Storage Resource Sharing with CASTOR

Olof Bärring, Ben Couturier, Jean-Damien Durand, Emil Knezo, Sebastien Ponce
CERN
CH-1211 Geneva 23, Switzerland
Olof.Barring@cern.ch, Ben.Couturier@cern.ch, Jean-Damien.Durand@cern.ch,
Emil.Knezo@cern.ch, Sebastien.Ponce@cern.ch
tel: +41-22-767-3967
fax: +41-22-767-7155

Vitaly Motyakov
Institute for High Energy Physics
RU-142281, Protvino, Moscow region, Russia
motyakov@mx.ihep.su
tel: +7-0967-747413
fax: +7-0967-744937

Abstract:
The Cern Advanced STORage (CASTOR) system is a hierarchical storage management system developed at CERN to meet the requirements for high energy physics applications. The existing disk cache management subsystem in CASTOR, the *stager*, was developed more than a decade ago and was intended for relatively moderate (large at the time) sized disk caches and request load. Due to internal limitations a single CASTOR stager instance will not be able to efficiently manage distributed disk cashes of several PetaBytes foreseen for the experiments at the Large Hadron Collider (LHC) which will be commissioned in 2007. The Mass Storage challenge comes not only from the sheer data volume and rates but also from the expected request load in terms of number of file opens per second. This paper presents the architecture design for a new CASTOR stager now being developed to address the LHC requirements and overcome the limitations with the current CASTOR stager.

Efficient management of PetaByte disk caches made up of clusters of 100s of commodity file servers (e.g. linux PCs) resembles in many aspects the CPU cluster management, for which sophisticated batch scheduling systems have been available since more than a decade. Rather than reinventing scheduling and resource sharing algorithms and apply them to disk storage resources, the new CASTOR stager design aims to leverage some of the resource management concepts from existing CPU batch scheduling systems. This has led to a pluggable framework design, where the scheduling task itself has been externalized allowing the reuse of commercial or open source schedulers.

The development of the new CASTOR stager also incorporates new strategies for data migration and recall between disk and tape media where the resource allocation takes place just-in-time for the data transfer. This allows for choosing the best disk and network resources based on current load.

1. Introduction

The Cern Advanced STORage (CASTOR) system[1] is a scalable high throughput hierarchical storage system (HSM) developed at CERN. The system was first deployed for full production use in 2001 and is now managing about 13 million files for a total data volume of more than 1.5 PetaByte. The aggregate traffic between disk cache and tape archive usually exceeds 100 MB/s (~75% tape read) and there are of the order of 50,000 – 100,000 tape mounts per week. CASTOR is a modular and fault-tolerant system designed for scalable distributed deployments on potentially unreliable commodity hardware. Like other HSM systems (see for instance [2], [3], [4], [5]) the client front-end for file access consists of a distributed disk cache with file servers managed by the CASTOR stager component, and an associated global logical name-space contained in an Oracle database (MySQL is also supported). The backend tape archive consists of a volume library (Oracle or MySQL database), tape drive queuing, tape mover and physical volume repository. The next section lists some of the salient features of today's CASTOR system and the installation at CERN. Thereafter are listed some requirements that the CASTOR system has to meet well before the experiments at the Large Hadron Collider (LHC) start their data taking in 2007.

The remaining sections (4 - 6) describe the new CASTOR stager (disk cache management component), which is being developed [6] to cope with the data handling requirements for LHC. It is in particular the expected request load (file opens) that requires some special handling. The design target is to be able to sustain *peak* rates of the order of 500 - 1000 file open requests per second for a PetaByte disk pool. The new developments have been inspired by the problems arising with management of massive installations of commodity storage hardware. It is today possible to build very large distributed disk cache systems using low cost Linux fileservers with EIDE disks. CERN has built up farms with several 100s of disk servers with of the order of 1TB disk space each. The farming of disk servers raises new problems for the disk cache management: request scheduling, resource sharing and partitioning, quality of service management, automated configuration and monitoring, and fault tolerance to unreliable hardware. Some of those problems have been addressed and solved by traditional batch systems for scheduling of CPU resources. With the new CASTOR stager developments described in this paper, the CASTOR team leverages the work already done for CPU scheduling and applies it for the disk cache resource management. This is achieved through pluggable framework design where the request scheduling is delegated to an external component. Initially LSF from Platform Inc [7] and Maui from Supercluster [8] will be supported.

The new system is still under development and only exists as an advanced prototype. The final system is planned for 2Q04.

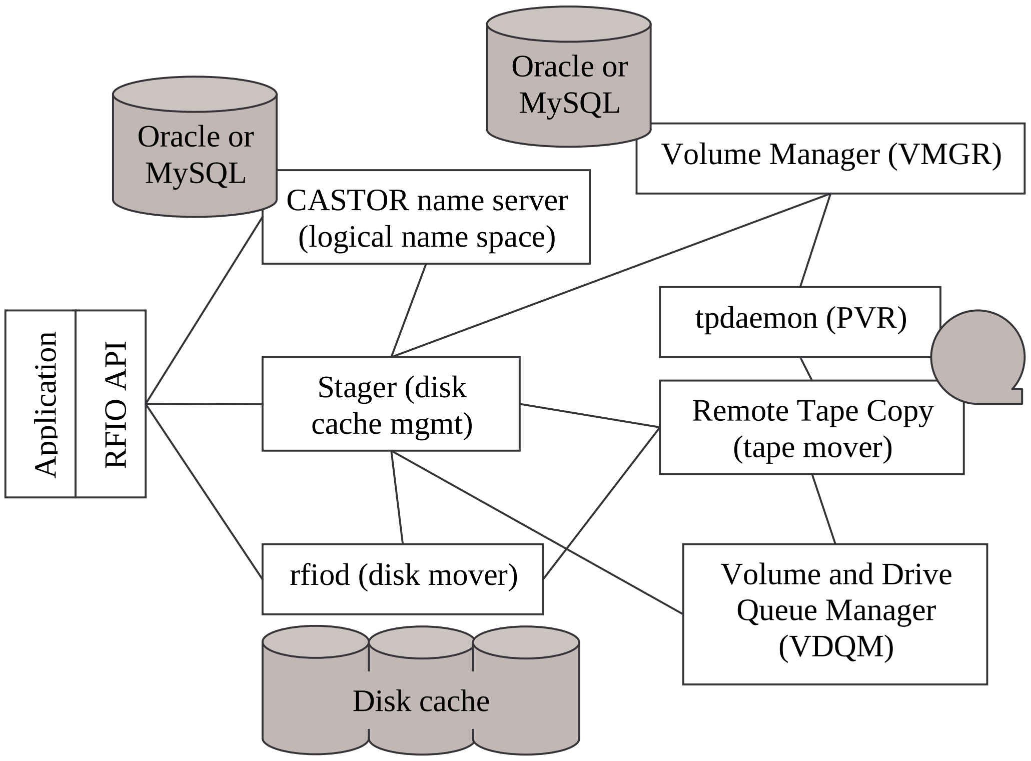

Figure 1: CASTOR components and their interactions

2. The CASTOR architecture and capabilities

The CASTOR software architecture is schematically depicted in Figure 1. The client application should normally use the Remote File IO (RFIO) library to interface to CASTOR. RFIO provides a POSIX compliant file IO and metadata interface, e.g. *rfio_open(), rfio_read(), rfio_write(), rfio_lseek(), rfio_stat()*, etc. For bulk operations on groups of files, a low-level stager API is provided, which allows for passing arrays of files to be processed.

The Name server provides the CASTOR namespace, which appears as a normal UNIX filesystem directory hierarchy. The name space is rooted with "/castor" followed by a directory that specifies the domain, e.g. "/castor/cern.ch" or "/castor/cnaf.infn.it". The next level in the name space hierarchy identifies the hostname (or alias) for a node with a running instance of the CASTOR name server. The convention is *nodename* = *"cns"* + *"directory name"*, e.g. "/castor/cern.ch/user" points to the CASTOR name server instance running on a node *cnsuser.cern.ch*. The naming of all directories below the third level hierarchy is entirely up to the user/administrator managing the sub-trees.

The file attributes and metadata stored in the CASTOR name space include all normal POSIX "stat" attributes. The CASTOR name server assigns a unique 64 bit file identifier to all name space elements (files and directories). File class metadata associates tape migration and recall policies to all files. These policies include:

- Number of tape drives to be used for the migration

- Number of copies required on different media. CASTOR supports multiple copies for precious data

In order to avoid waste of tape media, CASTOR supports segmentation of large files over several tapes. The tape metadata stored in the CASTOR name space are currently:

- The tape VID
- Tape position: file sequence number and blockid (if position by blockid is supported by the tape device)
- The size of the file segment
- The data compression on tape

There is also a ‘side’ attribute for (future) DVD or CD-RW support. The tape metadata will soon be extended to include the segment checksum. CASTOR does not manage the content of the files but a special “user metadata” field can be attached to the files in the CASTOR name space. The user metadata is a character string that the client can use for associating application specific metadata to the files.

The RFIO client library interfaces to three CASTOR components: the name server providing the CASTOR namespace described in previous paragraph; the CASTOR stager, which manages the disk cache for space allocations, garbage collection and recall/migration of tape files; the RFIO server (rfiod), which is the CASTOR disk mover and implements a POSIX IO compliant interface for the user application file access.

All tape access is managed by the CASTOR stager. The client does not normally know about the tape location of the files being accessed. The stager therefore interfaces with:

- The CASTOR Volume Manager (VMGR) to know the status of tapes and select a tape for migration if the client created a new file or updated an existing one
- The CASTOR Volume and Drive Queue Manager (VDQM), which provides a FIFO queue for accessing the tape drives. Requests for already mounted volumes are given priority in order to reduce the number of physical tape mounts
- The CASTOR tape mover, Remote Tape COPY (RTCOPY), which is a multithreaded application with large memory buffers, for performing the copy between tape and disk
- The RFIO server (rfiod) for managing the disk pools

The CASTOR software is an evolution of an older system, SHIFT, which was CERN’s disk and tape management system for almost a decade until it was replaced by CASTOR in 2001. SHIFT was awarded ‘21st Century Achievement Award by Computerworld in 2001’ [9].

The CASTOR software has been compiled and tested on a large variety of hardware: Linux, Solaris, AIX, HP-UX, Digital UNIX, IRIX and Windows (NT and W2K). The CASTOR tape software supports DLT/SDLT, LTO, IBM 3590, STK 9840, STK9940A/B tape drives and ADIC Scalar, IBM 3494, IBM 3584, Odetics, Sony DMS24, STK Powderhorn tape libraries as well as all generic SCSI driver compatible robotics.

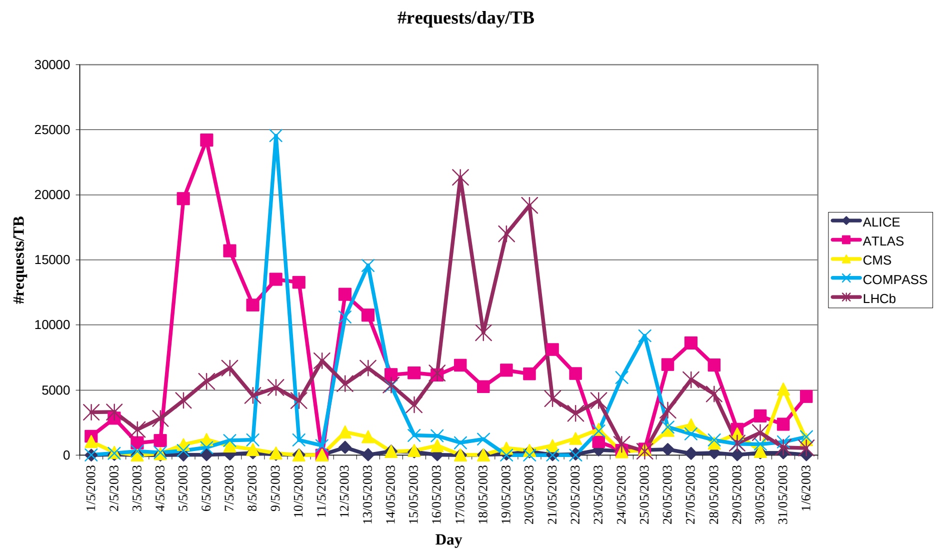

Figure 2: The number of requests per day normalized to TB of disk pool space. The plot shows one month of activity for 5 different stagers. ALICE, ATLAS, CMS and LHCb are the four LHC experiments whereas COMPASS is a running heavy ion experiment.

The CASTOR RFIO client API provides remote file access through both a POSIX I/O compliant interface and a data streaming interface. The former is usually used by random access applications whereas the latter is used by the tape mover and disk-to-disk copy. At the server side the data streaming mode is implemented with multiple threads overlaying disk and network I/O. When using the RFIO streaming mode the data transfer performance is normally only limited by hardware (see [10] for performance measurements). Parallel stream transfers are not supported since this is not required by High Energy Physics applications. An exception is the CASTOR GridFTP interface, which does support parallel stream transfers in accordance with the GridFTP v1.0 protocol specification.

3. Requirements for the LHC

CASTOR is designed to cope with the data volume and rates expected from the LHC experiments. This is frequently tested in so called ‘data challenges’ that the CERN IT department runs together with the experiments. In late 2002, the ALICE experiment ran a data challenge with a sustained rate of 280 MB/s to tape during a week [11]. The CERN IT department has also shown that the CASTOR tape archive can sustain 1 GB/s during a day [12].

In addition to the high data volume and rates generated by the LHC experiments, it is expected that the physics analysis of the data collected by LHC experiments will generate a very high file access request load. The four LHC experiments are large scientific collaborations of thousands of physicists. The detailed requirements for the LHC physics analysis phase are unknown but from experience with previous experiments it can be expected that

- The data access is random
- Only a portion of the data within a file is accessed
- The set of files required by an analysis application is not necessarily known
- A subset of the files are hit by many clients concurrently (hotspots)

The files must therefore be disk resident and since a substantial part of the physics analysis will take place at CERN it will result in a high request load on the CASTOR stager. Since the detailed requirements are unknown it is difficult to give an exact estimate for the expected request load on CASTOR. In order to obtain a design target the activity of five running instances of the current CASTOR stager was observed during a month, see Figure 2. The rates are normalized to the size of the disk pools in TB. The figure shows a peak rate of about 25,000 requests/day (0.3 requests/second) per TB disk. The average was about 10 times lower: 3,400 requests/day per TB disk. The associated data rate to/from the disk cache depends on the file sizes and the amount of data accessed for each file. The data volume statistics from the period shown in Figure 2 is no longer available but looking at a more recent period it was found that for the ATLAS stager the average data transfer per request is of the order 20MB (400GB for a day with 20,000 requests) whereas for LHCb it was only 3MB (50GB for a day with 17,000 requests).

Assuming that the number of file access requests scales linearly with the size of the disk cache, the peak request rate for a PetaByte disk cache would be of the order 300 requests/second. The objective for the new CASTOR stager is therefore to be able to handle peak rates of 500 – 1000 requests per second. Under such load it is essential that the new system is capable of request throttling, which is not the case for the current CASTOR stager. It is also known that the current CASTOR stager catalogue will not scale to manage more than ~100,000 files in the disk cache. Those shortcomings have already today led to a deployment of many CASTOR stager instances, each with its dedicated moderate size (5-10TB) disk cache. The disadvantages are manifold but most important are:

- Each stager instance has its dedicated disk cache and this easily leads to suboptimal use of the resources where one instance runs at 100% and lacks resources while another instance may be idle.
- The configuration management becomes complex since each stager instance has its own configuration and the clients must know which stager to use.

4. New CASTOR stager architecture

The new CASTOR stager, which is currently being developed at CERN, is specifically designed to cope with the requirements listed in the previous section. This section briefly describes some of the salient features of the new architecture.

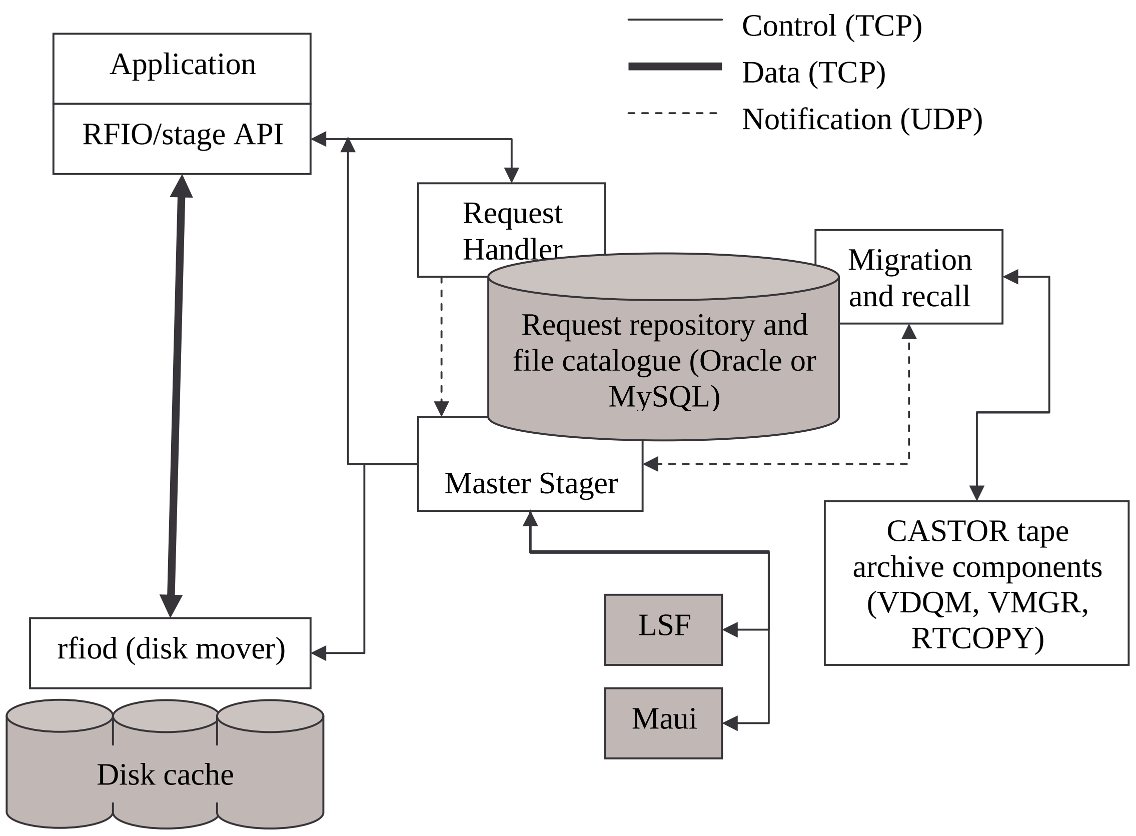

Figure 3: Overview of the new CASTOR stager architecture

4.1. Overview

Figure 3 shows a schematic view of the new architecture. The client application uses the RFIO API or the lower level stager API (for multi-file requests) to submit the file access request. Although it is not shown in the Figure 3, the CASTOR name server is called for all operations on the CASTOR name space as was shown in Figure 1. The request handler, master stager and migration/recall components communicate through a database containing all file access requests and the file residency on disk (catalogue). To avoid too frequent polling, unreliable notification is used between the modules when the database is being updated. The role of the request handler is to insulate the rest of the system from high requests bursts. The master stager is the central component that processes the file access requests and delegates the request scheduling decision to an external module. The first version of the new CASTOR stager will have resource management interfaces to LSF [7] and Maui [8]. The migration/recall components retrieve the tape related information and contacts the CASTOR tape archive (see Figure 1) to submit the tape mount requests.

Along with the new stager developments, it was decided to improve the security in CASTOR. However, for CERN physics applications, confidentiality is not a main requirement, and encrypting all the data during transfer would add significant load on the

CASTOR servers. Therefore, strong authentication of users in RFIO, stager and the CASTOR name server will be implemented using standard security protocols: Kerberos-5, Grid Security Infrastructure, and Kerberos-4 for compatibility with the current CERN infrastructure. Furthermore the architecture of the new stager and its interface with RFIO will allow only authorized clients (file access request has been scheduled to run) to access the files on the disk servers managed by the stager.

Below follows a more detailed description of the components and concepts used in the design of the new CASTOR stager architecture.

4.2. Database centric architecture

In the new CASTOR stager, a relational database is used to store the requests and their history, to ensure their persistency. The stager code is interfaced with Oracle and MySQL, but the structure allows plugging-in other Relational Database Servers if necessary. Furthermore, the database is also used for the communication between the different components in the stager (request handler, stager, migration/recall…)
This database centric architecture has many advantages over the architecture of the current stager.

From a developer's point of view:

- The data locking and concurrency when accessing the data are handled by the RDBMS: This makes the development/debugging of the code much easier.
- The database API is used to enter or read requests. The CASTOR application does not have to know whether the database is local or remote, the database API takes care of that (e.g. the Oracle API uses shared memory if the database is local or SQLNet otherwise).

From the CASTOR system's administrator point of view:

- The database being the central component in the architecture, the coupling between the different components is very low.
- This, as well as the persistence of the requests, and the proper use of status codes to identify their state, allows stopping and restarting the system, or specific components at any time. The administration is made easier.

For the CASTOR user, the overall scalability of the system is greatly improved as:

- RDBMS can cope with large amount of data. If the database tables are properly designed, the system should not have huge performance penalty when dealing with a large number of requests.
- The stager software components are stateless, which allows running several instances on different machines without problems. The real limit to the scalability is the database server itself.
- The database software is prepared for high loads, and there are some cluster solutions that can help coping with such problems.

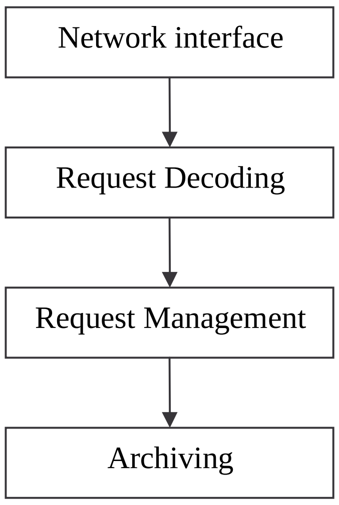

Figure 4: Request handler architecture

There are however some drawbacks to this architecture:

- The latency for each call is higher than with the current architecture, as the stager components have to access the database.
- The database server is a central point of failure and has to cope with a very high load. However, database technology is widely deployed and it is possible to get database clusters to mitigate that risk.

Furthermore, just using the database for communication between the components would force them to frequently poll the database, which creates unnecessary load, and possibly increase the latency of the calls to the stager. Therefore, in the current stager prototype, a notification mechanism using UDP messages has been introduced. This mechanism is very lightweight and does not create a big coupling between the different components as UDP is connectionless.

4.3. Request handler

The request handler handles all requests in CASTOR, from reception to archiving. The main purpose of the Request Handler is to shield the request scheduling from the possible request bursts. The request handler is also responsible for the decoding of the requests and storing them in the database. At all different stages in the request processing the requests are stored and updated in the database. The Request Handler architecture is sketched in Figure 4.

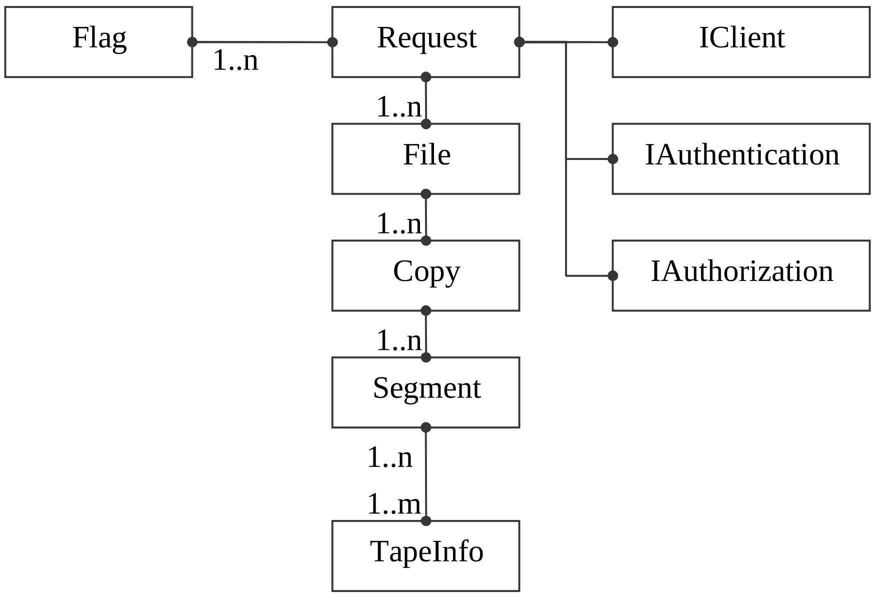

Figure 5: Request content

The **Network Interface** is responsible for handling new requests and queries coming from the clients. This is the component realizing the isolation against request bursts. It is designed to sustain high rates (up to 1000 requests per second). Therefore the only thing it does is to store the raw request as a binary object into a database for further processing by the request decoder. There are only two transactions on this table per request: one insert and one retrieve + delete. To guarantee the best insertion rate this database instance may be deployed separately from the rest of the request repository.

The **Request Decoding** takes place asynchronously. The decoded requests are stored into a database where they can be easily used and queried by the rest of the system, especially by the master stager scheduling component.

When a request is selected by the scheduling component for processing, it is moved to the **Request Management** components, which associate data to each request describing its status (flags), its clients (authenticated and authorized) and the tapes possibly required (see Figure 5).

At last, when the processing of a request is over, it is archived by the **Archiving** component.

The request handler manages two interfaces: the network interface that is used by external clients, and the interface to the Request Manager for internal communication with the other components of CASTOR:

- The network interface component allows clients to send requests to CASTOR. The API used is very much inspired by the SRM standard [13] so it will not be described in any detail here. Command line scripts are also provided.
- The internal interface has two parts:
    - The first part allows the stager component to handle requests and to update their metadata
    - The second part is used by the migration/recall component to get the list of tapes and segments of file to be migrated and to update the tape status.

4.4. Master stager and externalized scheduling plug-in interface

The master stager is central in the new architecture. It has two main functions:

- Manage all resident files and space allocations in the disk cache
- Schedule and control all client access to the disk cache

The master stager maintains a catalogue with up-to-date information of all resident files and space allocations in the disk cache. The catalogue maintains a mapping between CASTOR files (paths in the CASTOR name space) and their physical location in the disk cache. The mapping may be one-to-many since, for load-balancing purpose there may be several physical replicas of the same CASTOR file. An important difference to the current CASTOR stager is that the catalogue is implemented with a relational database. Oracle and MySQL are supported but other RDBMS systems can easily be interfaced. This assures that the system will scale to large disk caches with millions of files.
When the master stager gets notifications from the request handler that there are new requests to process, it uses the request handler internal interfaces to retrieve the requests from the database. The basic processing flows in the master stager are depicted in Figure 6:

- If the request is for reading or updating an existing CASTOR file, the master stager checks if the requested file is already cached somewhere in the disk cache. In case the file is already cached, the request is submitted to the external scheduler and queued for access to the file in the disk cache.
- If the request is for creating a new file, the entry is created in the CASTOR name space and the request is submitted to the external scheduler like in the previous case.
- If the request is for reading or updating an existing CASTOR file, which is not already cached, the master stager marks the file for recall from tape and notifies the recaller. Once the file has been recalled to disk the request is submitted to the external scheduler like before.

The scheduling of the access depends on the load (CPU, memory, I/O) on the file server and disk that holds the file. If the load is too high but another file server is ready to serve the request, an internal disk-to-disk copy may take place replicating the file to the selected disk server.

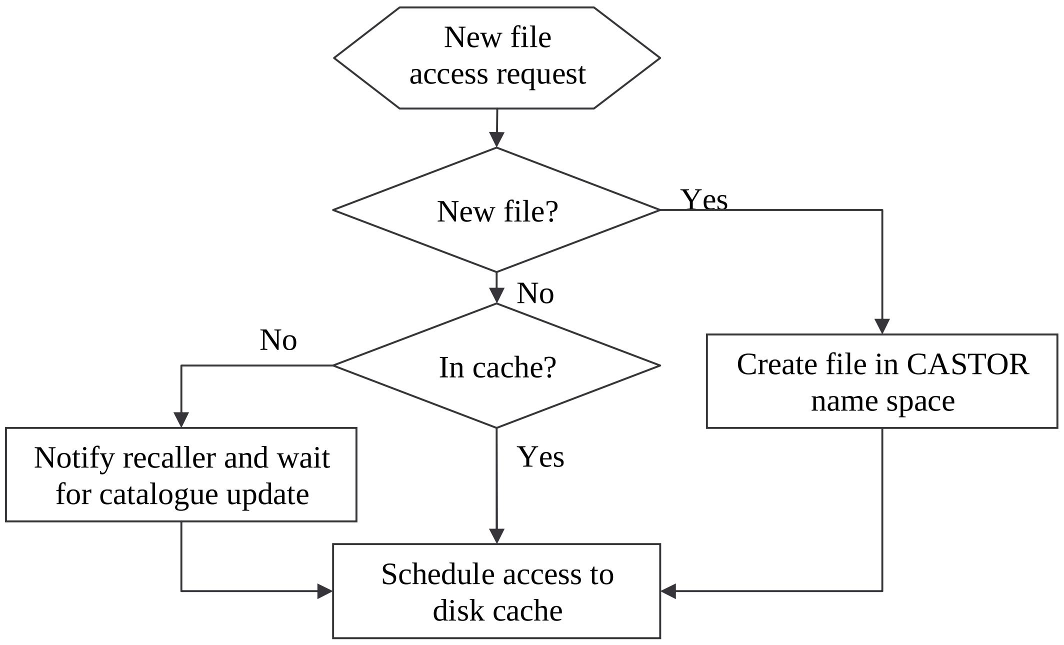
New file access request
New file?
Yes
No
No
In cache?
Create file in CASTOR name space
Notify recaller and wait for catalogue update
Yes
Schedule access to disk cache

Figure 6: Master stager basic flows

Request throttling and load-balancing are already established concepts in modern disk cache management [2] and distributed file systems [14], [15]. The novelty of the new CASTOR stager architecture is the delegation of the request scheduling to an external module, which allows for leveraging existing and well-tested scheduling systems. In the simplest case, the external scheduling module could be a FIFO queue. However an important advantage of using sophisticated scheduling systems like [7], [8], [16] is that they also provide established procedures for applying policies for quality of service and resource sharing. Interfacing the disk resource management with CPU batch scheduling systems prepares the ground for eventually marrying the CPU and storage resource management.

Because of the close resemblance between farms of CPU servers and farms of commodity file servers it is possible to re-use the CPU batch scheduling systems almost out-of-the-box (see Section 5). The only adaptation required was to accommodate for the fact that a disk cache access request not only targets a file server (node) but also the file system (disk) holding the file or space allocation. While the local pool (or worker directory) file-system is part of normal CPU scheduling attributes, it is normally assumed that there is only one file-system per node. A file server may have several file systems mounted. The CASTOR development team has established excellent collaborations with LSF [7] and Maui [8] developers. Both groups were interested in the problem of disk storage access scheduling and extending their systems to support file-system selection in the scheduling and resource allocation phases.

It should be noted that the similarity between CPU and file server scheduling is partly due to the particular NAS based disk cache architecture with many (100s) file servers holding only a few TB disk space each. For SAN based architectures some more adaptation of the CPU schedulers is probably needed.

4.5. Recaller and migrator components

The recaller and migrator components control the file movements between the CASTOR disk cache and the tape archive. Since most mass storage systems contains similar components, including the current CASTOR stager, only the most important features of the new CASTOR recaller and migrator components will be listed here.

Both the recaller and migrator read the request repository to find out what files needs to be recalled or migrated. While the process of copying files to and from tape is in principle trivial, the task is complicated by the significant delays introduced by tape drive contention and the tape mounting. Tape drive contention comes from the fact that there are usually much more tape volumes than drives and the access must be queued. Tape mounting latency comes from fetching the volume and loading it onto the drive.

Because of the inherent delays introduced by the tape drive contention and tape mounting, CASTOR recaller and migrator do not select the target disk resources that will receive or deliver the data until the tape is ready and positioned. This “deferred allocation” of the disk resources has already been used with good experience in the recaller component of the current CASTOR stager: the target file server and disk that will receive the file is selected in the moment the tape mover is ready to deliver it. In this way space reservations and traffic shaping, which are cumbersome to implement and often lead to suboptimal resource utilization, can be avoided. The selection is currently based on available disk space and I/O load but other metrics/policies can easily be added.

5. First prototype

An evaluation prototype of the new CASTOR stager has been built in order to test out some of the central concepts in the new architecture. The prototype includes:

- A request handler storing all requests in a request repository (Oracle and MySQL supported).
- A master stager reading requests from the request repository and delegates the request scheduling and submission to a LSF instance.
- A LSF scheduler plug-in for selecting target file-system and pass that information to the job when it starts on the file server.
- A load monitoring running on the managed file servers collecting disk load data
- A LSF job-starter that starts an *rfiod* (disk mover) instance for each request scheduled to a file server. The RFIO API client is notified about the *rfiod* address (host and port) and a connection is established for the data transfer.

The prototype was based on standard LSF 5.1 installation without any modifications. This was possible using the plug-in interface provided by the LSF scheduler. A CASTOR plug-in is called for the different phases in the LSF scheduling to perform the matching, sorting and allocation of the file systems based on load monitoring information. While this was sufficient for the prototype, it was recognized that some extra features in the LSF interfaces would make the processing more efficient. The LSF development team was interested in providing those features and a fruitful collaboration has been established.

The prototype is also prepared for interfacing the Maui scheduler and the Maui developers have been very helpful in providing the necessary interfaces between Maui and the CASTOR file system selection component.

While the prototype was aimed to prove the functionality rather than performance, it should be mentioned that already for this simple deployment using old hardware and without any particular database tuning, the request handling was capable of storing and simultaneously retrieve

- 40 requests/second using Oracle (a dual CPU 1GHz i386 Linux PC with 1GB memory)
- 125 request/second using MySQL (single CPU 2.4GHz i386 Linux PC with 128MB memory)

6. Development status

The developments of the new CASTOR stager started in mid-2003 and a production ready version is planned for 2Q04. Before that a second prototype including all new components and the full database schema is planned for March 2004.

7. Conclusions and outlook

The CASTOR hierarchical storage management system is in production use at CERN since 2001. While the system is expected to scale well to manage the data volumes and rates required for the experiments at the Large Hadron Collider (LHC) in 2007, it has been become clear that the disk cache management will not cope with the expected request load (file opens per second). The development of a new CASTOR stager (disk cache management system) was therefore launched last year. The important features of the new system design are:

- The request processing is shielded from high peak rates by a front-end request handler, which is designed to cope with 500 – 1000 requests per second.
- The scheduling of the access to the disk cache resources is delegated to an external scheduling system. This allows leveraging work and experience from CPU batch scheduling. The first version of the new CASTOR stager will support the LSF and Maui schedulers.
- The software architecture is database centric.

A first prototype of the new system was successfully built in order to prove the new design concepts. The prototype interfaced the LSF 5.1 scheduler.

A production ready version of the new CASTOR stager is planned for 2Q04.

8. References

[1] CASTOR http://cern.ch/castor
[2] dCache http://www.dcache.org/
[3] ENSTORE, http://hppc.fnal.gov/enstore/
[4] HPSS, http://www.sdsc.edu/hpss/

[5] TSM, http://www-306.ibm.com/software/tivoli/products/storage-mgr/
[6] New stager proposal
http://cern.ch/castor/DOCUMENTATION/ARCHITECTURE/NEW/CASTOR-stager-design.htm
[7] Platform computing Inc, http://www.platform.com
[8] Maui scheduler, http://supercluster.org/maui
[9] http://cern.ch/info/Press/PressReleases/Releases2001/PR05.01EUSaward.html
[10] Andrei Maslennikov, New results from CASPUR Storage Lab. Presentation at the HEPiX conference, NIKHEF Amsterdam, 19 – 23 May 2003.
http://www.nikhef.nl/hepix/pres/maslennikov2.ppt
[11] T.Anticic et al, Challenging the challenge: handling data in the Gigabit/s range, Presentation at the CHEP03 conference, La Jolla CA, 24 – 28 March 2003
http://www.slac.stanford.edu/econf/C0303241/proc/papers/MOGT007.PDF
[12] http://cern.ch/info/Press/PressReleases/Releases2003/PR06.03EStoragetek.html
[13] SRM Interface Specification v.2.1, http://sdm.lbl.gov/srm-wg/documents.html
[14] IBM StorageTank,
http://www.almaden.ibm.com/storagesystems/file_systems/storage_tank/index.shtml
[15] Lustre, http://www.lustre.org/
[16] SUN Grid Engine, http://wwws.sun.com/software/gridware/