Building the Mass Storage System at Jefferson Lab

Ian Bird, Bryan Hess, Andy Kowalski
SURA/Jefferson Lab
12000 Jefferson Avenue,
Newport News, VA 23606
Ian.Bird@jlab.org, Bryan.Hess@jlab.org, Andy.Kowalski@jlab.org
tel +1-757-269-6224
fax +1-757-269-6248

Abstract

Thomas Jefferson National Accelerator Facility (Jefferson Lab) is a U.S. Department of Energy Facility [1] conducting Nuclear Physics experiments that currently have data collection rates of up to 20 MB/second. Future experiments, however, are expected to greatly exceed these rates. Post processing and data analysis produce similar amounts of data. Both the raw and processed data are stored on tape and need to be easily accessible. Between data collection and processing, the Mass Storage System at Jefferson Lab currently moves over 2 TB of data per day.

This paper describes the requirements, design, and implementation of the Jefferson Lab Asynchronous Storage Manager (JASMine), developed to support multi-terabyte per day I/O. JASMine is a lightweight, distributed, scalable, high performance, high capacity system. It is built around a relational database and is implemented almost entirely in Java. As a result, any system that supports a Java Virtual Machine can run JASMine, or JASMine services, with few if any code changes.

1 Introduction

For the past few years Jefferson Lab has made use of Open Storage Manager (OSM), a commercial Hierarchical Storage Manager from Computer Associates to manage data stored on tape. OSM, however, is no longer supported and has many shortcomings that limit its performance and scalability. Since it is not a distributed system, all the tape drives are required to be connected to a single server, making the server an I/O bottleneck.

To work around these shortcomings, we have developed a front end and user interface. The front end, called TapeServer, provides disk cache, cache managers, data staging, smart scheduling, and multiple OSM server support. The disk caching provided by the TapeServer improves the I/O throughput of the overall system. Smart scheduling provides a way to prioritize and limit user requests. Making multiple OSM servers appear as a single system to the user helps with the centralization of metadata as well as the ability to add additional tape drives on new systems. Multiple instances of OSM, however, require additional administration.

It has become essential to replace the functionality of OSM within the TapeServer software. Although other Mass Storage Systems and Hierarchical Storage Managers exist, none simply fit into the existing infrastructure and provide the needed functionality. A lot of the required functionality (disk-cache managers, data staging, and smart bottlenecks including network interfaces, server IO boards, processor power, and disk IO. To eliminate such bottlenecks, the system must consist of distributed data movers.

Distributed data movers will also eliminate many single points of failure. A crashed data mover will simply degrade the overall performance of the system instead of halting it all together.

2.5 Smart Scheduling

Access to data cannot always be granted on a first come first served basis. Because tape mounts take time and reduce the overall throughput of the system, requests for files on the same tape should be processed consecutively, while the tape is mounted. While this will help maximize throughput, it could also lead to job starvation. Therefore, the system must be able to prevent job starvation by tracking usage by users and hosts. Requests should first be prioritized by usage and then by available resources.

2.6 Monitoring and Management

System administrators and users need tools to allow them to monitor and react to the status of the system. Values monitored should include the status of each server and service, the data rates of each server, resource utilization, and the prediction of request execution.

The status of a server or service is either available or not, allowing administrators to identify downed services easily. The data rates of each service that moves data will show the throughput of the system as well as identify bottlenecks. Resource utilization will also identify bottlenecks caused by insufficient amounts of resources such as disk space and tape drives. The estimates of request execution time will provide users with an idea on when they can expect their request to begin execution. This information will also be used by external systems requiring estimates for the time required to access data. Such systems could include Grid applications.

In addition to the statistics provided by monitoring, the system must provide a way to generate statistics from history files or logs. Such information will include the number of requests during a given period of time, the amount of data moved, and the availability of services and hardware components.

Finally, a set of management tools will include the ability to add tapes to the tape library, eject tapes from the tape library, enable/disable data movers, enable/disable cache managers, and manipulate user requests.

2.7 Error/Exception Handling and Recovery

When an error occurs, the system needs to provide an informative description of the error and execute an appropriate action to work around the problem. Such action may include retrying actions, disabling services, disabling the use of certain hardware components, or alerting administrators of problems.

In the case of a database failure, the system needs to be easily recovered. Procedures for database and metadata backup and recovery will be required. The metadata must be

3.1 Overview

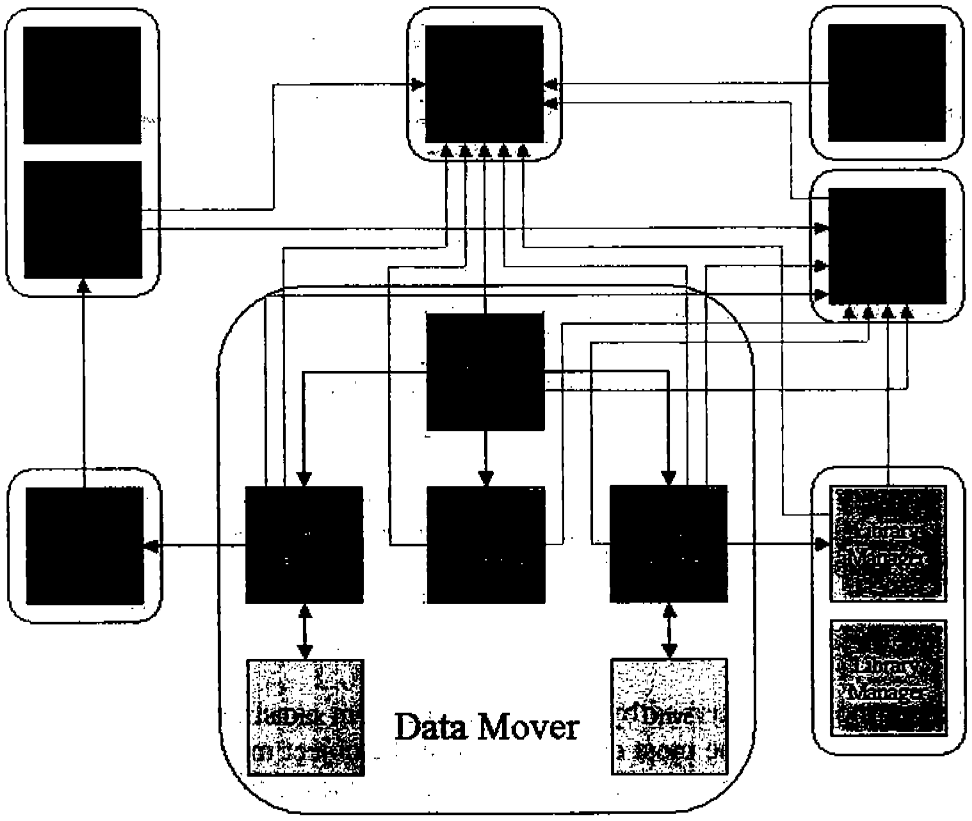

Figure 1.

3.2 Storage Objects

Storage objects fall loosely into two categories: The logical and the physical. Logical components are organizational constructs while physical components have a concrete aspect. A Storage object like a tape cartridge has both physical and logical attributes. The location, type of cassette, and status of the write-protect switch are physical attributes. The volume group to which a volume belongs and the file metadata contained on the volume are logical constructs. The union of all the attributes, both physical and logical, is the largest unit in the System, the Store. A store contains tape libraries, drives, volumes, bitfiles, and other items.

In figure 2 we show the inheritance diagram for Store objects. It may not be immediately clear why we would establish what is normally taught as an "is-a" relationship between, say, a BitFile and a Volume. One would not at first glance think that a bitfile "is a" volume, but consider a second interpretation of the inheritance model. A bitfile has a need for a superset of the attributes and operations that a volume has, and a bitfile always resides on a volume. It makes sense then to query a BitFile object for its volume number, or to ask if it is currently loaded in some tape drive.

classes are declared public and they make use of three privately declared methods, namely *refresh()*, *invalidate()*, and *needsLoad()* that are defined in every store-derived class. Because they are private they are not overridden in subclasses-- should a Bitfile object call *isMissing()*, a Volume method, this will in turn call the *refresh()* private method in Volume. Should the BitFile object then call *getWriteVols()*, a VolumeSet method, the *refresh()* that is private to VolumeSet will be called. If the *refresh()* method were not private then the overriding would break this implementation of closures. This is done so that we can preserve per-table loads from the database and maintain information for each table about its cache state even when the object contains caches of several database tables as a BitFile or Volume object must.

3.3 Authentication

The original TapeServer software required all clients to be part of Jefferson Lab's central computing environment and relied on host-based authentication. JASMine exchanges this for a framework of pluggable authenticator modules that establish identity on a network connection before it is returned to the application. Using these authenticators provides a low-level security API to establish identity before any other transfer takes place. For instance, off-site users could be required to authenticate with a username and password that are encrypted before transmission, but this rigor is not necessary for on-site transfers between persistent servers, they share authorization information through the database and need only a permissive authenticator for server-to-server traffic. Other mechanisms could involve ssh-keys, kerberos, or other single-sign on systems associated with Grid projects. This pluggable authentication scheme is applied to java RMI connections using custom socket factories and in peer-to-peer communication using the file moving protocol that we describe in section 3.5.1.

3.4 Tape Label and Data Formats

Like most tape storage systems, JASMine makes use of tape labels. Labels allow systems to verify that the correct tape has been mounted in a drive. Labels are also used to gather information required to utilize a given tape. Such information would include, but is not limited to, block size, data format, and file position.

JASMine makes use of ANSI X3.27-1987 standard labels [4]. The ANSI standard provides a standardized labeling scheme for the interchange of tapes between sites. Most tape storage systems make use of ANSI labels. Such systems can therefore make use of tapes created by JASMine.

The ANSI X3.27 standard, however, is old. As a result, certain fields in the 80-byte labels are not large enough to handle the values required by today's storage systems. For example, the StorageTek Redwood tapes drives are capable of using block sizes of 256 KB and cartridges capable of storing 50 GB of data. Such cartridges can easily hold more than 100,000 files. The block size and file number can thus be too large to fit in their respective fields in the ANSI label definition.

To get around these limitations JASMine specifies zeros in the fields that are too restrictive and stores the needed information in a hidden JASMine label located after the 80-byte ANSI label and in a database. The ANSI standard states in section 6.2.1:

minimum and a fast, thread-safe double buffer is used to avoid starvation of tape drives under system load.

There are other cases where the stream abstraction proves useful. Within the data mover, we pass streams through tape formatter objects that completely define the layout of a tape. If we need to read an OSM tape or an HPSS tape, we merely choose the correct tape formatter connecting tape I/O streams to one end and our job to the other. The tape label is part of the formatter object. This is the mechanism used to implement ANSI tape labels.

3.5.1 Protocol

There is extensive code reuse in the file-moving portions of the system. This is achieved by defining a simple but extensible protocol for file movement. JP, the JASMine file moving protocol, uses Java serialized objects as messages passed over streams. The protocol is strictly synchronous to avoid complexity. Interaction follows a basic message/acknowledgement scheme. When asynchrony is necessary additional connections and protocol instances are created. Multithreading is used to keep the protocol simple while allowing for multiple outstanding requests between servers. For performance reasons our implementation of JP always falls back to a simple raw data transfer over tcp when bulk data transfers are made. That is, Java serialized objects are only used for metadata transmission.

JP includes hooks that allow its behavior to be tightly controlled in three ways. First, an Authenticator interface is defined to allow for pluggable authentication. Secondly, a pluggable file transfer policy mechanism may be defined to guarantee expected behavior. As an example, a request only meant to download files will be blocked by policy from uploading or deleting files. The final means for altering the protocol includes a hook for a message dispatcher. This hook allows for protocol extension, and is the way we reuse the File Moving code to create Cache Servers, Stage Disks, and interactive FTP-like file transfer agents. New messages with hooks into the appropriate database(s) are created and the whole of the JP code is reused.

3.6 Metadata

Metadata is important to both the user and the system. Users need metadata to determine which file they want and what resources are required to obtain the file. The system uses metadata to determine which tape the file is on and where the file is located on tape. Therefore, the metadata needs to be accessible without having to read the tape. Three copies of the metadata are created. They are stored in a database table, on tape along with the file, and in a flat file, called a stub file, that represents the file to the user.

The metadata consists of the following information:

1. Bit File Identification (Unique string of alpha characters and numbers).
2. Original full path file name.
3. Owner.
4. Group.
5. Permissions.

manager also returns informational requests that can be immediately satisfied without queuing; such as job status and system utilization statistics.

For queued requests the scheduler will order them in the database according to some policy, and dispatcher threads will select jobs to run. Subsequent communication with the user commands will come from the data mover(s) that turn the requests into jobs. The request manager will not be contacted again after the initial query.

3.9 Scheduler

The scheduler is hierarchical. Groups of hosts or users are assigned a single share. These groups are then divided into subgroups that are assigned shares for that level. Since the share of the parent group is checked first, the shares assigned to subgroups are effectively sub-shares of the parent group's share. The result is a hierarchical share tree.

The shares assigned can be dynamically changed so that resources can be reallocated as needed. To ensure that requests from certain hosts, like batch farm hosts, are started promptly, host groups can be created and assigned large share values. This will prevent farm hosts from sitting idle while waiting for data from the tape library. Smaller share values can be assigned to groups of desktop systems. Similar actions can be taken by changing user shares.

Two share trees exist. There is one share tree for hosts and one for users. The scheduler assigns priorities to hosts to prevent the overuse of the tape library by remote mini-farms and desktop systems for which the network bandwidth and system efficiency is limited or not known. Overuse by inefficient systems would have a negative impact on the Jefferson Lab batch and analysis farms. Priorities are also calculated for the users to prevent resource hogging by one or more individuals.

Each group, host, and user is assigned a priority. This priority is calculated by using the following formula:

$$\text{priority} = \text{share} \;/\; (\,.01 + (\text{num\_a} * \text{ACTIVE\_WEIGHT}) + (\text{num\_c} * \text{COMPLETED\_WEIGHT})\,)$$

where ACTIVE_WEIGHT, and COMPLETED_WEIGHT are configurable parameters. The value of share is static and assigned to each group, host, or user. The number of active requests is represented by *num_a*. The number of completed requests is represented by *num_c* and is calculated for a given time period. The priorities represent the desired scheduling order for requests from hosts and users based on assigned shares and usage during that time.

When deciding on the ordering of the requests, the priorities of the host groups are first considered. This means that a low priority user will have their request dispatched before a high priority user if the submitting host for the low priority user has a higher priority than the submitting host used by the high priority user. This is to ensure that individual user requests from hosts throughout the Jefferson Lab network are not taking data resources away from the batch and analysis farms.

system with no overhead. No single failure will bring the system down. Even the database may be replicated.

The terms "cache server" and "cache client" used above need some clarification. A cache client is a component that initiates the connection; a server is a component that accepts the connection. There are situations where one cache server is required to initiate a transfer on behalf of some agent. This third-party copy means a server must also occasionally act as a client, not an uncommon occurrence in peer-to-peer networking models. This kind of behavior requires transitivity of authorization credentials.

The cache management software is the generalization of three distinct uses for disk in the mass storage system. When we say disk, we typically mean a large pool of disks in some RAID arrangement.

A first type of disk storage is the user-invisible stage disk used with data movers as a buffer in front of tape drives. Files are added and removed from stage disks using a reference counting algorithm. A file is placed on a disk and its reference count is incremented for each thread using it. A file is only removed from the stage disk when its reference count is zero and it has the oldest access time of any file on the disk.

A second type of disk is the least-recently-used file cache. Files are added to this cache area (from which we get the overall cache disk name) by user request. These disks are perpetually full and contain a subset of the files stored on tape. These disks are divided into organizational groups that usually correspond to experiments. When a request is received by JASMine to add a new file to a group's cache disk space must first be made for the new file. This is done by determining the oldest file in the group by access time and removing it. This removal process is repeated until there is enough space for the file to be added.

This is a relatively heavyweight operation. The initiator finds the least recently used file by performing a JP scatter-gather operation in which each of the cache servers containing data for a group must independently search and find its least recently used file. This collection of files is then forwarded to the initiator as the gather portion of the operation. Since the multiple searches are done in parallel, the search time scales linearly with the number of servers. Such a distributed operation is also ripe with opportunities for race conditions, requiring careful attention to resource reservation.

Either of these first two types of disk could be used to pre-load data for farm jobs. We currently use a sufficiently large LRU disk pool to cache data from tape before we start batch processing of a job. This is advantageous because jobs are never blocked on our CPU farm waiting for tape. Our batch farm system enforces this staging making more efficient use of the CPUs.

The final use of the disk management software is for persistent data. There are certain classes of data that are added to a user-accessible area with no intention of removal. These typically represent the final output of physics data processing. These files, stored in the silo, are also cached online with no intention of removing them. For this we define reservation and job claiming, which must be done in a more distributed fashion using database table locking, database atomic inserts, or a distributed locking protocol, which is far more complex.

A drive can be reserved for reading only, writing only, use by a select volume sets, use by a select storage group, or some combination of these. To make sure that high priority users can write their data to tape immediately upon request, a drive can be reserved as write-only for the volume set that the users are using. This is simply done by maintaining reservation fields in the database for a given drive. Other components, however, query the drive manager for drive status. The dispatcher, for example, queries the drive manager for a list of available drives and their reservations. The dispatcher would then only start jobs that can make use of the drives available based on their reservations.

3.11.2.2 Drive

The Drive object is responsible for manipulating a loaded tape drive. It keeps track of file position on tape, seeks to a given location on tape, reads and writes data to tape to and from tape, and ejects a tape from a drive. When a job is passed a Drive object from the DriveManager, it simply uses OutputStreams and InputStreams from the Drive object to write and read data to and from tape.

3.11.3 Volume Manager

Volume managers provide the data mover and its components with information about volumes. This includes the type of volume it is, its location, its state, and its ownership. The volume manager is also responsible for allocating unused volumes to jobs when they need them.

3.11.4 Dispatcher

There is one dispatcher per data mover. It has the responsibility to inspect jobs in the Request database, inspect local resources, and then claim jobs as it can process them. Roughly, the dispatcher does the following:

1. Check that job starting is enabled.
2. Get the list of available drives from the Drive Manager
3. Check for secure, local disk space.
4. Get a list of pending requests.
5. Claim a drive (if failure, start over at (1)).
6. Reserve disk space (if failure, release the drive and return to (1)).
7. Claim the job(s) (if failure, release disk space, release drive, return to (1)).
8. Start the job thread.

By having a dispatcher on every data mover, the system will not stop dispatching jobs because of a single system crash or failure. This also cuts down on the amount of information on available resources a dispatcher must keep track of. The processing required to determine which job must run next is distributed across the data movers and thus lessens the CPU resources required by a single dispatcher.

parameters that is correct for the calling machine. This allows global configuration changes to be made at a single point with no worry that some client was forgotten.

4 Future Work/Outlook

During the development of JASMine, some changes were suggested based on lessons learned from testing. In addition, other projects were started that influenced the design of JASMine.

The metadata stored in OSM needs to be imported into JASMine so that OSM can be decommissioned. Making use of newer tape drives and replacing OSM were the driving forces behind the development of JASMine.

Users have shown an interest in being able to access the data stored on Cache Managers via standard file copy protocols like ftp. To accommodate such requests, we will implement gateways to the Cache Managers that provide user interfaces for standard protocols to access data stored on them. Gateways will make implementing various protocols easier and will not require modifications to JASMine itself. These gateways will also provide external access to Cache Managers hidden behind firewalls. As part of Jefferson Lab's involvement in the Particle Physics Data Grid collaboration [5], we will provide interfaces to permit remote access to data on tape or on the local cache systems.

The naming scheme used to store data files within Cache Managers will be changed to include the entire full path name of the files original location unless the user provides an alternate name. This helps lessen the probability of two different files being given the same full path name for storage on disk. Although this is not a problem for JASMine alone, it might be a problem if the Cache Managers are used to store data for JASMine and other applications at the same time.

An XML based API will be created and implemented in JASMine. This will remove the requirement that clients be written in Java and use RMI or serialized Java objects for communication.

4.1 Web Clients

Users have shown great interest in having web-based clients that retrieve data files from JASMine. Such clients will authenticate the user and submit a request to get file or have files placed on a cache disk. In addition, web clients will be created to submit batch jobs to the batch farm. Such requests will gather all the required data files on cache disks before execution begins.

5 Conclusions

JASMine is a Mass Storage System that will scale to meet the needs of current and future high data rate experiments at Jefferson Lab. It is a lightweight, distributed, scalable, high performance, and high capacity system. The distributed and scalable design of JASMine make it suited to dealing with large volumes of data. The distributed nature of the Data Movers and Cache Managers are also essential to ensure the overall performance of the system. As the system is very modular, components can easily be used in other