TLFS: High Performance Tape Library File System for Data Backup and Archive*

Dan Feng, Lingfang Zeng, Fang Wang, and Peng Xia

Key Laboratory of Data Storage System, Ministry of Education
School of Computer, Huazhong University of Science and Technology, Wuhan, China
{dfeng,lfzeng,wangfang}@hust.edu.cn

Abstract. A tape library is seldom considered as a viable place for constructing a file system for a sequential write/read device. Storage virtualization technology has become a buzzword in technology circles lately, in this paper we propose a tape library file system, called TLFS. The purpose of TLFS is to maintain a consistent view of mass storage so that the user can effectively manage it. Like disk file system, TLFS provides some file system operations, such as create, delete, open, close, read and write files/directories. It supports remote data access, backup, mirroring, replication by iSCSI protocol and facilitates fast data backup and archive. Moreover, TLFS supports large-scale storage management, provides file system fragment management, defenses virus and has transparent file movement. The prototype system, which is built by a SCSI disk and a tape library, is present and some implementation details are shown. Also, the experiment results are analyzed.

1 Introduction

Tape, as the cheaper and removable storage media, is good for backup and archiving. Always, frequently accessed data are reserved in enterprise high-end storage subsystem, and archiving data are stored in the cheapest storage media. This policy is a good choice for the information lifecycle management (ILM). However, the ILM problems can be solved to some extent by hierarchical storage technology and they are limited to what the underlying operating systems and file systems can support. Thus, more effective solutions are needed for this problem. At the same time, many organizations are turning to solutions that combine the vast capacity of traditional archiving approaches with direct access to secondary disk storage for delivering recovery, or access times – which range from seconds to minutes [33]. The volume of data archived in a distributed application environment is relatively low compared to proprietary mainframe environments, but this is looked upon as an opportunity for newer technologies

* This paper is supported at Huazhong University of Science and Technology by the National Basic Research Program of China (973 Program) under Grant No. 2004CB318201, National Science Foundation of China No.60303032, Huo Yingdong Education Foundation No.91068.

and the combined use of magnetic disk drive-based (e.g. RAID) and tape library solutions.

It is the primary benefits from archiving for end users to reduce in expensive primary disk space and to improve backup and archiving performance (as expressed in faster response time). However, simply buying an inexpensive RAID and archiving by a disk file system, has some limitations: (1)The backup software may require an additional license, and the old backup software based on tape (library), skilled operators. And operation flow may have to be discarded. So many traditional storage administrators require expensive training cost. (2)Backing up to a disk file system is more complicated and costly in management than backing up to tape (library). For example, with its removable characters, while tape library subsystems can be dynamically shared across multiple backup servers, disk cannot. Therefore, users will need to create separate volumes for each backup server and manage those volumes as needs grow. (3)While most backup software products understand when a tape runs out of space, it's not as straightforward when a disk file system runs out of space. A tape will get marked as full, while the disk will simply report an I/O error, and warnings can go off all over the place. Some backup software products keep attempting to write to a file system, even after it's full. (4)General disk file systems are more prone to being infested with viruses. (5)File systems have an inherent problem of fragmentation.

Motivated by the above limitations of general (disk-based) file systems, in this paper we propose a tape library file system, called TLFS, in an attempt to address the above problems. TLFS effectively integrates the virtual tape library (VTL) [3] technology and the iSCSI [2] technology to provide transparent tape file access for users while retaining other functions of tape libraries. Our study of TLFS shows the main advantages of TLFS over the conventional tape libraries and conventional (disk-based) file systems as follows: (1) High backup and restore performance. (2) Low cost compared with simple disk-based system. (3) Some finer functions integrated both the disk and the tape library.

The main contributions of this paper are: (1) Gives some winged words for most mainstream disk file systems. (2) Provides a tape library file system based on previous works. (3) Implements the prototype system (virtual backup and archive system) based on TLFS via virtual tape library technology and presents and discusses the experiment results.

2 Background and Related work

Nowadays, tape technology features huge capacity per cartridge (close to 1 TB with compression), low cost per storage unit and high streaming rate (greater than 100MB/s). For off-site storing and possible disaster recovery, tape backup or archive is still a strong candidate and even a must for exploding valuable data. TLFS is designed to provide a tape library as an ordinary disk-like storage device to users for writes. And the tape library transparently interleaves multiple user data streams for maximum write performance. Requests batches are intelligently scheduled to be served by the system for reduced response time for I/O requests.

2.1 Background

(a) Life and lifecycle of data. The value of data often varies considerably during its lifetime. Not only is the value of data dependent on its application, but also on the number of users who need to have access to it. Traditionally, this problem is solved by the hierarchical storage system. The declining cost of commodity disk drives is rapidly changing the economics of deploying large amounts of on-line storage. Conventional mass storage systems typically use high performance RAID as a disk cache, often with a file system interface. The disk cache is backed by tape libraries which serve as the final repository for data. Information Lifecycle Management (ILM) [29] makes data preservation a lifetime storage management. There are about four fundamental stages in the lifecycle of computer data, such as data creation, data access, data archiving and data deletion. ILM solutions can significantly reduce the cost and complexity of data storage. ILM has two benefits for users. One is to minimize administration costs. The other is to make the most efficient use of storage hardware. However, for ILM, the most difficult implementation is the design of suited file system which gives attention to disk storage subsystem, tape library subsystem or optical disk storage subsystem.

(b) Backup window. Application servers traditionally stream data to a tape device for data backup or archiving. For many applications, this operation can take many hours. During the backup, the application might not be able to provide normal service. In cases where the application is still providing service, its performance will be severely degraded, as the backup involves moving a large amount of data to the tape device. Most businesses have constraints on how long the backup time can last. "Time is money", the longer backup windows, the more losses are for bank-like businesses. Obviously, a disk drive-based backup solution may reduce the backup window from a few hours to only a few minutes due to the relatively very high performance of disks in terms of bandwidth and throughput.

(c) Virtual tape library (VTL). Virtualization is key to managing today's demanding requirements of file storages. It makes it possible to reduce storage management costs through more effective use of storage resources by applying one or more concepts or technologies such as hardware and software partitioning, time-sharing, partial or complete machine simulation, and many others. A VTL [3] has a finite number of virtual tapes (e.g. often limited by the capacity of RAID). The VTL technology lets disks emulate tape (libraries) by creating separate virtual tape (libraries) for each host while sharing the same physical tape library. This allows users to share a tape library among incompatible backup applications. Also, the VTL technology can provide faster backup and recovery for large systems without disrupting existing tape processes.

(d) iSCSI technology. Internet SCSI, or iSCSI [2] is a TCP/IP-based protocol for establishing and managing connections between IP-based storage devices, hosts and clients. iSCSI is a transport protocol for SCSI which operates on top of TCP. It provides a new mechanism for encapsulating SCSI commands on an

IP network. Also, it is a protocol for a new generation of data storage systems that natively use TCP/IP.

2.2 Related general disk file systems

As mentioned above, with the Disk-to-Disk-to-Tape model [24], [25], [28], storage systems can access data at an online storage speed but at an offline storage price, approximately. However, disk-based backup has some shortcomings, as listed in Section 1, namely, mangled disk backup file directory, virus, disk failures, and unexpected data overwrite. Further, disk fragmentation is also a problem that can impact disks' write/read performances.

In additional, traditional disk file system can not be built on the tape (library) subsystem due to tape's sequential read/write characteristics, which in the most part has cost the popularity of tape (library) for general-purpose applications. The purpose of file systems is to maintain a consistent view of storage so that users can effectively managed storage. This is done in a way that allows the users to create files and directories as well as create, delete, open, close, read, write and/or extend the files on the device(s). File systems also maintain security over the files under their management and, in most cases, access control lists for a file.

There is a significant body of research related to distributed file systems [1], [4], [5], [8], wide area systems [9], [11], [13], [14], [15], [16], log file systems from a myriad of vendors such as Veritas (VxFS) [23], SGI (XFS) [18], IBM (JFS) [19], ADIC (StorNext)[26], HP (ADvFS) and Sun (UFS Logging), etc. and Linux file systems (EXT3 [21], GFS [22], ReiserFS [20]). However, they can not be built on a tape subsystem.

The integrated file system [31] is developed by IBM and it is a part of OS/400 that lets user support stream input/output and storage management similar to personal computers and UNIX operating systems, while providing user with an integrated structure over all information stored in the server by integrating with other file systems. Windows installable file system [32] facilitates the development of windows file systems and file system filter drivers. Its kit supports only windows environment. Neither reference [31] nor reference [32] is suitable for tape library storage management.

With the rapid development of Internet technology and information digitization technology, many file systems based on disk and tape storages have been presented. AMASS [26] is a cost-effective solution for enterprises that have more data than their disk capacity can support. However, as mentioned above, disks in AMASS are used based on the general-purpose file system, and thus they have to confront the same set of problems inherited from general-purpose file systems. SEPATON's Disk Dynamic File System allows large I/O streams to execute efficiently and has the built-in infrastructure to dynamically balance performance across all available disks in their VTL appliance. The Disk Dynamic File System has the important side effects of not only sustaining maximum throughputs, but also dynamically load balancing I/O streams without any requirement for performance "tuning" [8]. But, The Disk Dynamic File System is still a disk-related

file system and it is not feasible for the management of tape library storage subsystem.

3 Design and implementation of TLFS

3.1 Hybrid RAID-Tape-Library storage subsystem

Based on iSCSI technology, our target device comprises a RAID and a tape library connected by a SCSI channel, called a hybrid RAID-Tape-Library device. The console and application servers (Web server, E-mail server etc.) and the target device are interconnected by a TCP/IP network. The console, web server, e-mail server and backup server form an initiator and they access data in our hybrid device through the iSCSI protocol.

Utilizing an ordinary IP network, iSCSI transports block-level data between an iSCSI initiator on a server and an iSCSI target on our hybrid storage device. The iSCSI protocol encapsulates SCSI commands and assembles the data in packets for the TCP/IP layer. Packets are sent over the network using a point-to-point connection.

When an iSCSI initiator connects to our hybrid device (iSCSI target), the hybrid storage is seen by the operating system as a local SCSI tape (library) device. However, this hybrid device can be formatted like any other local disk-like device. And only with the help of our custom client software, the process is transparent to users to access our iSCSI hybrid storage device.

But, almost all of backup applications, such as tar [30], taper [12] and bacula [10], can access the target hybrid device. In this scenario, the hybrid storage subsystem more like a VTL, and TLFS is simplified to transform SCSI stream commands to SCSI block stream, or on the wary round.

The hybrid storage subsystem refers to an entity that presents itself as a SCSI direct access disk and tape sequential access while running within the Linux kernel space. The TLFS module is implemented in the initiator (still in the hybrid target device). It can work with the traditional disk-related file system and this solution is transparent for user space application. Also, users can custom their applications based on the APIs provided by the TLFS.

In the hybrid device, our function modules are implemented in SCST [17], which is a generic SCSI target middle level for Linux. It is designed to provide unified, consistent interface between SCSI target drivers and Linux kernel and simplify target drivers development as much as possible. Although data distribution policy is different comparing with reference [3] and reference [7], in substance, their implementation technologies are analogical. And they all have to record all the logical objects [6] on the RAID. The SCSI command analysis module receives SCSI sequential commands from the backup application, and determines whether the commands should be executed on the RAID or on the tape library. Then it delivers them to the proper module (or media). The SCSI command transform module is responsible for transforming SCSI sequential commands into SCSI block commands. The LBA (logical block address) mapping

module maintains the block mapping information, which associates the logical unit of an object with its logical block address in the RAID.

In addition, the TLFS provides a client application that is implemented using the API of TLFS. The client application may be deployed in the console and perform remote file management operations. Users also can implement their own remote file management application by the API of TLFS.

3.2 The implement details of TLFS's file operation

Traditionally, a file system represents the logical structures and software routines used to control access to the storage on a hard disk system. However, TLFS, a tape-based file system that hides the details about tapes and, provides an API for applications, is designed and implemented to provide the "access by name" functions, including tape file creation, tape file read, tape update, tape file deletion, tape file copy, tape file renaming and tape defragmentation.

```
/*virtual tape list*/
typedef struct VT_LIST {
        struct list_head link; /*Create bidirectional list*/
        struct LOL  *tape_logical_list; /*Cooralative logical list*/
        __u64 tape_size; /*Virtual tape capacity size*/
        __u32 begin_block; /*Begin block address*/
        int tape_id; /*Each tape id*/
        __u32 crr_block_address; /*Current logic block address*/
        struct LOL  *crr_logical_object; /*current logical object node*/
}Virtual_Tape_List;
/*logical object list, each virtual tape corresponding a logical object list*/
typedef struct LOL {
   /*Construct bidirectional list*/
        struct LOL *next;
        struct LOL *prev;
        /*Node type of logical object (logical block,setmark,filemark)*/
        __u8 type;
        __u32 LBA; /*Logical block address*/
        __u16 size; /*Data size of logical object*/
        __u8 tape_id; /*Virtual tape id*/
        __u64 node_id; /*Node id*/
   } Logical_object_Node;
```

Fig. 1. The virtual tape list and logical object list in the TLFS

(a) TLFS data structures and functions

The file allocation table (FAT) of every library slot and every tape are defined. The FILE_NODE constructs the file name information and the FILE_RECORD records some information of file in true tapes. In Figure 1, the virtual tape list and logical object list are defined. For the type element, its value may be one of the defined constants - LOGICAL_BLOCK, FILE_MARK, SET_MARK, BEGIN_NODE or END_NODE. Figure 2 shows main functions (in target) which response the requests from initiator(s). These functions dispose the transform from SCSI stream command to SCSI block command, such as INQUIRY (0x12), WRITE_FILEMARKS (0x10), REWIND (0x01), READ (0x8), WRITE (0xA),

MODE_SENSE (0x1A), SPACE (0x11) etc. Specially, for the RAID, the command type of write and read is 10, so the TLFS has two transform functions (transform_write_6to_10 and transform_read_6to_10).

```
static int get_inquiry_response(Scsi_Request *, int, int);
static int get_block_limits_response(Scsi_Request *, __u32);
static int transform_write_6_to_10(Target_Scsi_Cmnd *,int);
static int execute_write_command(__u16,Virtual_Tape_List *,__u8 *);
static int get_mode_sense_response(Scsi_Request *, int);
static int transform_read_6_to_10(Target_Scsi_Cmnd *, int);
static int execute_read_command(__u16,Virtual_Tape_List *,__u8 *);
static int transform_write_filemarks(__u8);
static int insert_list(Virtual_Tape_List *,Logical_object_Node *);
static int transform_space(__u8,__u8);
static int perform_rewind(void);
static int init_tape_list(void);
static int init_object_list(Virtual_Tape_List *);
void add_node(Logical_object_Node *,Virtual_Tape_List *);
```

Fig. 2. List of those main functions (in target part)

(b) The system/configure files in the TLFS

At the same time, the TLFS provides some system files and configure files which facilitate the configuration about the TLFS. The system files record some file metadata information in the TLFS, and the configure files provides the configure information of TLFS. For instance, the TapeLibrarySlotInfo.txt records the tape library slot information (defined by SLOT_FAT) both in the VTL and the true tape library. The TapeTLFSFAT.txt stores the information of tape file allocation table (defined by TAPE_FAT) in TLFS. The FileName.rec gives file name information (defined by FILE_RECORD), and the configure information for the TLFS is set or got by the administrator.

(c) File operation algorithms in the TLFS

This subsection shows those file operation algorithms, such as create, dir, read, update, copy, erase, rename and defragment etc. For each file operation algorithm, it begins at the initialization file system function - InitFileSystem(), and ends at the close file system function - CloseFileSystem(). Because those algorithms aim at the write or read files in a tape, they are all applicable to both the true tape library and the VTL. Also, as mentioned above, in the initiator, users can also use the APIs provided by the TLFS to implement tape file management. Key algorithms for the TLFS are as follows:

Create {

1: Open the file TapeLibrarySlotInfo.txt in the TLFS, create read/write buffer, initiate the process of read/write and open the device file (e.g. st0 or nst0) of VTL.

2: Judge if the online VTL tape runs out of space or not. If the tape runs out of space, exchange a new VTL tape.

3: Deal with the name confliction according to the SLOT_FAT and open the file of tape file allocation: TapeTLFSFAT.txt, and form the current TAPE_FAT.

4: Write data to the buffer.
5: The write process waits for that the buffer becomes full, and writes data to the VTL tape.
6: Write the remainder data in the buffer and close the FileName.rec, TapeTLFS-FAT.txt and TapeLibrarySlotInfo.txt.
7: Free the read/write buffer. And stop read/write process and close the device file of VTL. }

Dir {
1: Open the file TapeLibrarySlotInfo.txt in the TLFS, create read/write buffer, initiate the process of read/write and open the device file of VTL.
2: Read the every item in the TAPE_FAT of TLFS and list them one by one.
3: Close the TLFS and close the device file of VTL. }

Read {
1: Open the file TapeLibrarySlotInfo.txt in the TLFS, create read/write buffer, initiate the process of read/write and open the device file of VTL.
2: Judge if the required file is in the online VTL tape or not. If not, exchange a new VTL tape, and open the files: TapeTLFSFAT.txt and FileName.rec.
3: The read process reads the file to the buffer according to the first address of the TAPE_FAT and the FILE_RECORD.
4: The application reads data from the file buffer by the read function of TLFS.
5: Close the FileName.rec, TapeTLFSFAT.txt and TapeLibrarySlotInfo.txt.
6: Free the read/write buffer and stop read/write process and close the device file of VTL, close the TLFS. }

Update {
1: Open the file TapeLibrarySlotInfo.txt in the TLFS, create read/write buffer, initiate the process of read/write and open the device file of VTL.
2: Judge if the required file is in the online VTL tape or not by the SLOT_FAT. If not, exchange a new VTL tape, and open the file allocation table and the file record table of TLFS.
3: Receive the updating data from the application and write data to the buffer.
4: The write process writes the content of buffer into the tail of the VTL tape.
5: Update the file allocation table of the TLFS, follows as the four instances:

5-1. Insert the new data block into the file head and update the corresponding variable of the first record address in the TAPE_FAT. And insert new record item in the link head of FILE_RECORD and point to the new added address of data block in the VTL tape.

5-2. Insert new data block into the file tail and add new record item in the FILE_RECORD. And point to the address in the VTL tape for the new added data block.

5-3. Insert new record item in the middle of FILE_RECORD. And point to the address in the VTL tape for new added data block.

5-4. IF modify the file content. Write the data block which is updated into the VTL tape tail. And modify the relevant node pointer in the TAPE_FAT and point to the new data block address in the VTL tape tail.
6: Close the FileName.rec, TapeTLFSFAT.txt and TapeLibrarySlotInfo.txt.

7: Free the read/write buffer. And stop read/write process and close the device file of VTL. Close the TLFS. }

Copy {
1: Open the file TapeLibrarySlotInfo.txt in the TLFS, create read/write buffer, initiate the process of read/write and open the device file of VTL.
2: Judge if the required source file and the objective file are both online for the SLOT_FAT or not. If not, exchange new VTL tape(s).
3: Open the files: TapeTLFSFAT.txt and FileName.rec.
4: Read the data of source file to the buffer by the read process.
5: Write the content in the buffer to the objective VTL tape by the write process.
6: Update the FILE_RECORD and the TAPE_FAT and the SLOT_FAT of TLFS.
7: Close the FileName.rec, TapeTLFSFAT.txt and TapeLibrarySlotInfo.txt. 8: Free the read/write buffer. And stop read/write process and close the device file of VTL. Close the TLFS. }

Erase {
1: Open the file TapeLibrarySlotInfo.txt in the TLFS, create read/write buffer, initiate the process of read/write and open the device file of VTL.
2: Judge if the required deleted file is online in the VTL tape or not. If the file is not in the VTL tape, exchange a new VTL tape according to the SLOT_FAT.
3: Modify the SLOT_FAT and delete the file name in the SLOT_FAT. At the same time, reduce the number of file in the SLOT_FAT.
4: Modify the TAPE_FAT, and delete the corresponding item in TAPE_FAT.
5: The write process waits for that the buffer becomes full, and writes data to the VTL tape.
6: Close the TapeTLFSFAT.txt and TapeLibrarySlotInfo.txt.
7: Close the device file of VTL and TLFS. }

Rename {
1: Open the file TapeLibrarySlotInfo.txt in the TLFS, create read/write buffer, initiate the process of read/write and open the device file of VTL.
2: Judge if the required file is online in the VTL tape or not. If not, exchange a new VTL tape according to the SLOT_FAT.
3: Open the TapeTLFSFAT.txt.
4: Update the file name variable in the corresponding structure item in the TAPE_FAT.
5: Update the SLOT_TAPE and the corresponding file name variable in the FILE_RECORD.
6: Close the FileName.rec, TapeTLFSFAT.txt and TapeLibrarySlotInfo.txt.
7: Close the device file of VTL and the TLFS. }

Defragment {
1: Open the file TapeLibrarySlotInfo.txt in the TLFS, create read/write buffer, initiate the process of read/write and open the device file of VTL.
2: Open the files of TAPE_FAT both in the VTL tape1 and the VTL tape2, respectively.
3: Read the TAPE_FAT and the FILE_RECORD of the VTL tape2 into the memory.

4: Create the write/read buffers and startup the write/read processes.
5: Read the data in the source file to the corresponding buffer by the read process.
6: The read process sequentially reads the data of VTL tape2 according to the TAPE_FAT and the FILE_RECORD to the corresponding buffer. The write process sequentially writes the data in the corresponding buffer into the VTL tape1. At the same time, create the TAPE_FAT and the FILE_RECORD for the VTL tape1.
7: Close the FileName.rec and the TapeTLFSFAT.txt. Update the SLOT_TAPE and close the TapeLibrarySlotInfo.txt.
8: Free the read/write buffer. And stop read/write process and close the device file of VTL. Close the TLFS. }

3.3 Some features and test results

In traditional file server environments, adding data storage greatly increases management burdens and data exposure, and impacts performance and availability. Our TLFS allows administrators across large-scale, multi-vendor storage environments to efficiently manage their storage resource regardless of their size, data volumes, or high availability requirements. The TLFS solutions enable the mass storage system based on RAID and tape libraries to enjoy the benefits of capacity management, tiered storage management, disaster recovery, efficient utilization of small backup windows, reliable backup and restore, improved virus protection, file system fragment management, support large file and enhanced storage utility.

With the property of the tape, the TLFS decreases the impact of the fragment by sequential write in a "tape". If the primary backup system is a traditional RAID, then there is no protection from the virus. The virus simply transfers itself to the disk backup and continues eating away at valuable data. When an infected file is backed up onto our hybrid device, the virus goes into stasis and remains dormant. So the TLFS offers greater options and flexibility in defending against virus attacks. In addition, the TLFS permit selective restoration so only the infected files need be restored. The TLFS, unlike disk or optical file system, makes a copy of the actual file. A virus needs to be an executable file to do its damage and the TLFS is not an active traditional file system.

TLFS has the ability to provide location transparency for data actively being accessed without introducing data integrity, data access or performance risks. TLFS can transfer data from a virtual tape (always RAID) to a true tape in a tape library by using the logical object list. This process is transparent for users. According to some migration policies, TLFS will deal with the logical object list from the beginning node for flushing the data. When encountering a LOGICAL_BLOCK, a SCSI block READ command will be generated and delivered to the virtual tape. The command's logical block address field is the logical object's LBA; its TRANSFER LENGTH is the logical object's size. After reading data from the virtual tape, a SCSI sequential WRITE command can be generated, which only needs a transfer length. When a FILE_MARK or a SET_MARK

is encountered, a WRITE FILEMARK will be generated and delivered to the true tape in a tape library.

To test the write/read performance of our hybrid storage prototype subsystem, under the Redhat Linux system (Linux kernel 2.4.20-8), we adopted tar [30] and Taper [12] as the backup software. These three kinds of backup software performed well with the hybrid subsystem. The tape drive we adopted was HP MSL5030, and the tape media was hp ultrium 200GB data cartridge (C7971A). We adopted the SEAGATE ST3404LC SCSI disk to simulate virtual tapes. Our main concern is the backup time (write performance) and the re-

Table 1. The write/read performance in different Ethernet

Number of file	*Total size of file(Mbyte)*	**The hybrid device using the TLFS**			
		1000M Ethernet		**100M Ethernet**	
		Write Time(s)	*Read Time(s)*	*Write Time(s)*	*Read Time(s)*
1	50.0	2	3	3	4
1	200.0	10	17	12	21
1	1000.0	53	84	55	106
381	85.0	6	5	6	6
3386	455.1	39	29	40	30
22216	1000.0	112	79	112	91
10	2000.0	238	247	259	251

Table 2. The performance comparison of the local node of hybrid device and the physical tape

Number of file	*Total size of file(Mbyte)*	**The hybrid device using the TLFS**			
		1000M Ethernet		**100M Ethernet**	
		Write Time(s)	*Read Time(s)*	*Write Time(s)*	*Read Time(s)*
1	50.0	2	2	8	17
1	200.0	9	16	32	59
1	1000.0	47	80	163	305
381	85.0	5	4	16	24
3386	455.1	27	20	87	119
22216	1000.0	84	52	132	223
10	2000.0	92	112	268	330

store time (read performance) for different primary backup devices. Also, the write/read performance both in different Ethernet and in the local node was tested in our experiment. Table 1, Table 2, Table 3 and Table 4 show the results. Each table has four group data, and each group records those test results about the write/read time or write/read throughput. Moreover, we adopted different

number of file, and the total size of file(s) was also ranked from 50Mbyte to 1Gbyte. The results showed that the average write time of the hybrid device in

Table 3. Throughput in different Ethernet

Number of file	*Total size of file(Mbyte)*	**The hybrid device using the TLFS**			
		1000M Ethernet		**100M Ethernet**	
		Write Throughput(Mbyte/Min)	*Read Throughput(Mbyte/Min)*	*Write Throughput(Mbyte/Min)*	*Read Throughput(Mbyte/Min)*
1	50.0	1509.7	1000.0	603.8	750.0
1	200.0	1182.0	705.9	1006.4	571.4
1	1000.0	1115.1	714.3	1090.0	566.0
381	85.0	850.6	1020	850.6	850.6
3386	455.1	684.7	941.6	681.2	910.2
22216	1000.0	503.9	759.5	503.9	659.3

Table 4. The throughput comparison of the local node of VTL and the physical tape

Number of file	*Total size of file(Mbyte)*	**The hybrid device using the TLFS**			
		1000M Ethernet		**100M Ethernet**	
		Write Throughput(Mbyte/Min)	*Read Throughput(Mbyte/Min)*	*Write Throughput(Mbyte/Min)*	*Read Throughput(Mbyte/Min)*
1	50.0	1509.7	1509.7	377.4	187.5
1	200.0	1313.3	750.0	407.5	203.4
1	1000.0	1257.4	750.0	399.3	196.7
381	85.0	1020.7	1275.0	510.3	212.5
3386	455.1	989.1	1335.3	460.4	224.4
22216	1000.0	672.1	1153.8	427.7	269.1

1000M Ethernet is 99.14% of that in 100M Ethernet and the average read time in 1000M Ethernet is 95.68% of that in 100M Ethernet, which indicated that the different network environment has few influence for our hybrid subsystem. However, the test results in local node indicate that network has large influence for our hybrid subsystem. For instance, the average write time of the hybrid device in local host is 54.62% of that in 100M Ethernet and the average read time in local host is 61.64% of that in 100M Ethernet.

At the same time, the size of single file much affects the write performance of our virtual backup and archive system. For example, Table 3 and Table 4 show the write throughput are descending with the ascending number of file. Moreover, the average backup speed of the hybrid device in local node is 2.65

times of that of the physical tape and the average restore speed of the hybrid device in local node is 3.77 times of that of the physical tape, which indicates that the hybrid device can enhance the backup speed greatly. So we can get a conclusion that our virtual backup and archive system is fit for backup and restore applications, specially, for large size file.

4 Conclusion and the future work

The proposed TLFS, based on the VTL, RAID and iSCSI technologies, provides storage capacities matching or exceeding those of tape libraries but with performance and availability similar to those of a traditional RAID. Moreover, we show that through a combination of effective file system management of RAID and tape library, this high performance architecture can be implemented at a very low cost. Because TLFS uses a UNIX-based directory and file system interface, it is transparent to applications and users. It provides the same set of tools to organize and access near-line data as that used for accessing data stored on a RAID. TLFS is scalable, capable of potentially supporting archives with volumes ranging from a few gigabytes to more than a petabyte. Based on the prototype, system performance is presented and improvements are analyzed to achieve higher write/read performance.

References

1. Ji, M., Felten, E., Wang R., Singh, J. P.: Archipelago: An Island-Based File System for Highly Available and Scalable Internet Services. In: Proceeding of 4th USENIX Windows Systems Symposium. (2000)
2. Satran J., et al.: Internet Small Computer Systems Interface (iSCSI). Available from: http://www.ietf.org/rfc/rfc3720.txt (2004)
3. Mu, F., Shu, J., Li, B., Zheng, W.: A Virtual Tape System Based on Storage Area Networks. In: H. Jin, Y Pan, N. Xiao and J. Sun (Eds.): GCC'2004 Workshop on Storage Grid and Technologies, LNCS 3252, (2004) 278-285
4. Anderson, T. E., Dahlin, M. D., Neefe, J. M., Patterson, D. A., Roselli, D. S., Wang, R. Y.: Serverless network file systems. ACM Transactions on Computer Systems, 14(1), (1996) 41-79
5. Gronvall, B., Westerlund, A., Pink, S.: The Design of a Multicast-based Distributed File System. In:Proc. of OSDI, (1999)
6. ANSI: SCSI Stream Commands-2 (SSC-2), revision 09, 9 July 2003, http://www.t10.org
7. Myllymaki, J., Livny, M.: Disk-tape joins: synchronizing disk and tape access. ACM SIGMETRICS Performance Evaluation Review (1995) 279-290
8. Direct Access File System. Website, May 20, 2005, http://www.dafscollaborative.org
9. Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer,W., Wells, C., Zhao, B.: OceanStore: An architecture for global-scale persistent storage. In Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (2000) 190-201

10. Website, January 10, 2005, http://www.linux.org/apps/AppId_8816.html
11. Dabek, F., Frans Kaashoek, M., Morris, R., Stoica, I.: Wide-Area Cooperative Storage with CFS. Proceedings of the 18th SOSP (2001)
12. Website, January 10, 2005, http://www.e-survey.net.au/taper/
13. Gronvall, B., Westernlund, A., Pink, S.: The design of a multicast-based distributed file system. In: Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans, LA (1999) 251-264
14. Yu, H., Vahdat, A.: Design and evaluation of a continuous consistency model for replicated services. In: Proceedings of the Fourth Symposium on Operating Systems Design and Implementation, San Diego, CA (2000) 75-84
15. Keleher, P. J., Cetintemel, U.: Consistency management in Deno. Mobile Networks and Applications, 5(4), (2000) 299-309
16. Petersen, K., Spreitzer, M. J., Terry, D. B., Theimer, M. M., Demers A. J.: Flexible update propagation for weakly consistent replication. In: Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles, Saint Malo, France (1997) 288-301
17. Palekar, Ashish A., et al.: Design and Implementation of a Linux SCSI Target for Storage Area Networks. Proceedings of the 5th Annual Linux Showcase & Conference, (2001)
18. Website, May 20, 2005, http://oss.sgi.com/projects/xfs/
19. Website, May 20, 2005, http://oss.software.ibm.com/developerworks/opensource/jfs/
20. Website, May 20, 2005, http://www.namesys.com
21. Website, March 20, 2005, http://www.zip.com.au/ akpm/linux/ext3/
22. Website, February 11, 2005, http://linux4u.jinr.ru/LinuxArchive/Ftp/kernel/gfs/4.2/
23. Website, May 20, 2005, http://www.veritas.com/products/category/ProductDetail.jhtml?productId=filesystem
24. George Crump, SANZ Inc., Best practices for implementing disk-to-disk backup: Part 2. March 15, 2005. Available from: http://www.computerworld.com/printthis/2005/0,4814,100383,00.html
25. TotalStorage P2P Virtual Tape Server , IBM. Website, 2005. http://www-306.ibm.com/software/,23 May 2005.
26. Website, May, 2005, http://www.adic.com/
27. Paul Feresten. Comparing Host-Based D2D to VTLs for Backup and Restore - Part 2. Website, 2005. http://www.wwpi.com/index.php?option=com_content&task=view&id=132&Itemid=67
28. Fred Moore. Storage: The Outer Limits. February 24, 2005. Available from: http://www.cio-today.com/news/story.xhtml?story_id=023001I0WU0B
29. Reiner, D., Press, G., Lenaghan, M., Barta, D., Urmston, R.: Information lifecycle management: the EMC perspective. In: Proceedings of 20th International Conference on Data Engineering, (2004) 804-807
30. Website, January 10, 2005, http://savannah.gnu.org/projects/tar/
31. Website, July 3, 2005, http://publib.boulder.ibm.com/iseries/v5r2/ic2924/info/rzaia/rzaia_ifs_intro.htm
32. Website, July 3, 2005, http://www.osr.com/services_ifskitsupport.shtml
33. Thomas M. Coughlin and Farid Neema. Archiving Stakes Its Claim to Lower TCO. http://www.wwpi.com/