnt/tape/n bs=blocksize count=blocks &

6.2.1. Concurrent write performance evaluation We test if the tape file system (HPTFS) can handle concurrent writes and if HPTFS can stream the tape drive accordingly. The major file system calls of HPTFS are executed in user space, any data requested by application has to be passed to kernel mode by system call first, and then passed back to user mode by fuse kernel module, and finally results being passed back to kernel space and written to tape by user mode daemon. Without the HPTFS file system, data from application such as *dd* or *tar* is directly passed to kernel mode and written to tape.

Table 2 shows the concurrent write performance of HPTFS. As shown,under Setting B HPTFS streams the tape drive with concurrent data writes. Under Setting A, HPTFS achieves more than 80% of the rated tape drive write speed. It seems like a 600MHz CPU does not provide adequate processing power to achieve maximum write performance. Under Setting B, the changing degrees of concurrent writes do not significantly change the write performance of HPTFS. Tests also show that the HPTFS write performance is not significantly affected by changing data buffer sizes allocated for each individual write. However, if the buffer size is reduced to just one tape data block size, the system shows wide range of write speeds. In the following testing, we fix the buffer size as 40 tape blocks (10MB within our testings) for each individual write/read session.

Table 2. Tape write performance (MB/s, tape block size=256KB)

Degree of concurrency	Setting A rate		Setting B rate	
	Mean	Stdv	Mean	Stdv
2	24.148	0.433	37.709	0.004
3	24.222	0.392	37.713	0.005
4	24.169	0.373	37.719	0.005

6.2.2. Individual file read performance evaluation To test the individual file read performance of HPTFS, files with size of 500MB are first interleaved to tape with interleaving degree as 1, 2, 3 and 4, and then files are read from tape to memory one by one. The time interval between two reads are big enough to make sure the tape drive has been fully stopped. Table 3 shows the tape read performance versus data interleaving degree. For a specific file, the effective data retrieval rate is affected by the degree of interleaving. The results indicate that there are some improvement space for our current implementation to achieve the rated read speed.

Table 3. Tape read performance (MB/s, tape block size=256KB)

Degree of interleaving	Setting A rate		Setting B rate	
	Mean	Stdv	Mean	Stdv
1	16.170	0.03	31.811	2.896
2	9.097	0.094	15.095	0.665
3	6.391	0.199	11.420	0.657
4	5.104	0.164	8.632	0.379

6.3. System performance results from PostMark

We configured the PostMark to create 1,000 files on tape and perform 100 read operations. When a file is created, a random file length within the given range (256KB to 30MB) is selected, and text from a generated pool is appended up to the chosen length. When a file is to be read, a randomly selected file is opened, and the entire file is read into memory. For comparison, disk write/read performances are listed in Table 4 and Table 5 respectively. As mentioned before, the table shows the mean value and corresponding standard deviation from 20 runs, where each run writes 1,000 files and reads 100 randomly chosen files.

Table 4. Disk write performance with PostMark(MB/s, write block size=256KB)

Degree of concurrency	Setting A rate		Setting B rate	
	Mean	Stdv	Mean	Stdv
1	21.648	0.991	29.483	0.064
2	21.373	0.254	29.781	0.085
3	20.87	0.092	28.922	0.042
4	19.96	0.349	27.263	0.085

Table 6 and Table 7 show HPTFS write performance and read performance respectively when HPTFS is configured to write to tape only. Figure 6 shows the relationship between read performance and batch size for Setting A and Setting B when read requests are handled in batches. Tape requests batch operation combined with optimal scheduling [32, 33] is used to reduce the number of random tape I/O's. In the testing, requests are queued until the preset batch

Table 5. Disk read performance with PostMark(MB/s, write block size=256KB)

Degree of interleaving	Setting A rate		Setting B rate	
	Mean	Stdv	Mean	Stdv
1	25.903	1.538	17.357	0.566
2	8.827	0.307	12.804	0.339
3	8.232	0.165	11.104	0.212
4	7.433	0.208	10.523	0.566

size is reached. Requests are then ordered based on their position on the tape and the request positioned closer to the beginning of the tape is served first. There are many other scheduling algorithms are described in [34, 35]. Increasing batch size improves tape read performance by reducing random tape I/O operations.

Table 6. Tape write performance with PostMark(MB/s, tape block size=256KB)

Degree of concurrency	Setting A rate		Setting B rate	
	Mean	Stdv	Mean	Stdv
1	15.792	0.019	31.934	0.042
2	17.520	0.113	37.551	0.014
3	16.653	4.831	37.485	0.318
4	12.920	0.057	37.109	0.368

Table 7. Tape read performance with PostMark(MB/s, tape block size=256KB)

Degree of interleaving	Setting A rate		Setting B rate	
	Mean	Stdv	Mean	Stdv
1	1.750	0.021	2.975	0.106
2	1.835	0.049	2.754	0.014
3	1.695	0.034	2.265	0.021
4	1.470	0.127	2.085	0.022

6.4. CDP write performance

Table 8 shows the write performance of HPTFS when HPTFS is configured to write to disk and tape simultaneously. The tests are performed using application *dd* as described before.

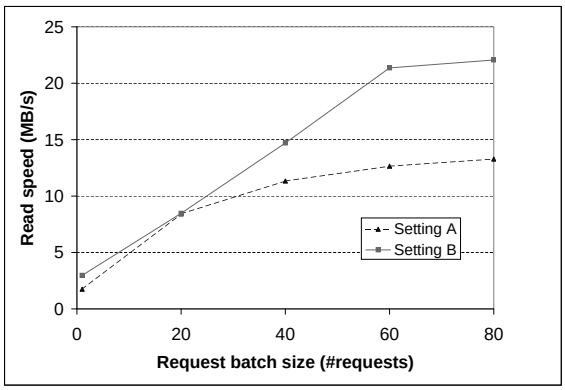

Figure 6. Tape read performance (degree of interleaving is 1)

Table 8. CDP performance(MB/s, write to tape and disk simultaneously, block size=256KB)

Degree of concurrency	Setting A rate		Setting B rate	
	Mean	Stdv	Mean	Stdv
2	12.232	0.704	37.391	0.151
3	12.209	0.901	37.632	0.009
4	11.907	0.680	37.329	0.281

It clearly shows that the write performance of Setting A degrades significantly compared to write to tape only configuration while the write performance of Setting B does not nearly change. Comparing the two settings, Setting A has a really slow single CPU, which leads us to believe the CPU processing power plays a big role here.

6.5. File operation correctness verification

With HPTFS write to tape is just like to write to disk. Command *cp* can be used to copy data back and forth between tape and disk. The following series of commands in Table 9 shows that data can easily and correctly be written to tape and read from tape. The digest results of SHA1 indicates the file content of fuse.c remains the same after being copied from disk to tape and then copied back from tape to disk.

7. Conclusion

We have described the design and implementation of HPTFS, a file system that provides a generic file system interface for applications to access tape data. As a file system, it provides a lot of desirable features, such as tape storage sharing, high performance write, etc. Being simple and easy to use is the main strength of this system. Within

Table 9. "Screenshot" and annotation

Commands and outputs	Notes
oak%./HPTFS /mnt/tape w	Mount tape in write mode at /mnt/tape
oak% ls -lt ∗.c -rw-r–r– 1 root root 61725 Jun 2 04:50 fuse.c -rw-r–r– 1 root root 12461 Jun 2 04:50 helper.c -rw-r–r– 1 root root 5064 Mar 21 05:37 fuse_mt.c -rw-r–r– 1 root root 3045 Feb 2 2005 mount.c	List all C files under current folder (on disk)
oak% cp ∗.c /mnt/tape	Copy all C files from disk to tape
oak%fusermount -u /mnt/tape	Write out metadata to tape and umount tape
oak%./HPTFS /mnt/tape r	Mount tape in read mode at /mnt/tape
oak%./HPTFS /mnt/tape r	Mount tape in read mode at /mnt/tape
oak%ls -lt /mnt/tape -rw-r–r– 1 root root 61725 Aug 15 23:55 fuse.c -rw-r–r– 1 root root 5064 Aug 15 23:55 fuse_mt.c -rw-r–r– 1 root root 12461 Aug 15 23:55 helper.c -rw-r–r– 1 root root 3045 Aug 15 23:55 mount.c	List all C files on tape media
oak% cp /mnt/tape/fuse.c ./fuse_1.c	Copy fuse.c from tape to disk as fuse_1.c
oak% openssl OpenSSL> sha1 fuse.c SHA1(fuse.c)= c8ab9be7c2edc1128db66f877b40ceeafffb74f6 OpenSSL> sha1 fuse_1.c SHA1(fuse_1.c)= c8ab9be7c2edc1128db66f877b40ceeafffb74f6	Comparing the original fuse.c on disk to fuse_1.c copied from tape with SHA1.

the designed file system, writing to a tape is as easy as writing to a disk. With memory buffering and data interleaving, backup can be operated with ease as well as high speed without using intermediate disk storage for data staging. With the high capacity of modern tape, the number of tapes used for a mid-scale enterprise within one year can be dramatically reduced to fit into a mid-scale tape library. The user perceives "infinite" storage space without worrying about storage quota limitations.

If enough write data is provided, the file system streams the tape drive; otherwise it will reduce the number of tape writes by efficiently buffering data in memory. By performing data buffering at physical memory instead of staging disk, HPTFS eliminates the cost and provisioning of disk capacity or the backend complexity of the data to tape later. With high storage capacity and high write speed, it provides a good platform for Internet backup applications. Across the non-secure Internet, users can send their valuable data to tape using their familiar secure protocols like scp, sftp, etc. Within such a file system, data from thousands of users can be supported by just one high performance tape drive, providing a cost-effective solution for personal data backup. HPTFS can be configured to allow applications to write newly created or updated data to disk and tape simultaneously for continuous data protection, eliminating the concept of a backup window and providing data real-time protection.

Backup software built over a tape file system would be greatly simplified. By moving tape data management to the file system, backup software can focus on policy scheduling and enforcement without going to the details of tape technology. The performance of such backup software would scale better with reduced complexity. A simplified backup process also means higher reliability, better protection and lower maintenance cost. The success of this file system motivates us to move the file system stack to the tape drive, therefore enabling an OSD tape drive. OSD (object storage device) [38] provides an object interface to applications and intelligent object management within the device; OSD enables the creation of self-managed, shared, and secure storage for storage networks. Backup software built over such an OSD tape drive would be further simplified with high write performance, better scalability and reliability.

Acknowledgment

This work is partially supported by DTC Intelligent Storage Consortium at University of Minnesota and its sponsor companies including StorageTek, Sun Micro, Symantec, Engenio, ETRI/Korea and ITRI/Taiwan. The authors would also like to thank Jim Diehl, Paul Drasler and Robert Lindgren for helping with the test environment setup.

References

[1] F. Moore, *Disk and tape pricing guidelines*, A white paper from Horison, Inc., 2002.

[2] Horison Information Strategies, *Data Protection and Recovery Strategies*, Sep. 2004.

[3] J. W. Toigo, *The Truth About Tape Storage*, Infostor, July 2005.

[4] *SAIT-1 Specifications*, http://www.aittape.com

[5] *Linear Tape Open (LTO) Ultrium tape drives*, http://www.storagetek.com

[6] *Sun StorageTek(TM) T10000 tape drive*, http://www.storagetek.com

[7] *Health Insurance Portability And Accountability Act of 1996*, Public Law 104-191-AUG. 21, 1996

[8] SEC 17 CFR Part 240, Release No. 34-38245, *Reporting Requirements for Broker Dealers Under the Security Exchange Act of 1934*, January 31, 1997.

[9] *SEC Interpretation: Electronic Storage of Broker Dealer Records*, Release No. 34-47806, US Securities and Exchange Commission. Effective May 12, 2003.

[10] *Meeting SEC 17a-4 Regulations Using WORM Tape and XenData Archive Series Software*, XenData white paper, May, 2004

[11] L. Zayatz, P. Doyle, J. Theeuwes and J. Lane (eds), *Information Explosion. Confidentiality, Disclosure, and Data Access: Theory and Practical Applications for Statistical Agencies*, Urban Institute, Washington, DC, 2001.

[12] P. L. Rao, *Tackling a data explosion, Network Magazine-Technology Decisions for The Enterprise*, Dec., 2002 http://www.networkmagazineindia.com

[13] D. Connor, *Fixed-content storage grabs users' attention*, Storage Networking World Online, June, 2003, http://www.snwonline.com

[14] *Data Storage Devices & Systems (DS2) Roadmap*, the Information Storage Industry Consortium (INSIC), Jan. 2005.

[15] http://www.bitpipe.com/plist/term/Backup-Software.html

[16] J. Ousterhout and F. Douglis, *Beating the I/O Bottleneck: A Case for Log-Structured File Systems*, ACM Operating System Reviews, Vol. 23, No. 1, pp. 11-28, January 1989.

[17] J. T. Kohl, Carl Staelin, and Michael Stonebraker, *HighLight: Using a Log-structured File System for Tertiary Storage Management*, In Proceedings of the USENIX Winter 1993 Technical Conference, pages 435-447, San Diego, CA, USA, 25-29 1993.

[18] D. A. Ford and J. Myllymaki, *A Log-Structured Organization for Tertiary Storage*, International Conference on Data Engineering (ICDE), 1996.

[19] *Beat the Clock On Shrinking Backup Windows*, Lakeview Technology, Nov. 2004

[20] *Data Protection - Backup Has Never Been Simpler*, HP white paper, July 2005

[21] R. Harada, *There's a Tape Solution For Every Organization*, Computer Technology Review, Nov. 2004. http://www.wwpi.com/

[22] R. Harada, *Scientific advancements continue to drive tape's market leadership*, Computer Technology Review, Jun., 2004.

[23] J. Yu and D. DeWitt, *Query pre-execution and batching in Paradise: A two-pronged approach to the efficient processing of queries in tape-resident data sets*. Proc. of 9th Int. Conf. on Scientific and Statistical Database Management, Olympia, Washington (1997).

[24] D. Colarelli and D. Grunwald, *Massive Arrays of Idle Disks For Storage Archives*. In Proceedings of the 15th High Performance Networking and Computing Conference, November 2002.

[25] V. Chinnaswamy, *Some Parameters in the Design of Streaming Tape Drives*. IEEE Symposium on Mass Storage Systems 1999: 241-251

[26] V. Chinnaswamy, *Analysis of Cache for Streaming Tape Drive*, Roc. Goddard Conference on Mass Storage Systems and Technologies, pages 299-310, Greenbelt, MD, Sep. 22, 1992.

[27] V. Chinnaswamy, *Mathematical Models for the Design of a Streaming Tape Drive*, Roc. of The IASTED International Conference on Modeling, Simulation and Optimization, Gold Coast, Australia, May 6-9, 1996.

[28] T. H. Cormen, C. E. Leiserson, and R. L, Rivest. *Introduction to Algorithms*. The MIT Press/McGraw-Hill, 1990.

[29] M. Szeredi, *FUSE: Filesystem in Userspace*. 2005. http://fuse.sourceforge.net/

[30] Dell Whitepaper, *Managing Data Protection with Red Hat Linux and Dell PowerVault Tape Autoloaders and Libraries*, Dell Power Solutions, Mar. 2004.

[31] J. Katcher, *PostMark: A New File System Benchmark*, Network Appliance, Tech. Rep. TR3022, October 1997.

[32] J. Li and C. Orji, *I/O scheduling in tape-based tertiary systems*, Journal of Mathematical Modelling and Scientific Computing, vol. 6, 1996.

[33] B. K. Hillyer , A. Silberschatz, *Random I/O scheduling in online tertiary storage systems*, Proceedings of the 1996 ACM SIGMOD international conference on Management of data, Jun., 1996, Montreal, Canada.

[34] S. More and A. Choudhary, *Scheduling queries on tape-resident data*, In Proceeding of the European Conference on Parallel Computing, 2000.

[35] S. Prabhakar , D. Agrawal and A. E. Abbadi, *Optimal Scheduling Algorithms for Tertiary Storage*, Distributed and Parallel Databases, v.14 n.3, Nov. 2003.

[36] http://www.kde.org/

[37] *Continuous Data Protection*, http://searchstorage.techtarget.com/

[38] *T10-OSD standard specification*, http://www.t10.org/