The Design and Evolution of Jefferson Lab's Jasmine Mass Storage System*

Bryan K. Hess
Thomas Jefferson National Accelerator Facility
bhess@jlab.org

Michael Haddox-Schatz
Thomas Jefferson National Accelerator Facility
mschatz@jlab.org

M. Andrew Kowalski
Thomas Jefferson National Accelerator Facility
kowalski@jlab.org

Abstract

We describe the Jasmine mass storage system, in operation since 2001. Jasmine has scaled to meet the challenges of grid applications, petabyte class storage, and hundreds of MB/sec throughput using commodity hardware, Java technologies, and a small but focused development team. The evolution of the integrated disk cache system, which provides a managed online subset of the tape contents, is examined in detail. We describe how the storage system has grown to meet the special needs of the batch farm, grid clients, and new performance demands.

1. Introduction

Jasmine is the integrated tape and disk mass storage system developed and used by the Thomas Jefferson National Accelerator Facility (Jefferson Lab). The lab is operated for the U.S. Department of Energy to conduct nuclear physics experiments examining the quark nature of matter. The data rates generated by these experiments are currently ~30MB/sec, and will eventually reach 100MB/sec. Jasmine manages more than 1 PB of data stored on tape. Aside from maintenance and upgrade periods, the system has run in a "lights out" mode with 24 hour operation and daytime administrative work since its deployment.

In this paper, we outline the design and operational challenges experienced in transitioning Jasmine from a mission critical storage system to a multi-purpose storage fabric for use by running experiments, batch farm processing, grid applications, individual users, and the lab's lattice QCD effort. The basic distributed framework of the system has scaled well. We describe modifications made to system components, like the disk cache, to grow it to its current state. Finally, we anticipate future changes based on current stresses on the system.

* The Southeastern Universities Research Association (SURA) operates the Thomas Jefferson National Accelerator Facility under DOE contract DE-AC05-84ER40150.

2. Jasmine Design

The initial design and implementation of Jasmine was presented in 2001[1]. We described at that time an almost pure Java, database-driven, distributed system. There were two primary customers for Jasmine's services. First, data acquisition from the experimental halls required immediate access to tape drives to write data without any queuing. Secondly, batch farm and user requests, which queued until the system could process the requests to pull files from tape. User access to the system was through user level commands to read files, write files, or request that files be added to the user visible disk pool.

The overall architecture is shown in Figure 1. Data mover machines manage file movement to tape. Disk Nodes store cached copies of files or files in transit to tape. A replicated MySQL[9] database forms the central core of the system. Data flow is between the client and the data movers or the disk nodes; there is no single data flow bottleneck. Design Goals

Jasmine was designed with the goals of minimizing single points of failure, creating a system that would scale for future needs, and be able to provide both queuing and non-queuing service for clients. Experimental data rates tend to increase over time as detector and data acquisition technology improve. We wanted Jasmine to scale with these changes, realizing that this could mean an increase in the data rate by an order of magnitude over five years. We anticipated that some of this data rate increase could be solved by new tape, disk, and networking technology, but that some of it would require a distributed architecture where

Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2005)
0-7695-2318-8/05 $20.00 © 2005 IEEE

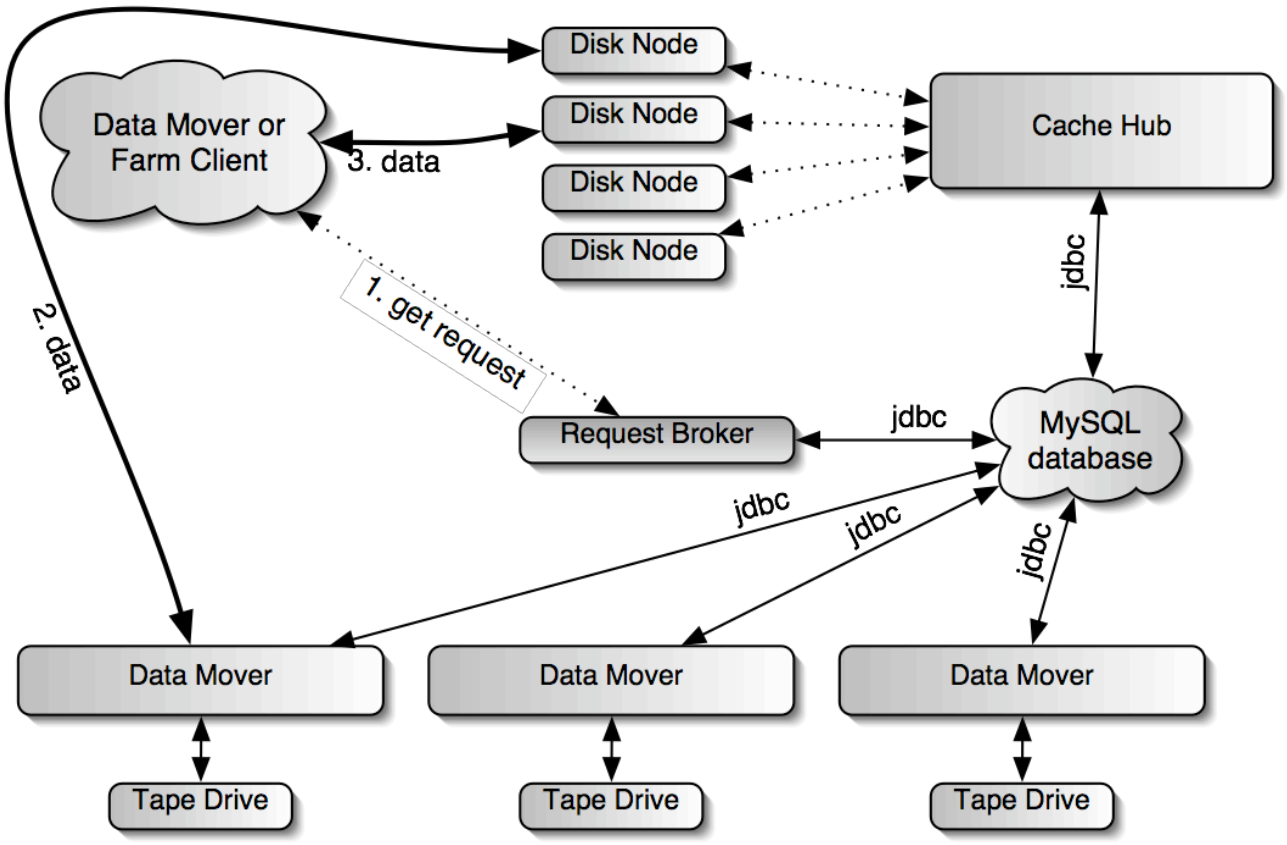

Figure 1. Overall Architecture

multiple data paths are used to achieve high aggregate performance.

There are two classes of failures addressed in the design. For those failures that can be managed in an automated way, it is important that they result in degraded performance rather than system-wide failure. That is, individual redundant system component failures should be detected and contained. Machines that manage tape or disk resources are examples that can fail in a contained way. Certain files or user requests may fail as a result of a single machine's problems, but those failures do not cascade into other parts of the system.

The second sort of error management is with regard to system critical components. Some single points of failure are too costly or complex to eliminate completely but the risk can be mitigated by selecting high quality solutions and documented recovery plans. In the design of Jasmine there are several of these components: The database, the network, and the tape robotics are the most obvious examples. In the case of the MySQL[9] database, we have added replication and frequent backups. In the case of the network we rely on shelf spares. For the tape robotics, we rely on 24 hour vendor support.

2.1. Name Space Management

Files are stored in Jasmine as digital objects (files) that reside on disk, tape, or both. These files are organized into hierarchies of directories in the Jasmine database. The directory structure and file entries are presented in an NFS-mountable file system that is browsable as any other file system. The entries in this file system are *stub files*, which are mere placeholders for the actual data. These stub files are used as the names for files on tape, but they are not the files. By convention we mount this stub file system on our scientific computing machines under /mss. Jasmine updates /mss as files are added and removed, but the database is the master record of the name space hierarchy.

When a user wants to place a file into tape storage, they run a command that contacts Jasmine and moves

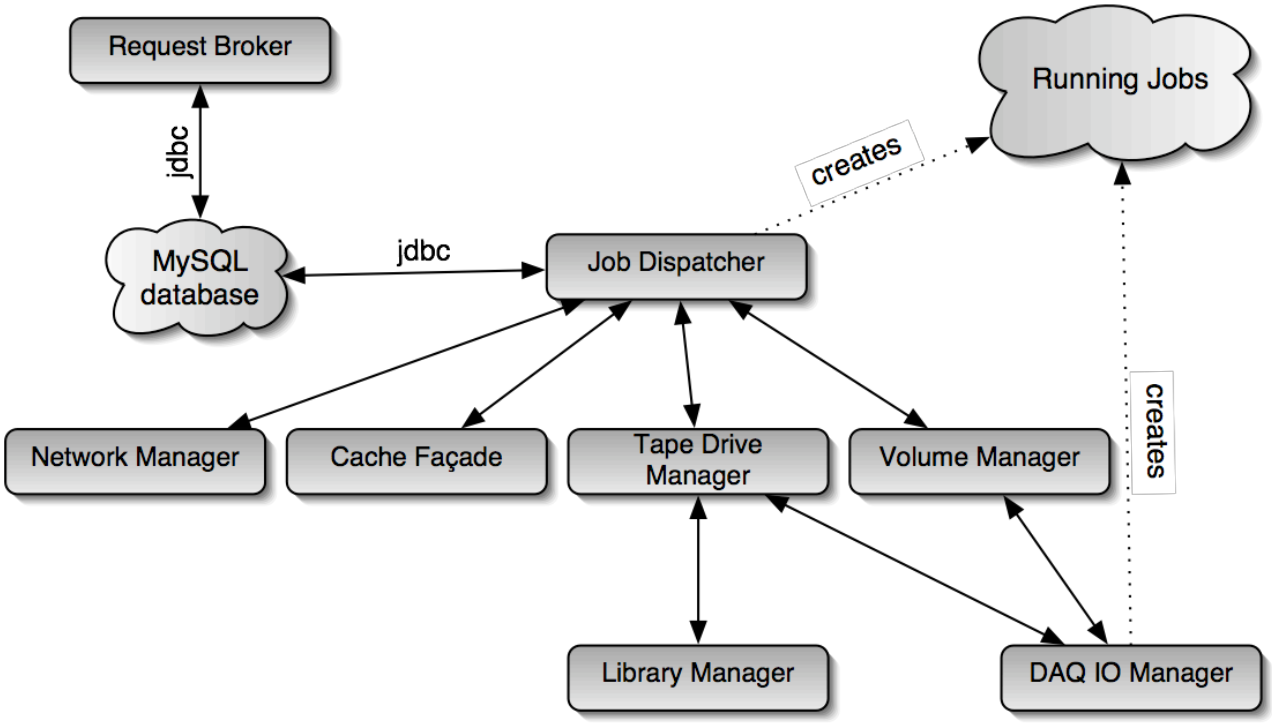

Figure 2. Data Mover Architecture

the file to tape. A name space entry under /mss is used as the name of the file on tape, and a stub file is written to show that the name is in use. Users retrieve files using these same stub file names.

When writing to tape the name space is used to choose the set of tapes that will be used for storing the file. Under /mss, directory trees are mapped to collections of related tapes, called volume sets. Files can only be written to tape if such a mapping exists. These mappings are inherited by subdirectories. For example, the path /mss/clas/eg3a/raw is mapped to a set of volumes reserved for raw data from that experiment. The parent, /mss/clas/eg3a could be mapped to a more general set of volumes for analyzed data so that any new subdirectories (other than raw) would inherit that mapping.

2.2. Data Mover

A data mover is a computer with an attached tape transport and some amount of local disk space for local staging of files to/from tape. Figure 2 shows the high-level architecture of the data mover. The dispatcher coordinates with the system resource manager for tape drives, disk space, and network bandwidth to decide which jobs to run. Each data mover culls through the job queue database looking for work to do based on available resources. When a group of runnable jobs is found, a cluster of running jobs is created. Jobs are gathered into clusters based on their resource needs. For example, a group of jobs in the queue that need files from the same tape are processed in the order of files stored on the tape.

The data mover always stages data in two phases, one is a tape transfer and one is a network transfer. In the case of writing to tape, the data is copied over the network to a pool of disk space dedicated to that data mover in the first phase. In the second phase that data is copied from the staging disk to tape. When data is read from tape, the order is the reverse: Phase 1 is a copy from tape to disk and phase 2 is a copy from the staging disk over the network to the client.

This two-phased scheme for data movement increases the latency of job processing, but has several advantages. First, it hides any data rate mismatch between the network and the tape drive when the disk is not the performance bottleneck. Secondly, it allows data being written from clients to be staged in from clients when no tape drive is immediately available to write the data. This allows for faster processing from the point of the view of the client. Finally, it allows the tape drive to

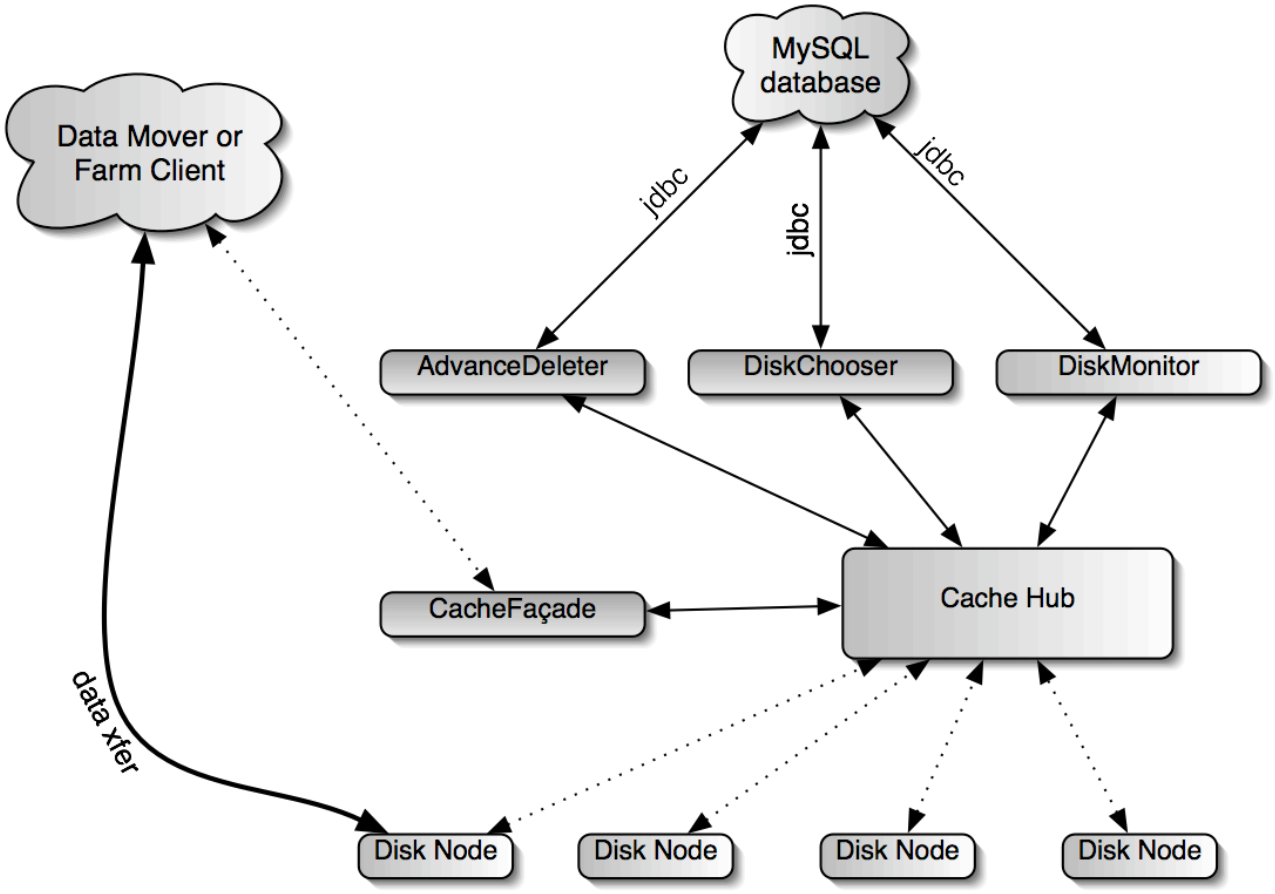

Figure 3. Cache System Architecture

read files from the tape at native speed, even when writing to slow clients. Using the disk as a buffer works best when a data mover has one tape drive, one network interface, and an appropriate amount of disk space. The tape drive and disk performance are knowable quantities. The network performance is the least predictable aspect of the system.

There is a second means for access to tape storage which neither queues nor uses the intermediary disk cache. This lower-level mechanism is designed for collecting data from experiments. The Data Acquisition Manager bypasses queue processing and allows running experiments to write running data to tape on demand. Tape drives may be reserved exclusively for data acquisition so that experiments never wait for tape access, or they can preempt the queue between clusters of jobs.

When a user requests a file from tape, the request is made based on the stub file name. These requests are placed in the queue database for processing. Data mover nodes process these requests when they have sufficient tape drive and disk resources. Our strategy for queue processing focuses on minimizing tape mounts and positioning time. When a request for a tape rises to the top of the queue, any other requests in the queue needing the same tape, regardless of queue position, is processed as part of the same job cluster. Jobs are also ordered to read the tape sequentially. Any new request in the queue for a mounted tape will be processed first to avoid an additional tape mount. This queue discipline hides some of the latency of tape processing at the expense of some fairness.

2.3. Disk Cache

A primary feature of the initial design of Jasmine was the integrated disk cache, a system that we developed and used prior to Jasmine and carried over into Jasmine's design. The cache keeps disk-resident copies of files stored on tape and deletes them in a least-recently-used order when space is needed for new files read from tape. This disk cache was exposed to users

and the batch farm using NFS, and provided an alternative file retrieval mechanism. Users could read files at will from the disk cache, but all the file additions and deletions were done by the system according to system policy.

The disk cache was designed to be used in several different roles: as temporary staging space on data movers, as long-term storage for frequently accessed data, and as a repository of input files for the batch farm. The graceful handling of failures was an important part of the system. The disk cache system has changed significantly since the first implementation of Jasmine, as discussed in section 3.1.

2.4. Library Manager

The library manager is the third major component of the system. A library manager coordinates tape mount and dismounts and answers data mover queries about the availability of volumes. The library manager provides a simple library abstraction on top of a vendor supplied API. Our library manager implementation is built on top of StorageTek's ACSLS.

3. Jasmine Evolution

Jasmine has undergone several significant software updates since the initial release. The current version, Jasmine 4, is described here. Changes were geared toward the redesign of the cache disk system and the specialization of client interfaces including batch farm processing, grid storage with Storage Resource Managers (SRM), and programmatic clients. The status web pages were rewritten using Java Server Pages (JSPs) and servlets to maximize reuse of the core classes and provide a more flexible means for maintaining web content. The mechanism for tape copying was replaced to better track the provenance of a file. Strategies for processing tape requests were also modified to make better use of tape mounts.

3.1. Disk Cache Redesign

Jasmine's initial disk cache system consisted of gigabit Ethernet attached hosts with hundreds of GB of disk space (cache nodes), which acted as file servers and cached copies of files read from tape. Space was allocated in logical partitions (*disks*) to experiment-specific cache groups. Files were added to the appropriate disk group's disks based on the user's experimental group membership. Files were deleted on a least-recently-used basis when new files were added. Failures were problematic because all of an experiment's files could be rendered unavailable with the failure of just one or two nodes where that group's files were concentrated.

The disk cache has been overhauled based on these experiences. A significant change to the architecture of the disk cache is that all the user-visible cache disks are treated as a single pool. Experimental groups are assigned allocations in this pool. When one experiment adds a large number of files to the cache for batch processing the files can be retrieved from tape and quickly spread over all the available disks. This improves the overall bandwidth of the system, and prevents individual cache nodes from being overwhelmed by hundreds of farm nodes. A single node failure effectively deletes some files from the cache, but does not stall any individual experiment's work because the files can be re-cached to other nodes.

One drawback to this approach is that failures are now more complicated. When a disk node fails, some percentage of all files becomes invisible, which can be perplexing to the user.

Figure 3 shows the major components of the new cache system. The cache façade is the API for other components, like the data mover and the batch farm, that use the cache for file storage. Cache requests are made through the cache façade, but the data flow is always directly between the client and the disk node machine that holds the data.

A central cache hub now monitors all disk nodes to make sure that they are alive and reporting back file access time changes that are needed for file deletion decisions. A disk node that does not provide periodic heartbeat messages is marked offline. The files stored on an offline disk are considered to no longer exist. This strategy allows the system to return to tape and re-cache any files that are requested but have dropped out of the cache. Should the failed disk node be repaired, the files are once again made available for user access.

The cache is organized into *pools* of disks. Each pool can contain many *families* of files, each of which has an allocation. These allocations may be strict or relaxed. Families with strict allocations may not exceed their limit in any case. Families with soft allocation limits may overrun their space when other families in that pool are not using their allocation.

Families use either automatic deletion or strict deletion. Automatic deletion is the usual case. Files are removed as they age out. With strict deletion, files are removed upon user request only. This is useful for keeping online data sets that need frequent reanalysis. The use cases are outlined in the table below.

	Strict Deletion	Automatic Deletion
Strict Allocation	**Frequently Accessed Summary Data**	**Data Mover Staging Space; User-Writeable Cache**
Relaxed Allocation	**Not used**	**Batch Farm Cache; User Requests**

Figure 4. Cache Family Types

In the original cache system, files were removed only when new files were added and space was needed. This policy made good use of disk space, but could become a performance block when a large number of small files needed to be removed. For this reason, file deletion is now handled in advance. This policy is essentially a garbage collection policy. The system strives to always keep some space free.

The Advance Deleter thread on the cache hub removes files from experimental groups that exceed their allocation. The deletion strategy is a relaxed-LRU policy that attempts to remove files from most nodes in a way that encourages new files to be spread over all available disks. It works in the following way: For each disk in the cache pool, the families represented on the disk are examined. Any family that is not over its allocation on all disks is skipped. For any remaining over-allocated families, the files on the disk in question are examined. If any of them are in the oldest 5% of files for that family over the whole system, then they are deleted up to a pre-defined per-disk free space goal. This goal is typically large enough to hold a file of the largest size expected, currently 2GB.

By working to maintain most disks with enough free space to add at least one large file, we ensure that there is very little delay between the start of the job and the time when the file streams from the data mover to the cache disk. This also gives us flexibility to add files to disk in a loose round-robin fashion making sure that no individual disk partition is overloaded.

Files stored in any family or pool may be pinned. A pinned file is not subject to deletion until that pin is removed. File pins are used to make sure that files not yet committed to tape are not accidentally removed. Pinning also permits a data mover to retain files to satisfy an upcoming request without going to tape. The file pinning system in this version of the cache manager is much more sophisticated than the simple use counts that were kept on files in the first implementation. Since pins are stored in the database, they are easily tracked and audited.

A potential downfall in the cache system redesign is that decision-making has become more centralized, and the cache hub, which contains all the logic for file allocation and deletion, becomes a single point of failure. It is our experience that this is less problematic than trying to make decisions of similar complexity in a more distributed fashion. It also adds more definitive detection of failures and anomalies. The cache controller, along with the library manager and the database, become the primary essential components in the system.

3.2. Batch Farm Interaction

The batch farm is Jasmine's largest client. To make efficient use of the batch farm, it is essential that a farm job not be started until the input data (typically ~2GB) is in the disk cache. It is also important that any files added to the disk cache not be removed until the farm job requesting them had copied the files to the local farm node for processing. This need for improvement led to the related Auger[6] batch farm management project. To support this, Jasmine includes a new farm cache interface which caches and pins files in place for farm use until they are explicitly released by Auger. This allows our compute farm to spend the majority of its time in compute-intensive tasks, despite the data intensive nature of the work.

With the redesigned cache system, the batch farm no longer uses NFS for file transfers. When a user's batch job is placed on a farm node, the client makes a call to the cache system to directly retrieve the file from the cache node that is holding it. This has several advantages. First, if that copy fails the system falls back to retrieving the file from tape again. Secondly, since NFS is not used, a number of the NFS-related Linux bugs that have caused productivity problems are avoided. Replacing NFS also allows us to spread the farm cache load over a large number of machines that may not be NFS servers. Should any of these machines crash, they do not cause NFS IO hangs, reducing the need for immediate system administrator interaction.

3.3. Metadata Tracking

There are two classes of metadata to consider. The first is the storage-related information: file information that would normally be found in a UNIX file system and extended information like checksum and duplicate tracking. The second class of information is physics-specific: detector calibration, experimental notes, processing parameters, and the like.

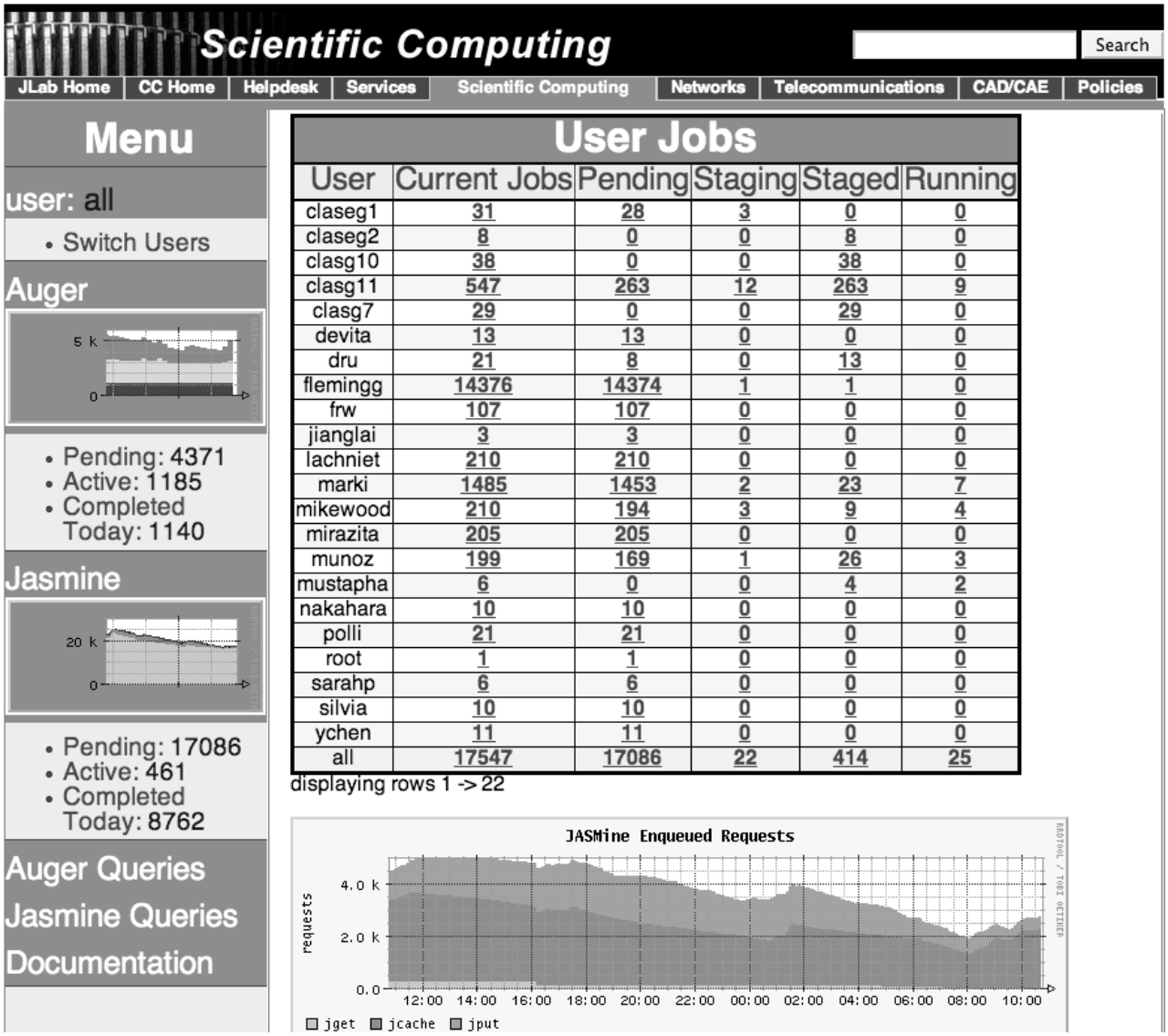

Figure 5. Jasmine Status Monitoring Web Page

Our decision was to track metadata that establishes data provenance, ownership, and usage information, but not to track data that is specific to physics data processing. We want to know, for each file, when it was written, what experiment it is charged against, and what type of data it is in a very broad sense: raw experimental data, duplicated experimental data, production data derived from raw data, or other unclassifiable data. Beyond this we break data down according to project to distinguish simulation output from, for example, data summary files that are used for final physics analysis. This level of detail is maintained to meet our requirement for system monitoring and accounting.

There is no barrier to establishing a comprehensive physics metadata catalog that would reference our metadata database directly, but because of organizational structure this responsibility falls to other groups within the laboratory.

3.4. User Interface

There are three sorts of user interface: Command line utilities, web pages, and programmatic access via API.

The user interface for the batch farm and for interactive access is through command line utilities to get and put files, to add files to the disk cache, and related administrative functions. User-level command

line tools are the choice in many cases because they can be used either for interactive work at the desktop, or in batch processing.

A combined web-page status and monitoring system for Auger and Jasmine is available[10]. This portal, shown below, is implemented using a typical J2EE[11] system involving servlets and Java Server Pages. A simple front page breaks down work by user and shows summary statistics for job processing during each day. This top-level view, along with time-series graphs of the batch and tape system load, give users and administrators an immediate view of the overall state of the system.

More advanced queries linked off the main page allow for reporting and graphing based on user-selected criteria. A goal for these web pages was to provide to the user immediate access to the most useful summary data (how many jobs do I have? What are their states?) while still giving enough control for system administrators and advanced users to make more detailed queries (Which of my batch jobs are waiting longest for file retrieval from tape? How many of my files have been staged to disk so far?)

For programmatic access to Jasmine, several special purpose interfaces have been implemented. These focus on providing file writing and retrieval services to other lab systems, such as the lattice QCD file manager, the Storage Resource Manager (SRM), and Auger batch system manager.

3.5. Tape Technology Migration

We have seen two major transitions in tape technology since the system began running. In both cases we decided to migrate the data from the old tape technology to the new. In the first case, this was a migration from the unreliable helical-scan Redwood tape drive to the reliable, but expensive 9840 tape. The second data migration was from the 9840 tapes to 9940B to achieve more storage per tape cartridge.

We wrote into the Jasmine queuing system a low-priority mechanism for migrating data from one set of tapes to another during system idle times. We anticipate this being a regular process every few years. In each case we copied over 4,000 tapes over a period of 9 to 12 months.

Although this data migration system has worked very well, it has brought forth some issues of data provenance. Data that is copied from an older tape technology to a new one still has value on the old tapes. They are a backup duplicate of the data should the new tape fail. Eventually the old data becomes unreadable as the drives are no longer available or supported, but up to that point they still have value. We track the existence of these duplicates. The tape-rewriting feature of Jasmine has also proven useful for rewriting and replacing tapes with high error rates or physical defects.

4. Operational Issues

The knowledge gained in the operation of Jasmine over the past few years has been useful for guiding future development. The system goes through development phases and production phases. During development phases we reexamine the architecture of components that need updating and make changes as necessary. During production periods, however, the emphasis is on operational integrity, and we shift to a system administrator's point of view, where the emphasis is on treating the software as a black box and working with configurations of the existing system to maintain performance and reliability.

4.1. Hardware selection and reliability

The greatest single impediment to system performance has been hardware reliability. As an example, the failure of cache disk nodes has been common. These failures have been because of unreliable power supplies, RAID controller failures, and similar causes. The commitment to low-cost PC hardware has allowed us to scale the system size, but the use of system administrator and technician time has been significant. Some failures have been complicated and hard to diagnose, leaving individual machines unusable for weeks or months at a time.

We have stepped back from the use of low-end commodity hardware in some areas. Our experience with white-box PC vendors has demonstrated that they frequently lack the experience with storage systems to provide reliable multi-TB RAID systems. Additionally, our experiences with PCI RAID cards under Linux has left us to consider that external hardware RAID or software RAID-based solutions may be a better fit for our needs.

4.2. Error Detection and Recovery

Our experience with unreliable hardware has encouraged us to put significant development into error detection and management. Our goal in error handling is to create a system as we described in our design goals—that is, one where faults result in degraded operation, not complete failure. In such a system, failures over nights

and weekends can be repaired at convenient times with a minimal amount of after hours support from on-call administrators.

When a tape fails to mount in a tape drive, both the tape and the tape drive are removed from the system and flagged as suspect. This allows a system administrator to examine both of them and make a decision. This prevents a bad drive from mangling multiple tapes, and it prevents a defective tape from being passed from drive to drive. We track error counts per-file, per-volume, and per-drive. This allows us to spot impending drive or tape failures and gives us a chance to copy data before it is lost. We log a complete history of what drives a tape cartridge has been in. This has helped us to track down tape drives that were damaging tapes in the past.

In the cache disk system, we remove any cache node from the system if it fails to open a file, usually an indicator of an underlying file system problem. We disable any system that does not report a heartbeat message to the cache hub within a settable interval. These measures tend to remove failing hardware while allowing the system to continue at a slightly decreased performance level.

Certain critical conditions, like a robotics failure or the disabling of all the data acquisition tape drives, automatically page on-call personnel for immediate repair.

4.3. System Administration

The developer's point of view is different from that of the system administrator's. Maintaining a complex in-house developed system like Jasmine is significantly different from building it. System administration tends to consider closed systems and how they can be integrated, managed, and repaired without access to internals. Even though we have access to the internals of the system, it is useful when in production to freeze the configuration of the system and take the system administrator's point of view.

We find that taking the system administrator's view during periods where the system is receiving data from running experiments to be useful. Code changes and updates are made as bug fixes only on scheduled maintenance days. Major code changes are held off until facility shutdown periods where changes can be introduced and tested. Freezing the code state and living with known bugs is useful because even innocuous code changes, if not well tested, can lead to unintended consequences in the running system.

5. Ongoing Development

There are two design aims that are driving current development efforts within Jasmine. The most obvious emerging need is for infrastructure to support data grid computing. This includes taking a fresh look at accounting, authorization, and authentication and how these needs can be met in a grid environment.

At the same time, development changes that can enhance the on-site performance of the system also continue.

5.1. Grid Fabric

Grids are emerging as a solution to the data and compute intensive requirements of high energy and nuclear physics experiments. Storage Resource Managers (SRMs)[7] provide grid-level access to storage at diverse sites through a common interface mechanism. Jefferson Lab has been active in the SRM community as part of the Particle Physics Data Grid (PPDG)[8], developing one of the first version 1.x SRM implementations, and now an SRM 2.1 implementation, both of which use Jasmine as the data store. SRM proof-of-concept has been demonstrated in SRM communication between different implementations among different labs.

5.2. User-Writeable cache

After the rewrite of the disk cache sytem, it became possible to consider allowing files to exist in the cache that are not yet on tape. This led to the development of the user-writeable cache, a system that is currently being tested.

The user-writeable cache maps a part of the cache file system space to a cache family. For example, /cache/farm_output/exp123 might be mapped in the cache database to a cache family for experiment 123 with an allocation of 10TB. Users, then, can add files under this part of the file hierarchy with a command line tool similar to the jput command that is used to write files to tape. It is important that the cache family used for holding these files have a strict allocation policy so that users do not completely fill up the cache system. Once a file is added to the cache in this way, the file is pinned in place and a record of who added the file and when is recorded.

Users can subsequently examine their files in the cache, retrieve copies of them from the cache, and then

choose to either commit them to tape, or unpin them and let the system garbage collect that space as it is needed.

It is important to point out that this is the same cache system that is used for the batch farm and the data movers. There is simply another layer of accounting information to track file ownership and other extended metadata.

There are two motivations for this system. One use for this is in large batch system runs where the output data needs some post-processing examination to decide if it should be committed to tape for permanent storage. A second use is to provide a high-throughput mechanism for moving output from many farm nodes to disk nodes in a way that spreads out the I/O load. Historically, the batch farm has been able to overload individual fileservers with output from many nodes at once.

6. System Performance

Jasmine is designed for high-throughput, sometimes at the expense of latency. Requests are ordered to make efficient use of the tape drives. In 0we show the number of bytes moved to or from tape over a period of several weeks. Peaks occur when the queue contains enough requests to keep all the tape drives busy. The troughs in the graph do not represent degraded performance, but cases where there is not sufficient work to keep all the tape drives and cache servers busy. 10TB/day seems to be the steady state performance of the system with a queue of about a day's worth of work. With modest hardware upgrades (faster staging disks, fewer drives per data mover) this number could be increased.

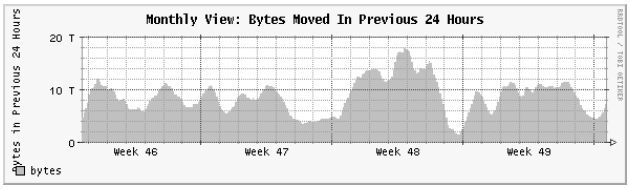

Figure 6. Typical Tape Data Volume

6.1. Data Access Patterns

We observe that our experimental data is typically accessed 10-20 times over its life. Accesses tend to come with some temporal locality. Because of this access pattern, the disk cache is useful during multiple runs through experimental data on the farm. As one data set is completed and another is accessed, the cache is cleared out to make space for other data. Some experiments require more passes through the data than others, improving cache efficiency.

The disk cache lifetime for batch farm processing has been fairly short, with files living in the cache for a few days to a week. If the cache size were increased by an order of magnitude, we would see a moderate increase in cache hits for large experiments.

7. Related work

Fermi Lab's Enstore[3] combined with DESY's dCache[5] provides similar functionality to Jasmine and handle data of approximately the same scale and metadata complexity. dCache also functions independently of Enstore as a disk-only mass storage solution in some Grid environments.

CERN's Castor[4] has historically had a different philosophy for queue processing, but is changing to meet LHC requirements, including a new queuing system and a different database backend. Castor has always offered a POSIX-like C library for access to files which gives more file system-like programmatic access to the mass storage system.

Enstore and Castor, like Jasmine, track metadata concerning data origin, use, and integrity. None of these systems tracks physics-related metadata, which requires knowledge of the stored data's structure. This metadata is handled by clients of the storage system that are used for analysis work.

It is interesting to note that all these storage systems are converging to an architecture involving significant disk cache on the front end, a complex database on the back end, and a scheduler to manage queuing in the presence of large numbers of requests.

Some systems, like SRB[12] have a much wider scope than Jasmine and offer digital library and metadata features not present in Jasmine.

8. Future work

We anticipate future work continuing in three areas: Addressing the needs of the growing batch farm; addressing the needs of the emerging data grid; and adapting to handle a wider variety of data access patterns and file sizes, including the handling of small files.

8.1. Small File Handling

One of the most significant challenges to the system is the handling of small files. Jasmine was designed as a mass storage system with the anticipation that 2GB files would be the norm. This is usually the case, but some new applications (Lattice QCD and certain experimental work) are testing the limits of the system to handle large numbers of files at once.

The original design of Jasmine called for files on the order of 2GB in extent. Handling smaller files (less than ~100MB) effectively has several performance barriers. First, Jasmine keeps significant per-file statistics in the database as the file's progress is tracked through the system. For tape performance reasons, small files require an aggregation layer that writes groups of small files to tape in some single-file archive format, like tar. Jasmine currently does not have an aggregator. Such an aggregator must also take care to update the database in a more efficient way that hides the latency of the per-file operations with a batch add facility.

Although this data is useful for mining accounting information from the database, it causes a lot of high-latency database interactions per file. Secondly, files are written individually on tape with file marks between them. The tape positioning and lack of streaming inherent in this process also slows the system. To properly handle small files in an efficient way without sacrificing the overall performance of the system will require reexamination of some key components.

8.2. Authorization and Authentication

The arrival of data grids challenges storage systems that have been designed with a UNIX-centric idea of user accounts. When a user writes a file into Jasmine, the file ownership and permissions must be tracked as long as the file persists in the system. This data may last longer than the UNIX user account that wrote it. The recent addition of grid user identities, typically represented as time-limited X509 certificates, compounds this problem by introducing a second type of user identity that could own a file. Storing simple Unix UIDs or usernames is not sufficient to maintain data ownership over time. We find it necessary to store user identity information beyond simple UIDs in the Jasmine database.

Jasmine maintains its own user identity database. This user identity can be mapped to Unix usernames, X509 certificate identities, or our own PKI-based identity. Allowing a many-to-one mapping of external identities to Jasmine identities helps to obviate the problem of long-term file ownership tracking at the expense of more accounting within Jasmine.

We anticipate that any fully grid-integrated storage system will need to do a similar level of user tracking. Further, we recognize the need for a more sophisticated authorization mechanism beyond this authentication-based user identity-tracking model.

9. Conclusion

Jasmine has successfully managed the data storage needs of Jefferson Lab since its introduction. The system has scaled well to meet the needs of new experiments and efforts at the lab. The cache system, when it was clear that it would not scale to meet emerging requirements, was replaced with a new system that has added new functionality used by the Auger batch system.

The core design of a database-centric, distributed system has not changed. The ability to add more data movers and cache nodes to accommodate more tape bandwidth and disk storage has allowed us to implement important hardware changes and upgrades with minimal impact to the user community.

Our chief operational problems have been from faulty hardware, and operating system specific problems including Linux NFS bugs and unexpected device drivers/kernel bugs.

We look forward to the continued development of Jasmine as needed to meet the growing edge of the lab's needs.

References

URLs in references were checked for validity on December 22, 2004.

[1] I. Bird, B. Hess, M. A. Kowalski, "Building the Mass Storage System at Jefferson Lab," *Proceedings of the 18th IEEE Symposium on Mass Storage Systems*. (Los Alamitos, CA: IEEE Computer Society Press, 2001)

[2] W. A. Watson, I. Bird, J. Chen, B. Hess, A. Kowalski, Y. Chen, "A Web Services data analysis Grid," *Concurrency and Computation: Practice and Experience*. (John Wiley & Sons, Ltd. 14:1303-1311, 2002)

[3] J Bakken, E. Berman, C. Huang, A. Moibenko, D. Petravick, "The Status of the Fermilab Enstore Data Storage System," *Computing in High Energy Physics 2004*.

[4] O. Bärring, B. Couturier, J.-D. Durand, S. Ponce, "Castor: Operational Issues and New Developments," *Computing in High Energy Physics 2004*.
[5] http://www.dcache.org
[6] http://auger.jlab.org/
[7] http://sdm.lbl.gov/srm-wg
[8] http://www.ppdg.net
[9] http://www.mysql.com
[10] http://jasmine.jlab.org/scicomp/
[11] http://java.sun.com/j2ee
[12] http://www.npaci.edu/DICE/SRB/