

ITStorage: A High Performance Internet Tape Storage

Xianbo Zhang, David Du

*Dept. of Computer Science and Engineering
DTC Intelligent Storage Consortium (DISC)
University of Minnesota, Minneapolis, 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 with diversified access patterns demands data storage to have different performance, cost and protection levels. ITStorage, an Internet Tape Storage system, is designed and prototyped to provide a cheap, “infinite” online data storage for data backups from personal users and small businesses, whose data are not as well protected as enterprise data. With Extended High Performance Tape File System (EHPTFS) as the technology core, data interleaving and tape switches are transparent to user applications while tape metadata and media information are consolidated in a commodity relational database. The performance evaluations show that ITStorage provides good online write performance through http interface even compared to a high performance SCSI hard drive, making it a good supplementary to disk-only solutions for holding infrequently accessed data backup/archive with reduced costs.

1 Introduction

When business and computer users rely on computer more than ever, valuable data loss in any form becomes increasingly unacceptable. Statistics shows that causes of data losses include 44% hardware or system malfunction, 32% human error, 14% software corruption, 7% computer viruses, and 3% natural disasters [1]. Each year computer users spend millions of dollars trying to recover their lost data. In general, 7 out of 10 small firms go out of business within a year if they experience a major data loss, according to UK Department of Trade and Industry estimates. Backing up data to offsite storage along with data encryption is an effective way to fight all these data losses.

Personal users and small to mid-sized organizations need a cheap, yet efficient, solution to protect their invaluable data just in case data loss/disaster should happen. However, personal users, small to mid-sized organizations can not afford the expensive solutions adopted by large enterprises. Internet storage (a.k.a. online storage, offsite or remote backup) offers a promising data protection solution

for individual computer users, almost any small businesses to backup their data remotely.

Unlike traditional data backup, Internet data backups are uploaded to an online remote server located at a secure data center, which is protected by the service provider following data protection practices, and can be accessed any time and anywhere when needed. It frees end users from managing the complex data backup process and physical storage medium, giving them peace of mind by storing data remotely and securely. The remotely stored data are usually mirrored to multiple sites with enough distance for enhanced data safety.

The explosive data growth, due to content-rich applications or regulatory requirements from HIPAA, the Sarbanes-Oxley Act or SEC Rule 17a-4, demands data storage with scalable storage capacity and affordable price. The diversity of growing data demands the diversity of online storage. However, almost all current online services are using disk as the primary data storage taking advantage of the random access nature from disk. For disaster recovery, tape is used by some service providers to backup their online servers storing customers' data backups. This practice leads to a relatively high system cost due to the expensive disk storage and backup process. Other service providers choose tape-free solutions, but suffer from high monthly energy bill in addition to the costly one-time disk purchase price. The everyday expenses limit the profit of online backup service and causes many online backup service providers to go out of business, downgrade their services or provide services with high monthly fees, which eventually hurt the online market. Inspired by the growing speed and capacity with lowering cost per GB from tape storage, we are proposing a new Internet backup/archive system based on tape storage. With the observation that data recovery operations happen much less frequently than data backup operations, and many data backups are so rarely accessed to be data archival, we believe the modern tape provides a good, cheap, yet effective, storage for personal users and small organizations to store their infrequently accessed data. As shown in [2], tape is getting faster, bigger (in capacity) and cheaper. By replacing disk with tape, we get “infinite” storage without the complexity

of disk-to-disk-to-tape (d2d2t) backup process or hierarchical storage management (HSM), and thus are expecting to have a reduced total cost of ownership (TCO) for online backup users.

However, a good “infinite” tape storage should provide the following characteristics that challenge a traditional tape technology:

- Concurrent write support to allow hundreds of users to write to the same tape drive simultaneously.
- Transparent tape switches to present users an “infinite” data storage.
- Adequate data retrieval time.

HPTFS, a high performance tape file system [3], provides a generic file system interface to access a physically mounted tape cartridge and transparently interleaves concurrent data writes to the tape. These features give us a good start point to build an online backup storage. In this paper, we focus on extending HPTFS to provide an “infinite” Internet tape storage (ITStorage) via transparent tape switch and efficient tape-resident data management.

For the rest of the paper, the online backup overview is given in Section 2. The tape based system architecture is described in Section 3. System component design and implementation are introduced in Section 4. Data restore approaches are discussed in Section 5. The performance of ITStorage is evaluated in Section 6. Conclusions and future work are finally given in Section 7.

2 Related Work

2.1 Online Backup Services

Many commercial Internet backups are provided as services nowadays [4, 8, 5, 6, 9, 10, 7, 11, 12, 13]. An Internet backup service allows end users to routinely backup their data to a secure and trusted storage on the Internet and recover their data from the storage when needed. Via a client- and/or browser-based application, standard backup features, such as full backups, incremental backups, data compression, and data encryption, are provided. Backups can be highly customized with extensive file filtering and policy scheduling. An advanced online backup agent (running on client side) may backup open files and various databases, or use continuous data protection (CDP) technique to provide end users real time data protection. Some service providers also support online file sharing, access control and file versioning.

Once the backup policy is scheduled, backup operations occur automatically without users’ attention. Backup logs and statistics are commonly available for users to verify a specific backup operation. For convenience, a user’s remote storage may be configured to appear as a local drive, and file drag-n-drop/copy-n-paste operations are supported accordingly. Since user data is stored on the service provider’s storage, ideally user data end-to-end encryption should be enforced to ensure data safety, i.e., data

are encrypted at the client site before transferring to the server site and remain encrypted on the server. However, many service providers do not provide real end-to-end encryption. Nearly all service providers offer user ID and password based authentication mechanisms to authenticate a log-in user.

A typical Internet backup is composed of an initial full backup and many incremental backups. A file level incremental backup may still causes the transmission of a large amount of data. For example, a minor change in a big file requires the entire file to be transmitted, which results in longer backup time, extra network utilization and more server storage space. To make online backup a viable backup choice for changing data, it is critical to only allow data changes in a file to be backed up instead of the entire file after a successful initial full backup. Two widely used backup innovations, delta blocking and binary patching process [14, 15, 16, 17], are briefly introduced in Subsection 2.2 even they are not applicable for fixed-content data storing on our proposed online tape storage.

2.2 Delta Blocking and Binary Patching

In essence, the delta blocking process breaks a file down into equal sized blocks and computes each block’s checksum, and then compares the checksum of each block from a modified file with the saved checksum of corresponding block from its previous version. Whenever the process finds a difference, it extracts a copy of the changed block into a delta file. The size of the delta file (cumulative size of these changed blocks plus location information) is usually less than the size of the original file. Backing up the generated delta file reduces the total backup size and corresponding transmission time. However, for delta blocking technology, change of only 1 byte in a data block results in that the entire block is backed up. Thus the size of a formed delta file can be significantly greater than the actual size of the changed data. Even worse, if updates in a block change the predefined block boundary, the rest blocks are considered changed blocks even what really happens there is just some data shifting. To identify this data shifting, Rsync [18] computes the weak, yet fast, 32-bit rolling checksum of each block starting at each byte of the considered file, for each checksum, it searches the checksum list from the previous version for a match. If a match is found, a strong checksum is computed to clear any false positive. If the strong checksum still matches, the data between the current file offset and the end of the previous match is considered changed data.

On the contrary, a binary patching process compares two different versions of the same file byte by byte and only extracts the changed bytes instead of blocks into a new file that is compressed into what is known as a patch file. The patch file is often 85% to 99.9% smaller than the original file which the patch was extracted from [14]. Since two versions of the same file have to be compared at the user site, the previous version of the changed file has to

be saved before any change is committed, which consumes more local disk space than delta blocking process that only stores checksums from the previous version of the considered file. The high efficiency of binary patching makes it possible to backup data even from slow dialup connections. Binary patching technology has been widely used by many hardware and software companies to distribute their software updates.

2.3 Disk based Online Storage

Online storage focuses on providing data storage online for convenient access and data protection and attracts big companies like Yahoo, Microsoft and Google. Microsoft plans to provide users with “Live Drive” while Google plans to provide “Gdrive”. Both “Live Drive” and “Gdrive” are in their early developing stage and are planning to provide users an unlimited amount of online storage [19, 20]. Considering the energy used to keep thousands of drives rotating, it is not wise to use these disks to hold “static” data from many content-rich applications. Comparing to disk, tape does not consume any energy if it is not being used.

3 Proposed System

The system framework and components are first introduced in this section. And then the focus is on ITStorage, an Internet tape storage that provides an “infinite”, cost-effective, high performance data storage for online backup. Compared to disk based online storage, ITStorage provides a much higher storage capacity based on automatic tape library (ATL). For example, a Sun STK SL8500 modular library system loaded with 1448 STK T10000 tape cartridges can easily provide 724 terabyte storage ($1448 \times 500\text{GB}$). The system can scale to 300,000 slots with 150 petabyte storage ($300,000 \times 500\text{GB}$) [21].

3.1 System Framework

The framework of the proposed tape-based online backup system is presented in Figure 1. A group of web servers are behind a load balancer to provide a scalable system performance with evenly distributed loads. The HPTFS described in [3] is extended to present an automated tape library as an infinite data storage interleaving multiple writes to a tape drive and hiding tape switch operation from data applications. The tape storage is accessed through generic file system API’s from inside (within the system). Tape metadata is stored on individual tapes and in centralized MySQL database as well. The tape library robot, tape drive and tape media information are also managed within the same database. From outside, the tape storage is accessed through http protocol by hundreds of users simultaneously across Internet. Thanks to the Extended HPTFS (EHPTFS) hiding tape storage from applications,

the http server can be easily replaced with other file transfer servers, like ftp server, psftp server, scp server, etc, without complex server re-engineering work. With EHPTFS, we can easily change to any file transfer server with proven reliability, scalability and high performance. In fact, any high performance commodity http server can be used in our prototyping.

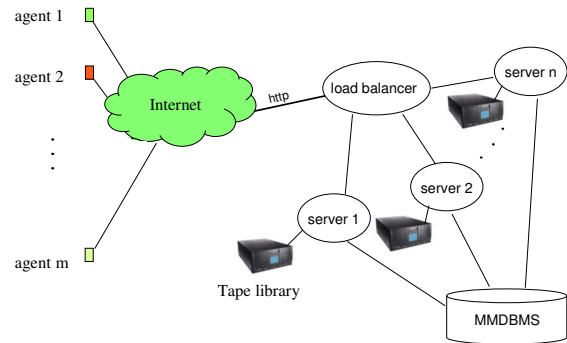


Figure 1. System big picture

Figure 2 shows the details of ITStorage. We briefly describe the system components as follows.

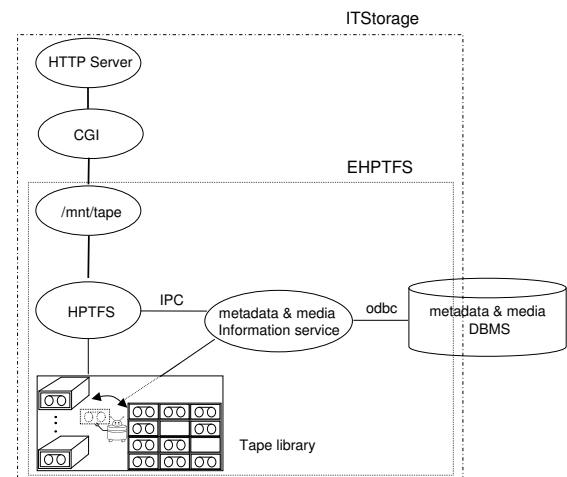


Figure 2. Components of a data server

3.1.1 Advanced Agent

The Advanced Agent running on client side is one of the most important components for a successful backup system. For our proposed system, the Advanced Agent needs to enforce backup policy, understand some application specific semantics, and handle data compression, encryption and transfer. End-to-end data encryption is enforced for data security, i.e., user data is encrypted with symmetric encryption algorithm at user side before it is sent across network to http server. Only the user knows the encryption key. Key management is always a challenge for any information system and is not addressed here. After a proper authentication process, the Advanced Agent can store files to ITStorage and retrieve a specific version of a stored file.

3.1.2 Server Load Balancer

This server load balancer balances data connections among web servers. When a new data connection comes in, it selects a server to serve the data write using round-robin algorithm or other load balancing algorithms. However, this load balancer has no effect on data retrieval. The server serving a specific data request is determined by the location of the requested data.

3.1.3 HTTP Web Server

HTTP (Hyper Text Transfer Protocol) is one of the most widely used data transfer protocols, which provides a generic tunneling for data transfer even across firewalls. Nowadays almost all computers have a web access and most firewalls have the port number 80 open. Large amount of data transfer is not a surprise for a data backup, and the HTTP server has to handle hundreds of data transfer connections with high performance and use as less system resources as possible.

3.1.4 CGI Module

A CGI (Common Gateway Interface) file is an executable file stored on the web server host, which is started by the http server as a separate process to handle a user request. In ITStorage system, each started CGI process receives uploaded files and writes them to the tape storage mount point – /mnt/tape. It is common to have multiple CGI processes concurrently write data to the same mount point. These data writes are written to tape storage by the Extended HPTFS (EHPTFS).

3.1.5 High Performance Tape File System (HPTFS)

HPTFS [3], as shown in Figure 3, explores a new way to federate tape storage into the current hierarchical storage system. The system bears the following features:

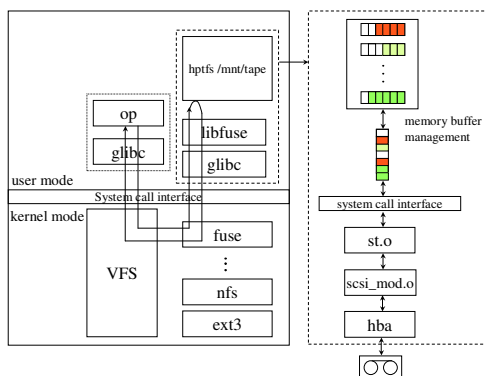


Figure 3. Architecture of HPTFS

- Providing a generic file system interface for applications to access tape storage. Applications based on file system interface can write and read tape data without any modification.

- 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.

- Sharing tape media among multiple users. The system supports concurrent writes for maximum write performance with 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.

- 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.

3.1.6 Extended High Performance Tape File System (EHPTFS)

EHPTFS, an Extended HPTFS as shown in Figure 2, is composed of modified HPTFS, Metadata and Media Information Service (MMIS), a metadata and media database management system (MMDBMS) and an automated tape library. Special efforts have been made to ensure that the high performance of HPTFS is not undermined by component interactions within EHPTFS. The component interactions among the modified HPTFS, MMIS and MMDBMS are shown as linked lines between them in Figure 2. The high performance of HPTFS is mainly reserved by MMIS' proactive monitoring and computation that makes information required by HPTFS locally available through shared memory (Inter-Process Communication) with little overheads. In addition to HPTFS functions described in [3], such as providing a mounted tape cartridge as a shared data storage with generic file system API access, EHPTFS enforces load balancing among tape drives attached to the same web server host whenever a new data connection is coming, performs tape switch when a mounted tape is full, and keeps MMDBMS database up-to-date with fresh metadata and media usage information. In all, EHPTFS presents an automated tape library as an "infinite" data storage to applications assuming tapes full of data can be swapped out of tape library while blank tapes can be swapped in.

3.1.7 Metadata and Media Information Service (MMIS)

As a separate process, MMIS is running on the same host as web server and caches data from the modified HPTFS before sending to MMDBMS. To reduce overheads, MMIS and HPTFS communicate through a shared memory. The race condition for the share memory is

avoided through a host wide semaphore. MMIS is necessary to hide the TCP/IP delay from the modified HPTFS and help it achieve the same performance as the original HPTFS. MMIS periodically runs a load balancing algorithm to find the right drive for new data connection. The computed result is updated in the shared memory and readily available for HPTFS whenever a new data connection comes. Some design details of MMIS is given in Subsection 4.4.

3.1.8 Metadata and Media DBMS (MMDBMS)

MMDBMS is implemented with open source MySQL database management system providing database management functionality for metadata, tape cartridge media information and other real time operation status information. Operation status, which provides inputs for load balancing among drives attached to the same host as where EHPTFS is running, includes data connection status, data transfer size versus time information, connection allocation for each drive, etc. As mentioned earlier, load balancing calculation only affects new data connection session. A data connection sticks to the chosen drive to limit the number of tapes involved in storing files from the same connection. Doing this reduces the number of tape switches for future data retrievals. In addition to the metadata described in [3], the metadata managed in MMDBMS also includes tape library and media volume information as well as data owner information. MMDBMS also maintains tape cartridge location information used by robot for tape switch operation. File versioning is easily supported within MMDBMS by associating a time stamp with each stored file.

3.2 ITStorage: A High Performance Internet Tape Storage

With transparent data interleaving and tape switch performed by EHPTFS, the “infinite” tape storage of an Automatic Tape Library (ATL) is available to data applications through generic file system interface APIs as shown in Figure 2. Within ITStorage each tape still has its own tape metadata stored as in HPTFS for portability. Meanwhile, tape metadata from all tapes and media volume information are consolidated into MMDBMS. A user file metadata is represented as a table record. Associated with a file creation time stamp, multiple versions of the same file is readily stored in ITStorage. Having EHPTFS work as the core and http server provide access interface, ITStorage provide an “infinite” data storage to applications across network. In the following we introduce some main operations of ITStorage.

3.2.1 Storage Bootstrap

Assuming no tape is mounted in tape drive at the earliest moment. MMDBMS is started first. And then MMIS is

started to communicate with MMDBMS to get tape media information and the configuration of robot and drives attached to the running host (Assuming a proper configuration has been stored in MMDBMS). Based on received information, MMIS instructs the robot to fetch a chosen tape cartridge from a specific location and insert it into a chosen tape drive. After all tape drives are loaded with tape cartridges, HPTFS is started to mount them as a file system at a given mount point such as /mnt/tape. Finally, http server is started to present http access interface to outside. With the help of HPTFS, http server writes data to or reads data from tape without the knowledge of tape media.

3.2.2 Storage Interface

Currently http server is running to provide web access. However, as mentioned before, http server can be easily replaced with other file transfer server, such as ftp server, scp server or psftp server, to provide other corresponding interfaces. It is worth mentioning that the EHPTFS mount point can be readily exported to other hosts through Network File System (NFS) since EHPTFS is a real file system supporting all necessary file system calls. In Section 6, the file transfer efficiency comparison between http and ftp is evaluated.

3.2.3 File Write within One Tape

When a client uploads a backup file using http protocol, the http server, a component of ITStorage as shown in Figure 2, starts a CGI process to process the file upload request. After a proper authentication process the CGI process starts to create a file under the mount point (a file directory) for future data writes. Once receiving the file creation request, EHPTFS chooses a tape drive based on the drive usage calculation from MMIS and associates the created file handle with the chosen tape drive. Which drive to choose is determined by the drive available bandwidth. Among all the available drives, the one with the maximum available bandwidth, the difference between the tape drive rated bandwidth and the tape drive current used bandwidth, is chosen assuming not all tape drives reach their rated maximum speed. If all tape drives have reached their maximum speeds, a random number is used to determine which one to choose. A file upload connection sticks with the chosen drive until all files are received. A data write operation path is shown in Figure 4 as 1→2→3→6→7→8→9. The write operation of EHPTFS works almost the same way as HPTFS described in [3] except that the modified HPTFS notifies MMIS of the number of bytes written and the corresponding write finish time for each tape write operation through a shared memory. Since a big memory buffer is maintained for each file handle and a tape write operation occurs only after the buffer is full, the number of notifications is not so big and controlled by the network connection speed and the used memory buffer size. These running status data are used to compute tape drives’ cur-

rent write speed that is the basis for the tape drive chosen algorithm.

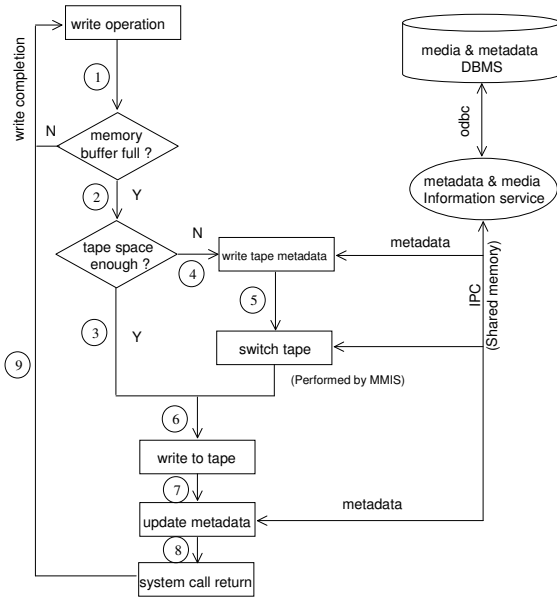


Figure 4. Write into infinite data storage

3.2.4 File Write across Tapes

Since the number of tape drives within a tape library is much less than the number of tape cartridges for economical reasons, tape switch is a must operation for an automatic tape library, which is one of the main differences between hard disk storage and tape storage. Each hard disk has its one drive performing its data write/read operations. Providing a transparent tape switch operation is critical for an “infinite” data storage system. The operation path for write across tapes is shown in Figure 4 as 1→2→4→5→6→7→8→9. The actual tape switch time is determined by the performance of the used robot ranging from several seconds to more than ten seconds. Using a bigger memory buffer to temporarily hold received file data writes can reduce the effect of tape switch. For the same amount of data, the bigger the capacity, the less frequently the tape switch operation. As shown in Table [2], the tape cartridge capacity is increasing amazingly, we are expecting the tape switch operation will not have a big impact on the whole system performance.

4 Design and Implementation

High component performance and efficient component integration/communication are essential to achieve a high system performance. Some of the design considerations and choices are described in the following Subsections.

4.1 Choice of Web Server

Conceptually, a web server creates a socket and binds the socket to the well-known port at which it waits to receive requests. The server then enters an infinite loop in which it accepts a request initiated by a client, processes the request, formulates a reply, and sends the formulated reply back. Based on their design and implementation differences, web servers are classified into four generations as follows [22, 23].

The first generation of web servers were written to handle requests one at a time. Their users noticed that while the server was servicing a request from some other user, they could not get their own requests serviced. An unpredicted pause is inevitable for such a simple implementation.

The second generation of web servers took a straightforward attack on this problem by forking off a child process for each request. Instead of an unpredicted pause, a constant startup delay is introduced for each request due to the relatively expensive forking operation. Each started process occupies certain amount of system resources, like memory and CPU time, and the number of concurrent child processes is limited due to the resource constraints. After serving a request, the child process goes away. *CERN* and *NCSA 1.3* falls into this generation. To reduce the overhead of forking a process, “lightweight processes” (LWP) or “threads” are used to replace processes on some operating system platforms.

The third generation of servers have a pool of child processes/threads that are kept alive to serve coming requests. *NCSA 1.4*, *Apache* and *Netscape Netsite* belong to this group. These servers have great performance with a complex implementation, and can handle much more connections per second than the second generation servers. The implementation of this type of servers usually take advantage of the unique feature from a specific operating system.

The fourth generation of web servers have just one process to handle concurrent requests through non-blocking I/O and `select()/poll()/kqueue()` calls. *Spinner*, *Open Market*, and *thttpd* belong to this generation. Memory use of this type of web server is great. The performance is reportedly good under heavy loads.

It is not an easy task to design and implement a high performance, secure web server. Since there are so many open source web servers available, we decide to choose one instead of developing a new one. After some comparison, we chose *thttpd* as our system web server providing http interface to user applications since *thttpd* is an open source server implemented in just a little more than seven thousand lines of code and it is much easier for us to do any modification if needed. Comparing to *Apache*, *thttpd* provides just enough features for our needs and uses less system resources that is important to us.

4.2 Efficient CGI Implementation for File Upload Requests

The Common Gateway Interface (CGI) [24] is a standard for external gateway programs to interface with information servers, such as HTTP servers. A CGI code called by the HTTP server was referred to as a CGI script due to the fact that many CGI applications were written in script languages such as Perl, UNIX shell script, etc. As the popularity of the web grew and the need for dynamic content increased, CGI applications written in languages other than script languages became more and more popular. However, these applications are still referred to as scripts. In this paper, the terms script and application are used interchangeably for convenience.

To meet the growing use of web applications, many free or commercial packages/libraries are developed to ease CGI implementation. They make CGI request parsing and http response creation much easier, and CGI developers do not have to know all the details of http protocol. The popular Perl CGI package provides a simple, convenient way to handle a file upload request. File upload, a POST method, is the CGI method web browsers use to transfer data from client side to server side. Popular web browsers such as Internet Explorer and Firefox all support file upload method.

The file upload CGI script in Perl uses just several lines of code to store an uploaded file onto server side storage. However, Perl CGI package creates a temporary file on hard disk and parses the temporary file to extract the real uploaded file. This approach gets disk involved into the data path to tape, which reduces the performance of the whole process and is against our design principle of putting data directly onto tape storage. After some experiments with the CGI implemented in Perl, we have to give it up due to its poor performance although we love its simplicity. *cgi* [26] and *cgicc* [25] are CGI C function library and CGI C++ class library respectively. However, function calls from *cgic* or class methods from *cgicc* do not parse a user request until all the request data is received into the main memory. This creates a big pressure on the system memory when the CGI is dealing with file upload and puts a limit on the file size a user can upload. This approach is not acceptable for our system since we do not want set a hard limit to the size of a file the user can upload. The performance of handling just several concurrent uploads with each in the amount of 300 MB is terrible for a computer with 512MB main memory. We come to the conclusion that these packages/libraries can not meet our performance requirements. We studied the http upload method to implement a file upload CGI to store received data directly onto tape by writing data to the EHPTFS mount point, which improves system performance with a reduced data path and less required system resources.

4.3 Tape Switch Operation

Tape switch operation is an inevitable operation in an Automated Tape Library (ATL) due to the nature of ATL: the number of tape drives is much less than the number of tapes. For an online backup system based on tape, it is critical to perform a seamless, "hot" tape switch. Due to the data compression nature used by many tape drives, it is difficult to predict whether a given file can be stored by a considered tape. Thus it is possible for an uploaded file to be split onto across tapes. There are some other legitimate reasons to have a file split across tapes, like not wasting tape space, a file is too big to be held by a tape and so on. When splitting happens, a tape switch should be transparent to the client, and data loss/retransmission in any format should be avoided. After sending out the tape switch request to MMIS, HPTFS sets a tape switch flag. Seeing the flag, all threads of HPTFS slow down their data writing processes. Accordingly CGI processes slow down their data receiving processes resulting in that http server slows down its request processing speed and eventually clients slow down their file uploading speeds due to the TCP/IP protocol. Once the tape switch operation is completed and the tape switch flag is cleared, HPTFS, CGI processes and the http server will soon resume their speed, and gradually clients will speed up their file uploading speeds. This approach has the advantage of simplicity. Everything happens naturally without any additional care. As mentioned before, the increasing tape capacity reduces the frequency of tape switch operations, and tape switch is not expected to have a big impact on the whole system performance. Alternatively, disk can be used to temporarily hold all coming data during the tape switch process without slowing down clients' uploading speeds. However, the disk staging process increases the system complexity and may slow down the system performance if the disk staging cannot go away since the aggregate speed of clients' uploading equals the tape drive speed.

4.4 Metadata and Media Information Service (MMIS)

The information concerning tape cartridge, tape data location and tape library robot is maintained by MMDBMS and is used to determine where and how to store/retrieve a file within ITStorage. MMIS, a component of EHPTFS, is running as a daemon process on the same host as HPTFS and provides HPTFS services to deal with metadata and media operations by interacting with MMDBMS and robot. MMDBMS is usually running on a separate host and may be shared by multiple MMIS instances running on different hosts. To reduce the overheads of network delay and database operation, HPTFS talks to MMIS instead of MMDBMS. Once the data that need to be committed in MMDBMS are stored in the shared memory between HPTFS and MMIS, HPTFS considers the operation has been committed within MMDBMS and con-

tinues its work. MMIS periodically communicates with MMDDBMS through ODBC (Open Database Connectivity) over TCP/IP to perform database operations. MMIS also proactively computes tape drive bandwidth usage based on data write information from HPTFS and stores the results in the shared memory. The computed results are used by HPTFS to choose a drive with the maximum available bandwidth to handle a new data connection. A tape cartridge is reserved in the database for a drive by MMIS when the drive's mounted tape is close to be full with data. Once receiving the tape switch request, MMIS issues a series of commands to instruct the robot for tape switch operation without any delay. This whole process hides the potential overheads of network and database from HPTFS and makes sure that the modified HPTFS performs as well as the original HPTFS. The tape switch operation is implemented in MMIS instead of HPTFS with two considerations: (1) Any media operations, unrelated to data, should not be handled by the modified HPTFS leaving it focus on handling file system calls to keep its high performance; (2) Keeping HPTFS from being involved into network communication to keep a clean design. The modified HPTFS may run on a host attached with drives but no robots. As shown in Figure 5, drives hosted in the same automatic tape library are connected to different Host A and Host B while the only robot is controlled by Host B. Host A has to communicate with Host B to carry out any necessary robot operation.

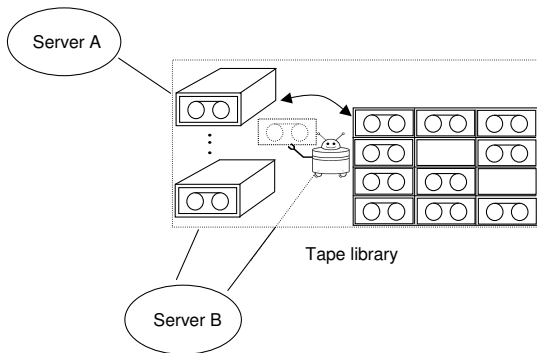


Figure 5. Tape library robot shared between two servers

4.5 Metadata and Media DBMS (MMDDBMS)

MMDDBMS is MySQL based information database management system. Some table structures in MMDDBMS are listed in Table 1 to Table 4.

User authentication table, as shown in Table 1, maintains system wide unique user identification and SHA-256 hash result of user password. Through a secure channel, user provided *userID* and *password* are verified based on a corresponding table record. The SHA (Secure Hash Algo-

Table 1. User authentication (userat)

Field name	Data type
userID	CHAR(128)
SHA256PWD	UNSIGNED CHAR(36)
userquota	UNSIGNED INTEGER
...	...

Table 2. File information (fileinfo)

Field name	Data type
userID	CHAR(128)
fname	CHAR(256)
fsize	UNSIGNED INTEGER
ctime	UNSIGNED BIGINT
barcode	CHAR(32)
sp	UNSIGNED MEDIUMINT
ep	UNSIGNED MEDIUMINT
fragno	INTEGER
...	...

rithm) family is a set of related cryptographic hash functions. No real attacks have yet been reported on the SHA-256. File metadata table, as shown in Table 2, maintains metadata for each stored tape file, like the location of a file (including tape cartridge barcode, *sp* - start position and *ep* - end position and so on). The file creation time (*ctime*) on tape is important for file versioning. The combination of *userID*, *fname* and *ctime* identifies a particular version of a specific file. If there are multiple records with the same user ID (*userID*), file name (*fname*) and creation time (*ctime*), these records indicate that the considered file spans multiple tapes and the lowest *fragno* indicates the first part of the file and the highest part indicates the last part of the file. Since the capacity of a tape cartridge is in the amount of 100's GB, we expect rare files span multiple tapes because of their sizes. However, files may span multiple tapes due to the tape switch operation as mentioned before. If EHPTFS is configured to write to disk and tape simultaneously, a file spanning two tapes can be easily avoided when the file size is relatively small (like less 200MB) and the spanning is caused by a tape switch operation. This can be realized by deleting the corresponding metadata entry from the unmounting tape and then moving part of the file that has been written into disk to the newly mounted tape. The reason of using disk in the system is for data retrieval as discussed in Section 5. Basically, a request asking for data residing on a tape cartridge in write mode can be served by disk instead of tape. Changing tape from write mode to read mode is expensive due to tape's costly repositioning. Drive information table, as shown in Table 3, maintains information for each tape drive, like its responsible robot, the drive host, etc. Field *stat* indicates the drive status, like up, down, tape loaded or not, and so on. Field *barcode* records the barcode of the tape that is physically mounted in the drive. *MBperSec* tracks the ac-

Table 3. Drive information (drvinfo)

Field name	Data type
drvhost	CHAR(256)
drvpath	CHAR(256)
robothost	CHAR(256)
robotno	INTEGER
robotpath	CHAR(256)
stat	SMALLINT
barcode	CHAR(32)
MBperSec	UNSIGNED MEDIUMINT
rate	UNSIGNED MEDIUMINT
...	...

Table 4. Media information (mediainfo)

Field name	Data type
barcode	CHAR(32)
capacity	SMALLINT
stat	SMALLINT
slotno	UNSIGNED INTEGER
availspace	SMALLINT
robothost	CHAR(256)
robotno	INTEGER
...	...

tual speed of a drive while *rate* records its maximum rated speed from its specification. Tape cartridge table, as shown in Table 4, maintains the tape cartridge's responsible robot, tape location, tape capacity and available space, etc. For a mounted tape in write mode, part of these mentioned fields are periodically updated by MMIS.

4.6 Resource Locking Mechanism

Operations of a robot have to be serialized to avoid robot malfunction or potential hardware damages. The serialization is realized by the MMIS running on a host with the robot attached. As shown in Figure 5, the MMIS running on Server B is the MMIS responsible for issuing robot operations. The MMIS running on Server A can only request for a robot operation through the MMIS running on Server B. Concurrent writes to a drive have to be serialized to avoid any data corruption. Each drive has its own locking object to improve write performance by allowing write parallelism across drives. Conflicting accesses to the shared memory between HPTFS and MMIS also need to be serialized to ensure correct communications between the two components. For all cases, special attentions have to be paid to avoid any potential deadlock.

5 Data Backup Retrieval

Intuitively, tape storage gives rise to the concern over data restore performance due to its sequential access nature. However, after looking deep into the stored data and

possible restore requests, we found that the sequential access may not play a big role in the whole restore process. Especially for data archival, they may not be accessed once during its lifetime. What's important for this type of data is that they are protected and can be retrieved if needed. The retrieval speed does not play a big role here.

A restore request could come for a tape that is performing data writes, which is not good for a sequential access tape media. For this situation, data read operation is expensive considering the series of tape drive repositioning. The tape write operation has to be stopped and the tape has to be rewound to the proper location before the read request can be served. To mitigate this issue, EHPTFS can be configured to write to disk and tape simultaneously. When the tape cartridge is in write mode, data read request is served by disk instead of tape. Thus EHPTFS supports read while write (RWW). To reduce the storage capacity requirement for disk, we only require the disk capacity match with the capacity of physically mounted tape. Once the tape is full and unmounted, corresponding disk data may be removed and disk is ready for a new mounted tape.

The restore request may ask for a small amount of data for unmounted tape full of data. Once the tape is mounted, the high speed tape drive can easily saturate the network connection even the requested data is interleaved with other data. Except for a slow startup, the data restore bottleneck will be the network connection instead of the tape drive. If concurrent data restore requests are received, a proper data retrieval scheduling may mitigate the tape sequential access nature [27, 28, 29, 30, 31]. If multiple requested data happen to be interleaved on the same tape, one tape drive read pass may serve them all together. For this case, tape is first positioned to the start position of an object whose first block is closest to the tape beginning, and then starts to read data and de-interleave data to serve multiple read requests simultaneously. During the data request serving period, new requests may come. A dynamical scheduling algorithm is needed to improve data read performance as a whole.

If a restore request involves huge amount of data, such as required by a data disaster recovery operation, it is better to restore the requested data to DVD's or other portable physical storage media, and ship them to the customer site through an express service. This physical data delivery is the only choice no matter the data backup is stored on disk or tape at the service provider's data center. If the requested data is in the amount of 100's GB, it is better to use commercial express delivery service to move data to the client site instead of the Internet. The Internet speed is not fast and reliable enough to handle 100's GB data. In fact the wide area network (WAN) speed has never grown fast enough to match the disk or tape density growth rate [32]. Requested data can be dumped to portable media such as tape, USB disk or RAID system and then be shipped to client site within one business day. The same idea is actually being used for data backup across network by IBBackup

as described in Section 2.

6 Performance Evaluation

The test configuration is shown in Figure 6, where each link has the speed of 100Mbps and each client has a Foxfire web browser (version 1.5.0.4).

The server has

- Four 3.2GHz Intel(R) Xeon(TM) CPU
- 3GB main memory
- one FUJITSU MAT3073NP SCSI high performance hard drive (10,000 RPM, 132MB/s internal data transfer rate)

The attached tape library is Exabyte 221L that hosts one LTO-1 tape drive. The LTO-1 drive has

- 100GB native capacity
- 15MB/s native data transfer rate
- 64MB memory buffer

The tape drive is rated as 37.854 MB/s with a standard deviation of 0.003 MB/s by using command *dd* to send 4GB zeros from memory directly to the tape drive. If 1GB is used, the drive only reaches 35.762 with a standard deviation of 0.006 MB/s. This setting is used to simulate a real environment where the aggregate client bandwidth is greater than the bandwidth of the host to which the tape drive is attached. It is worth mentioning that the link speed to the server (100Mbps is about 12.5MB/s without considering the TCP/IP overheads) much less than the drive speed when the drive compression is turned on. Thus the performance evaluation is focused on the worst case for ITStorage where the tape drive cannot work in streaming mode.

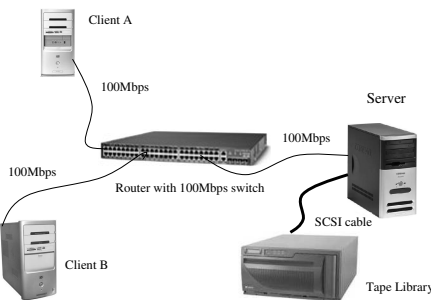


Figure 6. Performance evaluation setting

We first evaluate the performance of ITStorage with client(s) sending data across the network. And then we evaluate the system performance when tape is replaced by disk with the same client(s) sending data across the network. For data transfer protocol comparison we also evaluate the system performance with http replaced by ftp. The

file transfer protocol ftp is supposed to be a better protocol for file transfer. And finally we get the maximum performance of ITStorage by uploading files from the local SCSI hard drive. The bandwidth of the SCSI drive is higher than the tape drive speed.

Among the following tests, each case is repeated 20 times and the memory buffer for each opened file is allocated as 80MB in the modified HPTFS. The uploaded files are filled with the same character “A” to the specified sizes. With tape drive compression turned on, a tape drive is tested as a drive with higher native transfer rate via using this type of files. Table 5 shows the performances of ITStorage, where uploaded data is written to tape directly. The EHPTFS is configured to write to tape only. As shown in the table, when just one client uploads a 50MB file, the system reaches the maximum speed among measured speeds that is close to the network speed considering the TCP/IP overhead. This is not a surprise since the allocated main memory buffer is 80MB, ITStorage can hold all the uploaded data before writing to tape. Thus only one tape drive repositioning is required for the whole data write. For the uploaded 100MB file, it requires two repositioning since the size of 100MB fills the ITStorage data connection buffer twice and there is a time gap between the two tape writings. Similarly, the 150MB file requires two tape repositioning but the second write carries 50MB more data compared to the 100MB file. Thus we see the worst write performance happens to the 100MB file among the 50MB file, 100MB file and 150MB file as clearly shown in Table 5 and Figure 8. With two concurrent uploads, the measured system speed is pretty stable – above 10MB/s – as shown in the table. Considering the network fluctuation, the effect of an uploaded file size can be ignored.

Table 5. ITStorage write performance (MB/s, tape block size=256KB, 100Mbps network)

File size (MB)	1 Upload		2 Concurrent Uploads	
	Mean	Stdv	Mean	Stdv
50	11.500	1.369	10.833	0.556
100	8.485	0.339	10.002	0.002
150	8.754	0.185	10.345	0.001
200	8.286	0.126	10.256	0.002
250	8.065	0.001	10.136	0.118
300	8.288	0.002	10.481	0.091

Table 6 shows the performances of ITStorage when tape is replaced with disk, i.e., data is uploaded through the same web browser from client side and is directly written to disk on the server side. As shown in the table, two concurrent uploads achieve a little higher speed. This may be caused by the network bandwidth allocation difference for one connection and two connections from the network router. Figure 7 shows the system performance comparison between disk and tape, where the performance data are

from Table 5 and Table 6. Among all the tested cases with the chosen configuration, tape is slightly slower than the SCSI disk at most time. The maximum performance difference of about 19% happens with the uploading of a single 300MB file. However, for two concurrent file uploading, the maximum performance difference is only about 10%. It is worth mentioning that tape achieves the maximum performance and is higher than disk when the size of the single uploaded file is 50MB. In reality, concurrent uploading happens more often than a single uploading. With compressed incremental backup, data transmission in the size of 10's MB is expected to happen more frequently. Considering that the tested tape drive is LTO-1, a several-year-behind drive, we believe replacing disk with tape will not introduce any big performance overhead if it does exist.

Table 6. ITStorage write performance with tape replaced by disk (MB/s, 100Mbps network)

File size (MB)	1 Upload		2 Concurrent Uploads	
	Mean	Stdv	Mean	Stdv
50	10.500	1.118	11.111	0.001
100	10.444	0.609	11.111	0.001
150	10.286	0.391	11.111	0.002
200	10.421	0.235	11.111	0.001
250	10.083	0.186	11.111	0.001
300	10.207	0.189	11.111	0.003

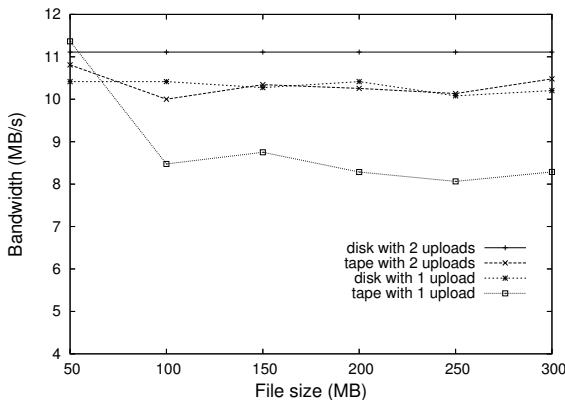


Figure 7. Performance comparison between tape media and disk media (100Mbps network)

Table 7 shows the performances of ITStorage with http replaced by ftp, i.e., file is uploaded using ftp protocol. The speed comparison between http and ftp is shown in Figure 8, where the average speeds from Table 5 and Table 7 are used. As shown in the figure, ftp has a little higher

speed than the corresponding http. Calculations show that the maximum difference between among all counterparts is less than 8%, which happens with the uploading of a single 150MB file.

Table 7. ITStorage write performance with http replaced by ftp (MB/s, 100Mbps network)

File size (MB)	1 Upload		2 Concurrent Uploads	
	Mean	Stdv	Mean	Stdv
50	11.111	0.001	11.042	0.064
100	8.910	0.006	10.027	0.730
150	9.528	0.012	11.101	0.057
200	8.897	0.036	10.427	0.402
250	8.562	0.001	10.498	0.450
300	8.899	0.015	10.528	0.593

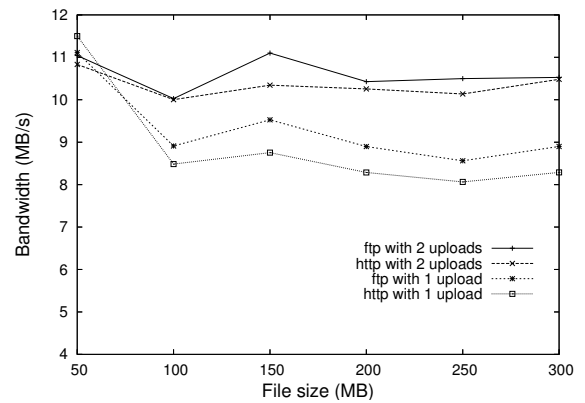


Figure 8. Performance comparison between http and ftp (100Mbps network)

Table 8 shows the maximum ITStorage write performance when files are uploaded from the local SCSI hard drive. As shown in the table, the tape drive can almost be streamed, which indicates the system integration is satisfactory.

We also evaluate the effect of memory buffer allocated for each connection in HPTFS. We changed the buffer size from 1MB to 80MB, and found the buffer size has little impact on ITStorage performance. Since the tape drive also has 64MB memory buffer, this is expected. The system overall performance is affected by the combination of HPTFS buffers and the tape drive buffer.

7 Conclusions and Future Work

A high performance data storage with scalable capacity is the key to an online backup service when backup/archive

Table 8. ITStorage maximum write performance(uploading files from local disk to ITStorage with 2 concurrent writes)

File size (MB)	Write speed (MB/s)	
	Mean	Stdv
300	36.012	1.274
400	32.444	0.770
500	34.510	1.191

data is increasing with an explosive growth rate due to content-rich applications and compliance requirement for data retention. Providing personal users, small business and enterprise remote offices a cheap, yet efficient, data backup solution is the goal of this research work. In this paper, we have described the design and implementation of ITStorage, a High Performance Internet Tape Storage for online backup, which provides tape storage from an automatic tape library as an “infinite” online storage for applications to backup data across network via http protocol. EHPTFS, an Extended High Performance Tape File System, works as the core of ITStorage, handles data interleaving and tape switch transparently. It provides a generic file system interface for applications to write data to tape with high performance. Corresponding metadata, tape media information and other management information is stored in a commodity relational database management system that provides a convenient database interface to handle metadata operation.

ITStorage takes advantage of the ubiquitous web access, and the high capacity and high performance of the modern tape storage technology. Through the universal http protocol, user data are directly written to tape without staging disk. Our experiments show that ITStorage write performance achieves more than 92% of the performance of a high performance SCSI hard drive with two concurrent Internet data connections even the tape drive is not working in streaming mode. With high speed data connection, the write speed of ITStorage is close to the rated tape speed. Proactive database operations and efficient component communications make EHPTFS work as efficiently as the original HPTFS. Online backup services built over ITStorage can provide an “infinite”, cheap, yet efficient data storage to protect the explosively growing data.

As discussed in Section 5, tape data read may not be an issue for data recovery. However, data reads from tapes with interleaved data leaves a lot of space for further optimization. Improving data retrieval concurrency is a promising way to improve the data restore performance. Real-time data retrieval scheduling and potential data re-assembling are required to achieve a reduced tape pass. A reduced tape pass indicates a higher data read speed. For example, when File A is being retrieved from Tape T_1 , the system receives file retrieval request for File B, and File A

and B happen to be interleaved on Tape T_1 . Reading file A and B simultaneously, which has to de-interleave File A and B on the fly, is beneficial to the data retrieval process. However, due to the file request timing difference and the tape file alignment difference, parallel retrieval may have to cache part of File B on disk or memory before data block reassembling happens. We plan to introduce the data read parallelism into ITStorage to support concurrent on-line data retrievals.

References

- [1] Remote Data Backup Software. <http://www.protect-data.com/information/statistics.html>
- [2] *Information Storage Industry Consortium (INSIC) Tape Roadmap 2003*. <http://www.insic.org>
- [3] X. Zhang, D. Du, J. Hughes and R. Kavuri, *HPTFS: A High Performance Tape File System*, In Proceedings of the 14th NASA Goddard Conference on Mass Storage Systems and Technologies/23th IEEE Symposium on Mass Storage Systems (MSST2006), pp.275-288, May 15-18, 2006, College Park, Maryland USA
- [4] EVault Inc. <http://www.evault.com>
- [5] Intronis Technologies. <http://www.intronis.com>
- [6] U.S. Data Trust Corp. <http://www.usdatatrust.com>
- [7] Iron Mountain Inc. <http://www.livevault.com>
- [8] Amberwave Communications LLC. <http://www.firstbackup.com>
- [9] Officeware Systems. <http://www.filesanywhere.com>
- [10] Pro-Softnet Corp. <http://www.ibackup.com>
- [11] Xdrive LLC. <http://www.xdrive.com>
- [12] AmeriVault Corp. <http://www.amerivault.com>
- [13] NovaStor Corp. <http://www.novastor.com>
- [14] NovaNet-WEB FastBITTM Patching Process. <http://www.online-backup.com/datasheets/bpt.html>
- [15] Smartways Technology Ltd., *FastBIT Binary Patching vs. Block Technology: A Comparison of New Incremental Backup Technology*, 2005. <http://smartstore.smartways.net/index.asp?200932>
- [16] <http://www.novastor.com>
- [17] <http://www.datavaultcorp.com/data/fastbit.shtml>
- [18] A. Tridgell and P. Mackerras, *The rsync algorithm*, TR-CS-96-05, The Australian National University.
- [19] B. Charny, *Google Continues Drive for Unlimited Storage*, March, 2006. <http://www.eweek.com>

- [20] M. J. Foley, *Microsoft Readies Storage Service to Rival Google's 'Gdrive'*, April, 2006. <http://www.microsoft-watch.com>
- [21] Specifications. Technical report, StorageTek. <http://www.storagetek.com>.
- [22] ACME Laboratories. <http://www.acme.com/software/thttpd>
- [23] D. Comer and D. L. Stevens, *Internetworking with TCP/IP Volume III: Client-Server Programming and Applications Linux/POSIX Socket Versions*, Prentice Hall, 2001
- [24] The Common Gateway Interface Specification. <http://www.w3.org/CGI/>
- [25] CGI C++ class library. <http://www.cgicc.org>
- [26] <http://www.boutell.com/cgic>
- [27] J. Li and C. Orji, *I/O scheduling in tape-based tertiary systems*, Journal of Mathematical Modelling and Scientific Computing, vol. 6, 1996.
- [28] B. K. Hillyer , A. Silberschatz, *Random I/O scheduling in online tertiary storage systems*, In Proceedings of the 1996 ACM SIGMOD international conference on Management of data, Jun., 1996, Montreal, Canada.
- [29] S. More and A. Choudhary, *Scheduling queries on tape-resident data*, In Proceeding of the European Conference on Parallel Computing, 2000.
- [30] S. Prabhakar , D. Agrawal and A. E. Abbadi, *Optimal Scheduling Algorithms for Tertiary Storage*, Distributed and Parallel Databases, v.14 n.3, Nov. 2003.
- [31] B. K. Hillyer, R. Rastogi, and A. Silberschatz, *Scheduling and data replication to improve tape juke-box performance*, In Proceedings of International Conference on Data Engineering, pages 532–541, 1999.
- [32] H. Newman, *Why Tape Won't Die*, Enterprise Storage Forum, June 16, 2005.