Journal of Physics: Conference Series **219** (2010) 052008 doi:10.1088/1742-6596/219/5/052008

A high performance hierarchical storage management system for the Canadian tier-1 centre at TRIUMF

D. C. Deatrich[1], **S. X. Liu**[1], **R. Tafirout**[1]

[1] TRIUMF, 4004 Wesbrook Mall, Vancouver, Canada V6T 2A3

E-mail: storage@lcg.triumf.ca

Abstract. We describe in this paper the design and implementation of Tapeguy, a high performance non-proprietary Hierarchical Storage Management (HSM) system which is interfaced to dCache for efficient tertiary storage operations. The system has been successfully implemented at the Canadian Tier-1 Centre at TRIUMF. The ATLAS experiment will collect a large amount of data (approximately 3.5 Petabytes each year). An efficient HSM system will play a crucial role in the success of the ATLAS Computing Model which is driven by intensive large-scale data analysis activities that will be performed on the Worldwide LHC Computing Grid infrastructure continuously. Tapeguy is Perl-based. It controls and manages data and tape libraries. Its architecture is scalable and includes Dataset Writing control, a Read-back Queuing mechanism and I/O tape drive load balancing as well as on-demand allocation of resources. A central MySQL database records metadata information for every file and transaction (for audit and performance evaluation), as well as an inventory of library elements. Tapeguy Dataset Writing was implemented to group files which are close in time and of similar type. Optional dataset path control dynamically allocates tape families and assign tapes to it. Tape flushing is based on various strategies: time, threshold or external callbacks mechanisms. Tapeguy Read-back Queuing reorders all read requests by using an *elevator algorithm*, avoiding unnecessary tape loading and unloading. Implementation of priorities will guarantee file delivery to all clients in a timely manner.

1. Introduction

The ATLAS experiment [1] at the Large Hadron Collider [2] (LHC) complex, located in Geneva (Switzerland), will collect an enormous amount of data at an unprecedented scale. The experiment will essentially operate continuously and will produce several petabytes of data each year. The great majority of this data will be stored on a tertiary storage system (tape library infrastructure).

For efficient access to the data, the ATLAS Computing model is based on a set of tiered computing centres which are connected via high performance and dedicated networks, the Worldwide LHC Computing Grid [3] (WLCG). The primary data will be initially stored at the Tier-0 centre at CERN, and then distributed to ten Tier-1 centres around the world. Each Tier-1 centre has custody of a share of the data. TRIUMF [4], Canada's laboratory for nuclear and particle physics, located in Vancouver, is hosting the Canadian Tier-1 centre.

The storage infrastructure is perhaps the most challenging aspect for the Tier-1 centres which operate continuously with an expected data availability and reliability better than 98%. In order to manage efficiently a large storage infrastructure, the dCache [5] software suite is used at TRIUMF. There are several key benefits in using dCache: it is a scalable solution, it

is highly configurable, and it has a common file namespace which allows for an aggregation of non-homogeneous server nodes with different file systems.

One of the primary roles of the Tier-1 centre is to reprocess all the primary data of a given period several times a year whenever improved calibration constants are made available. Since most of this data is stored on tape, an efficient and scalable Hierarchical Storage Management (HSM) system is very important. We expect the tape system to be accessed continuously. For efficient operations, the HSM system should be able to handle request prioritization and do reordering, file grouping and tape grouping, since different data types have different access patterns. The goal is to minimize tape mounts and maximize the read throughput for each mount.

In the following sections we describe Tapeguy, a high performance non-proprietary HSM system that was developed at TRIUMF. It is interfaced with dCache as a plug-in to its HSM layer [6]. After describing in some details its design, architecture and implementation, we will show some performance results and then finally conclude.

2. Architecture

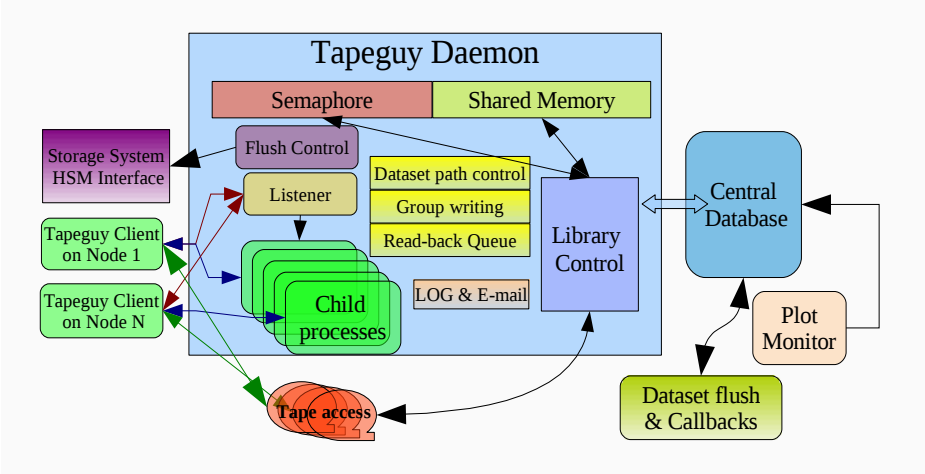

Figure 1. Tapeguy Architecture

The architecture of Tapeguy is represented in Figure 1. Tapeguy uses the TCP socket-based server-client model. The Tapeguy server daemon runs on the HSM server node and is capable of controlling multiple tape libraries. It references and records information about tape library configurations, tape volumes, tape operations and file archives in a relational database.

HSM-configured dCache pool nodes are attached to the tape libraries, and have access to the tape drives for efficient I/O operations. The HSM pool nodes either send files to tape or receive files from tape when they call the Tapeguy client program for each file request. Each Tapeguy client contacts the Tapeguy server daemon on the server node with a *get* or *put* request for a single file. The file request also includes metadata such as the directory path and the file size.

Tapeguy clients connect to the Tapeguy server for each HSM request (see section 3). The server daemon forks a server child process for each client. The child server prepares the tape drive with the required tape volume. Then the server child passes back to the client the full *tar* command that the client must run. The client runs the command on the server's behalf

and passes back the result to the server child. The server child updates the database, and both server child and client clean up and exit.

The child server process organizes client requests in order to optimize tape operations and maximize global data streaming. For tape writing Tapeguy uses the file path information to group files together in close proximity on tape (Dataset path control, Group writing). For tape reading Tapeguy queues requests so that files are retrieved from tape as efficiently as possible (Read-back queue).

To control access to tape libraries and tape drives Tapeguy uses standard semaphore and shared memory inter-process communication features. In particular the tape drive occupancy and tape slot information are recorded in shared memory, and is therefore accessible in a controlled fashion to Tapeguy server children. Semaphores control access to the library hardware so that competing child server processes do not concurrently send commands to a tape library. Similarly, semaphores control access to tape drives. Semaphores also protect some database operations.

Not all Tapeguy operations occur in the context of the server-client paradigm. Independent processes (e.g. cron, curl/web callbacks) trigger Dataset flushes to tape. For example, a cron-based Tapeguy program updates the database independently with information that the Tapeguy server uses to decide when to flush queued file datasets to tape.

Event logging and email notifications are also built into Tapeguy. Finally, the database back-end is available for web processes which independently plot data throughput and performance.

2.1. Implementation

Tapeguy is implemented in Perl [8]. It is comprised of a TCP socket-based server daemon, a simple TCP socket-based client and a supporting set of Perl modules. The Perl modules are divided into three logical files

- common functions for the daemon, the client and the other modules
- database-access functions
- device-access functions

No proprietary library or drive access code is used. Tapeguy makes use of standard primitive Linux utilities - mtx [7], mt and tar - to control tape libraries and devices operations.

On the Tapeguy server MySQL [9] is used as the database back-end. The choice for MySQL is because of its familiarity, simplicity and reliability. The Perl DBI database abstraction layer and the Perl DBD::mysql driver are used for database communication. Currently we are using MySQL version 4.1 as supported by Red Hat Enterprise 4, and the default MyISAM [10] storage engine is used for the database.

The Tapeguy client and the common module are installed on all HSM dCache Pool nodes. Additionally on the Tapeguy server node the daemon is installed along with all Tapeguy Perl modules.

A handful of Perl and shell scripts provide important additional functionality, including the dCache HSM interface shell script which is specified in the dCache configuration. This script is installed on all dCache HSM Pool nodes, and is called when dCache accesses the HSM layer.

At start-up the Tapeguy server daemon reads a configuration file which lists the name of the tape library it controls and the list of client host names which are allowed to connect. It then initializes the data structures, gets the library device parameters from the database, initializes the device semaphores, reads the status of the tape library using mtx, and writes the current library status information into shared memory and into the database. The daemon then loops listening for client connections.

A typical tar command to write a file to tape resembles the following:

```
/bin/tar -b <Blocksize> -C <DirectoryPathToData> -cf <deviceFile> <PNFSID>
```

A blocking factor of 8192 (or 4 MiB) was chosen because the files stored on tape have an average size of the order of 1 GB.

During operation the daemon writes various log files on the HSM server node, including a general log file with debug information and a log file of MySQL statements which modified the database, used for tracking, debugging or recovery in case of serious database issues.

Tapeguy runs as an unprivileged user on the Tapeguy server node. However the daemon must have library and tape drive device access. This is accomplished by using *udev* [11] to assign ownership of the changer and the tape drives to the Tapeguy user. udev is also used to provide persistent naming of Fiber Channel (FC) devices across reboots where World Wide Names (WWN) and World Wide Port Names (WWPN) are unique.

As well as udev, at system boot-up Tapeguy uses *stinit* from the *mt-st* [12] package to correctly initialize the tape drives.

Tapeguy clients runs on HSM client nodes with the privileges of the user who runs the HSM interface service. The clients and server have access to all tape drives in the same FC Storage Area Network (SAN) zone.

3. Communication with dCache HSM Interface

dCache HSM pool nodes are configured to call an interface shell script for *put* and *get* file operations. The shell script implements the dCache HSM interface; that is:

```
<cmd> put | get <pnfsid> <localFileName> -si=<storageInfo> [options]
```

The script does some elementary error checking, and then converts the HSM command-line interface into a Tapeguy client command with the appropriate arguments. The script also defines the location of informational log files on each HSM dCache pool node.

Allowed script return code ranges for dCache are defined in the dCache HSM Application Programmer Interface (API). The Tapeguy client makes use of the user-defined error code ranges from 30 through 39 for site-specific information messages. For instance, for PUT requests an error code of 30-39 deactivates the request.

In addition to the standard dCache HSM interface, Tapeguy also implements a dCache-specific administrative control interface. The interface allows Tapeguy to initiate file flush control and to rotate dCache pool modes between read-only and read-write modes.

Unfortunately there is no dCache Administrative API. Thus the Tapeguy daemon user must be granted ssh access to the dCache Admin Interface [13].

4. Tape Library Management

The Tapeguy daemon must know the status of the tape library at all times. Using vanilla *mtx* to get this information is a resource-intensive operation, since the vanilla version of mtx queries all elements in the library. This is useful at start-up, when the daemon wants to know the real state of the library so that it can initialize data structures, shared memory and database information. Later, it is more efficient to let child processes get library information from shared memory. Access to the library for status information is only needed when reading, writing, mounting and unmounting tapes. Most of the time the daemon only wants to know about the state of the tape drives, for example before and after writing a file to tape. A modified copy of mtx named *mtx-driveinfo* is installed. It only queries the state of the tape drives without the overhead of querying tape slot information.

Tapeguy is designed for multiple tape libraries. Each library device has a global semaphore that controls its access. In addition, at the tape drive level, Tapeguy uses a database table named *operations* for logical control of the drive. Each drive has a lock field indicating if the drive is available, and an assigned operational type - *WRT*, *RD* or *PUB*. A drive that is of

type *WRT* only writes files, a drive of type *RD* only reads files, and a *PUB* drive may perform either operation depending on the demand. To balance tape drive usage, Tapeguy uses a Least Recently Used (LRU) algorithm when picking the next drive to use. By using the operational type and the LRU algorithm Tapeguy is able to balance drive usage, as well as dynamically shift tape operations from one mode to the other.

Tape and library operations are recorded in the database for diagnostics, tuning and statistics gathering. The logged data includes the volume label, device name, file size and the dCache pnfsid filename along with the elapsed time and the event, which may be one of: read mount, read unmount, write mount, write unmount, write, read, position, rewind, library status query, drive status query, or block position query.

5. Dataset Group Writing

The TRIUMF Tier-1 share of the LHC ATLAS data that will be stored and reprocessed indicates a total of about 400 MB/s of tape throughput capability would be required (reading and writing). A handful of LTO-4 [14] drives each streaming at their native speed of about 120 MB/s could in principle provide ample I/O capability. In reality, random requests combined with slow seek times would effectively limit severely the throughput. Therefore, it is important to organize and group the data at the writing stage for an optimal reading access later.

The ATLAS experiment produces data types destined for a tertiary storage with similar processing times and access patterns, so it is possible to group and pack such data together when writing them to tape. Packing data on tape dramatically reduces tape mounts and tape head seeks when reading. Tapeguy groups data based on dataset names and types as extracted from the path names. Tapeguy uses the physical directory tree to group the data into tape families. ATLAS data is organized in the following manner:

```
<site ATLASDATATAPE>/project/datasetType/datasetName/filename
```

Figure 2 illustrates the relationship between path name structure, data types and tape groups. A file which is under a level 1 directory, with project name *data*, and datasetType RAW is a valid structure to Tapeguy and will be automatically registered under its dataset. A new data

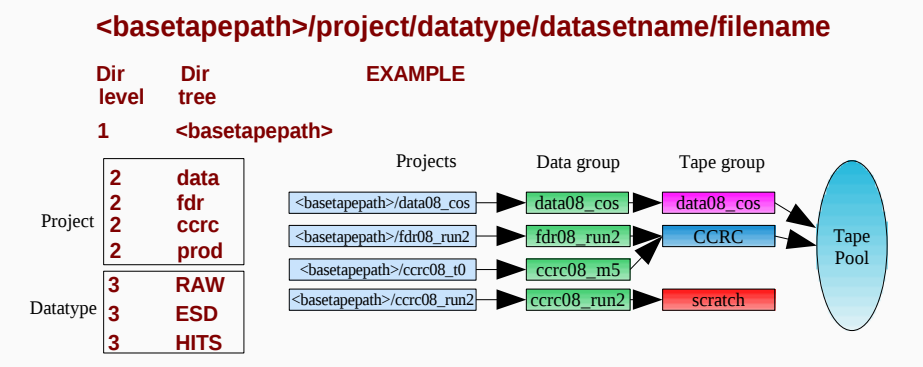

Figure 2. Example ATLAS Path Structure

group will be created automatically when a new file with a valid directory structure comes to Tapeguy. The data group will be named using the project name. A notification e-mail will be sent out for new directory structures that are unknown to Tapeguy. This is to avoid random assignment of tapes.

A *tape group* will also be paired to a data group. A tape group can be shared in some small data groups manually to maximize tape cartridge usage. Tapeguy automatically allocates tapes from the default tape pool on demand. The administrator automatically receives e-mail notification for each new data group created and for each tape assignment.

Further, dataset files are packed together on tape. A file is registered in a dataset when it arrives at Tapeguy from the HSM interface. It stays on disk until Tapeguy asks the HSM interface to flush it to tape. Flushing starts when one of the dataset writing thresholds is reached. Processing time of a dataset is one threshold used; another is data volume. An external callback mechanism from the ATLAS data management system can also be used.

The first flushed file in a dataset pre-reserves drive(s) for the whole dataset. The number of pre-reserved drives depends on the number of tapes the dataset needs at that moment and the number of available $WRT + PUB$ drives. Thus data will be packed closely on tape. First-In First-Out (FIFO) is the policy used when Tapeguy flush files in a dataset.

A special tape-group named *scratch* is used for tape functional tests. It is effectively a circular buffer, where Tapeguy recycle tapes in turn in this tape group. The lifetime of functional test files on disk determines how many tapes are manually assigned to the *scratch* tape group.

Metadata information for each file is recorded in the central database. A logical verification is also in place to verify the number of blocks a file consumes on tape.

6. Read-back Queuing

Because of effective dataset group-writing to tape, subsequent read-back from tape is more efficient due to the proximity of files on a small set of tapes. Nevertheless, tape loading, unloading and file seeking are expensive operations, and some effort is required to optimize throughput of tape reads when multiple requests are queued.

Two dynamic queues are used in read-back queuing. The *general queue* is used to avoid unnecessary tape loading and unloading, and the *tape queue* is used to avoid unnecessary tape head seeking.

The *general queue* is used to decide which tape should be loaded first. Tapeguy loads the tape which has the most requests at the moment amongst the unmounted tapes. Once a tape is loaded in the drive, it remains open for new requests and only is unloaded when all requests for it are served.

The *tape queue* is used for reordering read requests in a tape. Tapeguy uses C-SCAN algorithm to reorder file requests according to the file position on that tape. Since there is a request cap in the HSM interface itself, the algorithm is effectively N-step C-SCAN. This algorithm optimizes tape head positioning. Requested files are served in order *from the current tape head position* until the last requested file on that tape. Then the tape head will reposition to the first requested file by position earlier on the tape, and repeat this process until all requests are served. This is also good for LTO-4 drive R/W behaviour since they write back and forth. If a tape is written to full capacity, the last pass is a reverse pass leaving the head at the beginning of the tape.

A read-back request coming to Tapeguy is assigned a minimum waiting time if the tape it needs is not in a drive. It will be registered in the queue but will not be served until the minimum waiting time is reached. The queuing process dynamically calculates each request's approximate waiting time at that moment, then closes the session. The request will remain alive at the client, which sleeps during the waiting time. This procedure is repeated until it is eventually served. In the current dCache HSM interface design, there is no parameter to handle priorities, but there is a time limit for each read-back request.

Currently each request has the same priority. The priority will be adjusted with the time waiting. In the *general queue*, a tape which has one or more zero priority requests (zero is the highest priority) has the same opportunity as a tape which has the most requests at that

moment. The tape with zero priority is more likely to be served, assuring client access to the file in a timely manner.

7. I/O Balancing

Four LTO-4 drives can easily saturate the FC bandwidth of 4 Gbps (or the total I/O that a host is capable of). However, the tape speed adjusts to the available data stream within the minimum and maximum streaming speed. The tape streaming speed will be very low when the host disk I/O is poor, and will not change unless the current tape write or read is finished. It is even worse if the data transfer rate falls below the minimum streaming threshold. The drive will eventually move back and forth the tape then resume operations. This significantly affects the throughput as well as the drive and tape lifetime.

When Tapeguy starts writing data to tape, the disk write pools are typically also busy receiving data. Tapeguy uses a *write pool rotation* policy to avoid disk and tape I/O competition. With write pool rotation enabled, Tapeguy separates write pools into two shifts - it sets one pool to read-only mode while the other pool remains writable in order to accept new files. Tapeguy only flush files in read-only mode pools, therefore reducing competition between disk I/O and tape I/O. Tapeguy reverses the pool modes between the two shifts when the current writable pool's dataset must be flushed to tape.

FIFO is the policy used when files in a dataset are flushed to tape. Tapeguy also considers I/O load balancing across write pools. The file in a pool with less load will be chosen for the next file to be flushed, as illustrated Figure 3. In this example where two drives are used for

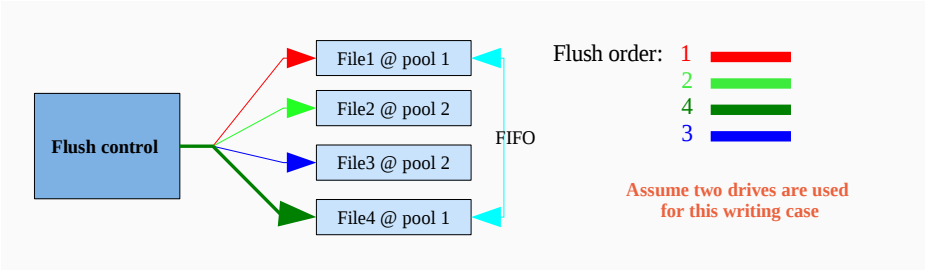

Figure 3. Flush Control

this writing, the file flush order ideally should be: 1,2,4,3

So far, there is no mechanism for the dCache HSM to allow Tapeguy to reassign a request from one disk pool to another. For I/O balancing at the reading stage Tapeguy tries to avoid too much tape-reading on one host at a time. The parameter *max_reading_on_one_pool* is set. The reading requests will be queued and wait until the current tape reading on that host is less than that threshold. It results in stable and more efficient read streaming speed on hosts.

8. Performance

The current tertiary storage infrastructure at TRIUMF consists of the modular TS3500 Tape Library (formerly known as IBM 3584) and it is based on LTO-4 technology. The current hardware configuration consists of two library frames fully populated with a total of 700 tape cartridges. Each tape cartridge has a capacity of 800 gigabytes (native), thus providing a total capacity of 560 TB. At the moment there are eight LTO-4 tape drives.

Various bulk read-back (also known as *data prestaging*) tests have been done so far. The tests match closely the expected use cases and expected access patterns. The current system

can prestage a data volume up to 1 TB per hour in a sustained way when using up to 6 LTO-4 drives. The throughput as a function of time is shown in figure 4 and the corresponding mass storage system metrics [15] as show in Table 1. The system consisted of four HSM dCache pools (dual Intel Xeon 5160, 8-10 GB of RAM and 10 GigE), where one of them was also running the Tapeguy daemon. The performance is very good and will scale with the addition of hosts and tape devices. We believe that the current results are perhaps limited by the hosts and not by Tapeguy design.

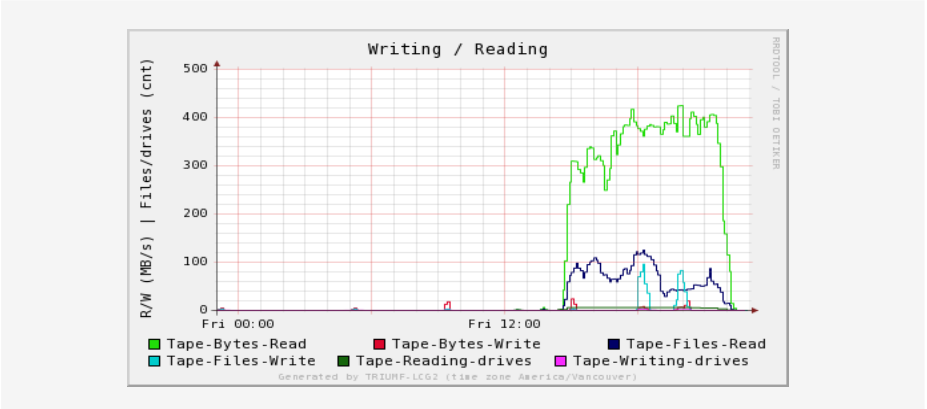

Figure 4. Tapeguy throughput during prestaging

Parameter	Value
Read Rate (MB/s)	65.5
Write Rate (MB/s)	52.1
Average File Read Size (MB)	3001
Average File Write Size (MB)	4160
Read/Tape Mount (MB)	849740
Write/Tape Mount (MB)	37440
No. of Repeat Read Mounts	0
No. of Repeat Write Mounts	0

Table 1. Mass storage metrics

The XFS file system is used due to its high performance and proven architecture when dealing with very large directories. There is no need for large parallel file systems since it is very important to have some kind of a modularity in such a way that a failure of a particular component or server does not affect the performance and availability of the system as a whole.

9. Conclusion

At the TRIUMF Tier-1 centre, we have chosen not to use a proprietary tape library management systems as provided by the vendor. We believe that Tapeguy provides us with greater control and flexibility. This allows us to tune the system in order to meet the various ATLAS use cases and access patterns. We have described in this paper the design and implementation of Tapeguy and shown that the system performs well. The system is expected to be scalable in order to match an increasing throughput demand in the coming years. This is important especially since

the Tier-1 centre goes through an expansion stage every year, when more clients nodes and more storage capacity are added to the system. Tapeguy is interfaced with dCache at TRIUMF, but we believe it can be interfaced with any storage system requiring tertiary storage.

Acknowledgments
Acknowledgements The Tier-1 project at TRIUMF would not be possible without the contributions and support from: Canada Foundation for Innovation (CFI), British Columbia Knowledge Development Funds (BCKDF), National Research Council (NRC), National Science & Engineering Council (NSERC), CANARIE, BCNET and HEPNET Canada.

10. References

[1] For more information on ATLAS experiment, visit http://www.atlas.ch/
[2] http://public.web.cern.ch/public/en/LHC/LHC-en.html
[3] http://www.cern.ch/lcg
[4] http://www.triumf.ca/
[5] For more information about dCache, visit http://www.dcache.org/
[6] http://www.dcache.org/manuals/experts_docs/dCache-Hsm-Interface.html
[7] http://mtx.opensource-sw.net/
[8] http://www.perl.org/
[9] http://www.mysql.com/
[10] http://dev.mysql.com/doc/refman/5.1/en/myisam-storage-engine.html
[11] http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html
[12] http://mtx.opensource-sw.net/
[13] http://www.dcache.org/manuals/Book/start/intouch-admin.shtml
[14] http://www.lto-technology.com/technology/udata.php?section=0&subsec=udata
[15] https://twiki.cern.ch/twiki/bin/view/FIOgroup/TapeRefEfficiencyMetrics