SLAC-PUB-14709

Grid data access on widely distributed worker nodes using Scalla and SRM

Pavel Jakl[1], **Jerome Lauret**[2], **Andrew Hanushevsky**[3], **Arie Shoshani**[4], **Alex Sim**[4] **and Junmin Gu** [4]

[1] Nuclear Physics Institute ASCR, Prague, CZ 18086, Czech Republic
[2] Brookhaven National Laboratory, Upton, NY 11973, USA
[3] Stanford Linear Accelerator Center, Menlo Park, CA 94025, USA
[4] Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

E-mail: `pjakl@bnl.gov`

Abstract. Facing the reality of storage economics, NP experiments such as RHIC/STAR have been engaged in a shift of the analysis model, and now heavily rely on using cheap disks attached to processing nodes, as such a model is extremely beneficial over expensive centralized storage. Additionally, exploiting storage aggregates with enhanced distributed computing capabilities such as dynamic space allocation (lifetime of spaces), file management on shared storages (lifetime of files, pinning file), storage policies or a uniform access to heterogeneous storage solutions is not an easy task.
The Xrootd/Scalla system allows for storage aggregation. We will present an overview of the largest deployment of Scalla (Structured Cluster Architecture for Low Latency Access) in the world spanning over 1000 CPUs co-sharing the 350 TB Storage Elements and the experience on how to make such a model work in the RHIC/STAR standard analysis framework. We will explain the key features and approach on how to make access to mass storage (HPSS) possible in such a large deployment context.
Furthermore, we will give an overview of a fully "gridified" solution using the plug-and-play features of Scalla architecture, replacing standard storage access with grid middleware SRM (Storage Resource Manager) components designed for space management and will compare the solution with the standard Scalla approach in use in STAR for the past 2 years. Integration details, future plans and status of development will be explained in the area of best transfer strategy between multiple-choice data pools and best placement with respect of load balancing and interoperability with other SRM aware tools or implementations.

1. Introduction

The amount of scientific data generated by simulations or collected from large scale experiments have reached levels that cannot be stored in the researcher's workstation or even in his/her local computer center. The amount of data is no longer measured in megabytes, but instead it scales in order from terabytes to petabytes. In the data analysis phase, the scientist typically wants to extract a subset of the data based on some criteria. Typical and well known example is Particle and Nuclear Physic data mining and analysis of detector data. The Data Acquisition System in these detectors records information about collision events between particle beams. The information is stored in multiple files, where each file contains information about thousands of such events. Typical analysis of data involves searching for rare and interesting processes

Work supported in part by US Department of Energy contract DE-AC02-76SF00515.

SLAC National Accelerator Laboratory, Menlo Park, CA 94025

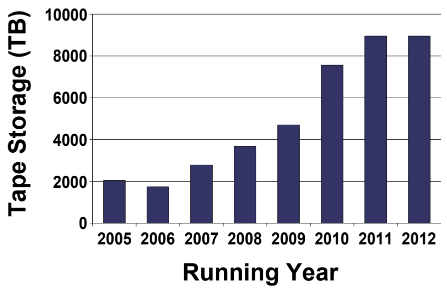


Figure 1. Raw data projection for the STAR experiment

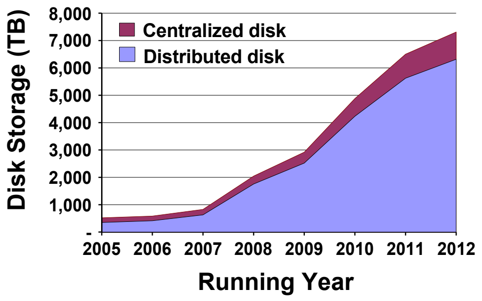


Figure 2. RHIC computing facility capacity profile

and is performed in multiple phases involving classification and summarization. In addition, the same data files may be shared simultaneously by several different groups of scientists with different interests.
A such real example is the STAR experiment, one of the four physics experiments at the Relativistic Heavy Ion Collider (RHIC) at Brookhaven National Laboratory (BNL), USA. In this kind of experiment and science, the fraction of interesting data is usually very low and in order to have a significant statistical data sample, the experiment has generated enormous amount of data since May/June 2000. The word enormous means magnitude of several Peta bytes (10^{15} of byte) of data stored in over 10 millions of files over past 6 years of collider's running. The picture 1 shows size expectations of data for up-coming years with regard to STAR physics program and planned upgrades which generate increase of data to be taken.
Any experiment facing Peta bytes scale problems is in need for a highly scalable tape storage system to keep a permanent copy of the data. In general, the tape system offers cheap and reliable storage, but on opposite side very slow access and data needs to be migrated on temporary and faster locations where could be available for physicist's analysis. Such a system responsible for managing a temporary storage needs to fulfill several requirements. In most cases, the temporary and faster location has smaller capacity than the entire tape system and therefore it needs to make decisions which files should reside, which of them evict by taking into account user's access pattern. This includes many features such as enhanced cache management containing advanced techniques such as pinning of files, lifetime for spaces and files or assigning spaces with different capabilities.
We describe a system that was developed to perform described huge data access taking advantage of two technologies used in distributed systems. The first is Scalla (**S**tructured **C**luster **A**rchitecture for **L**ow **L**atency **A**ccess) [1] designed for low latency data access in distributed environment, and the second is Storage Resource Managers (SRMs) [2] that manage the space usage of storage systems and the dynamic content of the storage.

2. Storage model design

There are two coexistent methods for connecting disk-based storage to computing nodes for fast data access. First, **centralized storage** that is a storage with many heterogeneous servers connected to one single storage space. Usually they contain one or more hard disks, often arranged into logical, redundant storage containers or RAID arrays connected over the network to computing node. Second, **distributed storage** that is a storage that has many geographically-dispersed disk drive units, usually spread over many hosts or servers. All the hosts or servers are connected together through the network. The storage devices are part of the host computer and directly attached to it. The computing nodes must therefore "*physically*"

contact the server (the host owning the storage) in order to connect to the storage device.
Although distributed storage introduce many components within a complex server/server and server/clients layout, from economical statistics, the initial purchase price is cheaper by factor of 10 comparing to the centralized storage. As a consequence, even though the implementation of centralized storage is growing at a faster rate than that of distributed storage (mainly due to the lack of ready-to-use solution to manage data distribution), its cost cannot compete to the possibility offered by distributed storage solutions. When considering distributed disk, it is important to understand what the data availability requirements are. In order for clients on the network to access the storage device, they must be able to access the server it is connected to, speaking nothing of getting information which server contact. If the server is down or experiencing problems, it will have a direct impact on user's ability to access the data. In addition, the server also bears the load of processing applications which can at the end slow the IO throughput. Considering physics data and environment, the availability and data protection aspects could be reduced by re-copying the lost data from master copy on the tape drive to the other server. Speaking about scalability and capacity of distributed storage, one could imagine linear growth of storage simultaneously with computing nodes, since the storage is attached. There is no other need for extra hardware in order to increase the size of the storage. The maintenance resources are reduced in case of distributed disk, since there is no need of having two separated persons for maintaining computing and storage element, one person can serve both of them. As a conclusion, the distributed storage seems as a better solution for physics data and is bringing cheaper, scalable, capable solution, but on the other hand worse manageability, sometimes called: "Islands of information". The difficulty relies on management of space spread among multiple servers, not mentioning load balancing issue, obtaining highest performance and scalability (since CPU and storage are now coexisting).
Driven by the need for vast amount of data and economics, the STAR software & computing project has taken the decision to move toward to a distributed storage model infrastructure as their primary storage solution as illustrated in Figure 2. This satisfies the needs of the collaboration and the requirements for the upcoming years. This is not without challenge and the next chapters will be focused on the architectural and technological aspects of the publication, management and performing large scale data access on the distributed storage co-existing with computing nodes.

3. Software challenges

In contrast to storage model challenges, one can imagine building a truly distributed network data access system with no central bottleneck or single point of failure. There are many main and basic features such a system must accomplish. One of them is ***data consistency***, where the system must provide mechanism in order to ensure that each user can see changes that others are making to their copies of data. All of this together with **single global unique name-space** which allows the name of a file to look the same on all computers. The system should also has flexible **security** module which would allow to use any security protocol (pwd, gsi etc.). The system also should give a high degree of **fault tolerance** at the user's side to minimize the number of jobs/applications failures after a transient or partial server side problem or any kind of network glitch or damaged files. In order to support incredible amount of data, **mass storage system integration** is required in order to satisfy all user's requests for different kinds of data-sets. The system should contain any **load balancing** mechanism in order to efficiently distribute the load between clusters of servers and preventing hot spots in the cluster. More importantly, when the storage element shares the same resource as computing element. One of the most important requirements of such system is the **scalability** where the performance of the system must scale with number of clients and servers and it needs to be able to support thousands of clients simultaneously with no performance decrease or even with any short random disruption of the entire system. Last, but not least it should have **Grid support** meaning an ability to connect to other instances located in different parts of the world. More precisely, the capability to share and interchange data with other storage solutions. One of the solutions partially complying with all these mentioned requirements, well known in HENP computing is Scalla (Structured Cluster Architecture for Low Latency Access) aka Xrootd [3]. The purpose of this paper is not dedicated to explanation of Scalla package architecture [1], but rather the focus was given on features which this tool lacks and also to present a solution providing them.

4. Enabling MSS access in large deployment context

At STAR, the usual analysis is performed with the help of software called SUMS (STAR Unified Meta Scheduler). Each user has to describe its intend with special language as for example: input to the task, location to store output of a task, program which will be used for processing etc. Input to the task can be specified as a list of files or using meta-data query (energy, collision etc.) to the STAR File catalog which resolves this query into particular physical data-sets. According to the user's specification, SUMS orders the data-sets, splits them into particular jobs and submits them into preferred batch system queue. As one can imagine, there must be a limitation of job's run-time to control resource sharing in such multi-user shared environment. The usual practice to limit user's job run-time is having a hard limit in clock time. In simply way, this means that job is killed when it is running too long time. This finding comes to the initial problem when a user accesses a file from the tape system within a job. Delays are highly expected and when the access and performance is not enough efficient, the jobs is sooner or later killed.

From our observation and usage at STAR, the average time to restore one file from the tape system was about $\sim$ 24 minutes. By simple counting, when a user requests 1000 files, we get the time period of 400 hours. This number is almost impossible to adjust as a hard limit. What about jobs requesting more then 1000 files ? The one of the next facts confirming very slow performance for dynamic disk population is the plot showing performance of the system *DataCarousel* [4] used for efficient management of the unique system which needs to be shared among many users requesting files from/to tape system. In respect to STAR/RHIC experiment, the tape system is represented by High Performance Storage System (HPSS).

The plot 6(a) shows performance in MB per second in relation to time range. Obviously, the performance is in average 10Mb/s upon 9 tape drives. The theoretical limit for one tape drive depends on drive technology being used (in STAR **9940b**-$\sim$ 30 MB/s, **LTO-3**-70-80 MB/sec). Most of the STAR files (around 98%) are located on 9940b tapes where 9 tape drives corresponds to the *theoretical* data throughput of 270 MB/sec. All these observations call for studying key parameters how to optimize and tune performance of the tape system.

5. Key parameters of tape system performance

There were performed several studies on how the performance of the tape system can be increased and its key parameters influencing the efficiency [5], [6]. From those studies, there are several inquiries which directly affect the performance of one tape drive, but also parameters implicitly influencing the overall aggregate performance of the system:

(i) measured maximum theoretical performance value per one drive (9940b 25-30 MB/s, LTO-3 70-80 MB/s)

(ii) the coupling to disk storage cache and the performance of it

(iii) the access pattern defined by user application and its disposal on efficient managing of multiple tape drives

(iv) the length of uninterrupted streaming of a file from/to tape and per file overhead on tape seeking, both of which are affected by the file size

The first testifies about theoretical limit of a tape performance which is not possible to overcome, for STAR case it is mostly 30MB/sec per one drive. The second consideration gives a small efficiency gain, since most raid arrays of disks have higher or comparable theoretical data throughput limit as one tape drive or more. Hence, our interest is focused on last two bullets which are influencing the performance at most and are directly connected to the usage of the system rather than a configuration of the system. The picture 3 shows two most important key parameters of tape drive performance **size of the file** and **number of files restored per one tape mount**.

5.1. File size parameter

The first parameter affects per file overhead and it is related to seeking through the tape, partly due to the time it takes to read/write two file-marks between files (4-5 seconds). This corresponds to processing about 10 000 files/day. In order to overcome this limitation, the file size should be such that the time it takes to read/write the file on tape is significantly more than the overhead (say a factor of 10). For a 30MB/s 9940B drive, this is about 1 - 1.5GB. For LTO-3 streaming at 80MB/s this corresponds to 4GB. As mentioned before, tape drives have a rate that they stream at, that is, they continue writing (reading) at this rate without starting and stopping as long as data is written (read) at this rate or faster. A drive will stop streaming to write/read a file-mark. Keeping a drive streaming provides the best rate possible and does not wear the drive as much as the continual starting and stopping when not streaming. A drive will not stream for very long when small files are written to it, or if the rate the file is being provided is slower than the drive theoretical maximum streaming rate. Files which are being used for analysis at STAR are *MicroDST files* and its average size is ∼ 88 MB. One can easily distinguish from the plot in Fig. 3 that by increasing the file size up to 1 GB, the performance efficiency gain is 40%.

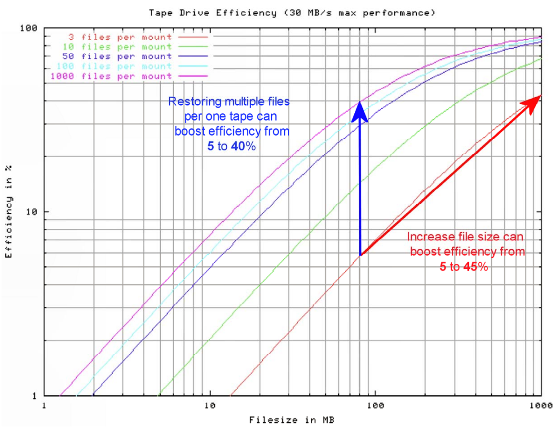


Figure 3. File size and multiple files per tape mount vs efficiency of the HPSS

5.2. Number of files per tape mount parameter

The second parameter is directly affected by the access pattern of the application which requests files from/to tape system. Here, for simplicity we consider this application as Scalla/Xrootd system. However, the access pattern is far behind the system itself, users who performs the analysis are the "*generators*" of the access pattern. Moreover, the access pattern is defined partially by user's intend of requested files, but also by Scalla/Xrootd system, since the system can have some of the files already on the disk and therefore doesn't need to access them from the tape.

By observing the plot 3, we can see that an increase of multiple files per one tape mount, we can boost the performance by 35%. However, this scenario is not feasible in real world production of huge amount of data and files spread over many tapes. The most likely number is *10* files per tape mount which corresponds to 10% gain of performance efficiency upon 88MB files. However, the growth sharply accelerates when the size increases and the performance is more than 60%.

The Scalla system uses another software tool to manage an access to HPSS, *DataCarousel*. The integration of this tool within Scalla architecture is well described in [4], but we will describe the purpose of this system in more details for further understanding. The DataCarousel (DC)

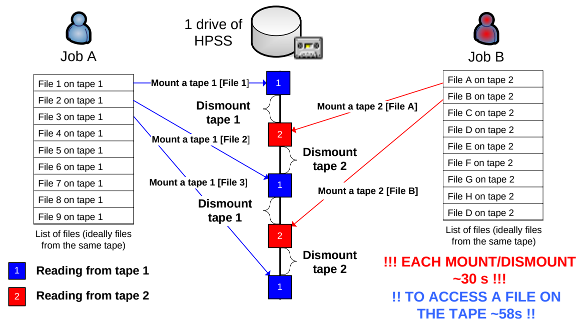


Figure 4. The hassle of two jobs for HPSS drive

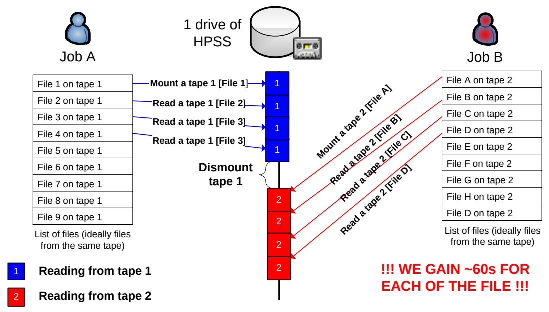


Figure 5. A pre-staging effect on HPSS drive

is an HPSS front-end which main purpose is to coordinate requests from many un-correlated client's requests. Its main assumption is that all requests are asynchronous that is, you make a request from one client and it is satisfied "later" (as soon as possible). In other words, the DC aggregates all requests from all clients (many users could be considered as separate clients) and re-order them according policies, and possibly aggregating multiple requests for the same source into one request to the mass storage. Policies may throttle the amount of data by group (quota, bandwidth percentage per user etc.), but also perform tape access optimization such as grouping requests by tape ID. Currently, the policy is based on satisfying following strategy which is the tape with the largest number of files requested. The second strategy is driven by user priorities where Scalla has the highest priority over the other users, since all files are requested within waiting jobs.
To achieve efficient performance from/to tape system using DC, one needs to submit "*enough*" long list of files to allow creating reasonable sorting and queries according to location of a file on a tape. This is obviously in clash with Scalla "*random*" access and sequential processing of user's list of files. The picture 4 shows the hassle of two jobs for one HPSS drive where the excessive mounting of the same tape is a consequence of the sequential processing.
For simplicity, lets assume just on HPSS drive and two jobs, but there are always more then 2 jobs and one HPSS drive in real production scenario. Each of the jobs has list of files to be processed where the files are likely on the same tape due to assumption that alphabetical order sorted by SUMS means files on the same tape. Likely means that there is no confidence that files have to be on the same tape, HPSS doesn't offer any attribute how to ensure that files will reach the same tape. For the next explanation, lets also assume that files on the list are presented nowhere on Scalla system and needs to be always requested from the tape system.
First **Job A** requests its first file from the list, the *"Tape 1"* is mounted for the *"File 1"* and file is read from tape to disk cache and transferred from cache to a node where job can process the file. Obviously, in the meantime where HPSS transfers the *"File 1"*, the *"Tape 1"* is dismounted to satisfy the request of the second **Job B** where the *"Tape 2"* is mounted for the *"File A"*. This situation is repeated for second files from lists and both jobs, it goes as following:

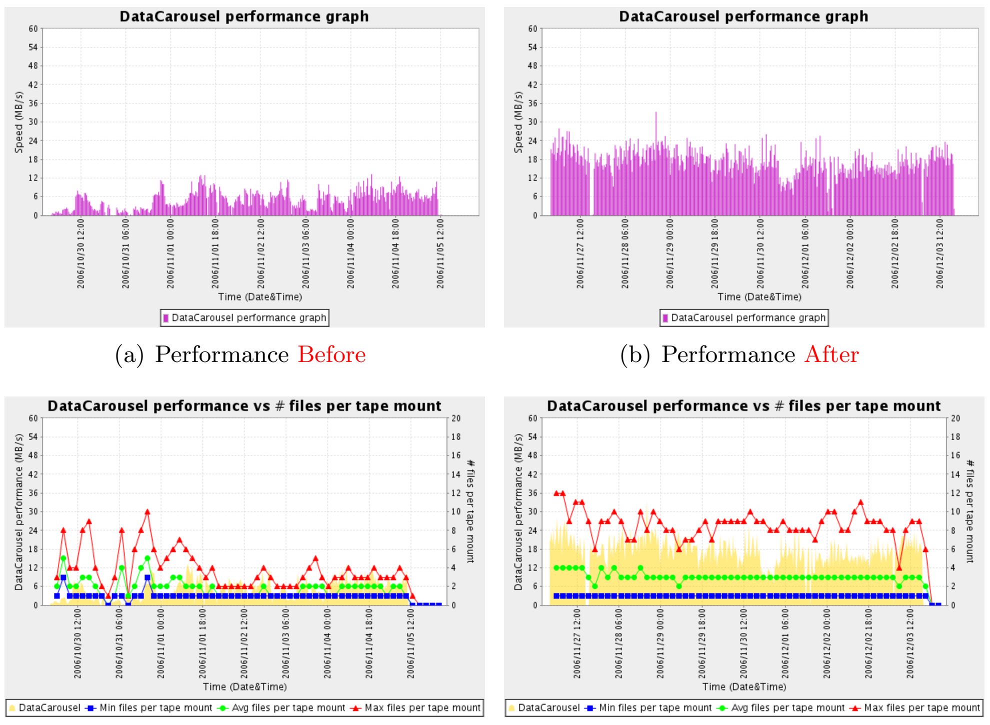
(i) mount *"Tape 1"* for *"File 2"*
(ii) dismount *"Tape 1"*
(iii) mount *"Tape 2"* for *"File B"*
(iv) dismount *"Tape 2"*
⋮
(v) mount *"Tape 1"* for *"File 9"*
(vi) dismount *"Tape 1"*

(vii) mount *"Tape 2"* for *"File D"*

(viii) dismount *"Tape 2"*

From the pattern, it is transparent that the same tape is mounted and dismounted constantly during a fixed period of time. As was mentioned before, this is counted to sequential processing where requests are arriving to DC system one after one and the optimization of tape sorting is therefore inefficient, since the list for sorting and making queries is too short. It should be also mentioned that each one mount and dismount together takes in average about ~ 60 seconds. A question here is: *"How we can increase the number of files for efficient sorting per tape in DC when using Scalla system ?"*.
Evidently, there is a solution in form of **publishing** the whole list of files to the system before starting to process them. This ensures the increment of files in DC and prevent the sequential processing defect on the tape system.For this purpose, we have implemented new feature to Scalla system called **Pre-Staging**.
The picture 5 shows the effect of the pre-staging feature on the HPSS drive. It assumes that all files are not presented on the Scalla system cache. Since the client publishes the whole list before processing, all files arrive to the system at one time and the one tape can be efficiently used to satisfy all requests from the same tape and therefore **same job**. Within this solution and example, the gain is 60 sec per each of the file from the list, in total for 9 files it is 360 sec = 6 minutes.

6. Results of pre-staging



(a) Performance Before

(b) Performance After

(c) Mounting efficiency Before

(d) Mounting efficiency After

Figure 6. The comparison of before and after using pre-staging

We have built a set of monitoring plots to monitor the efficiency of multiple files per tape and the size of the file in relation to performance of the HPSS system. The monitoring set combines values from production database of HPSS system and DataCarousel database. It gives a illustration of mentioned parameters in relation to performance and time statistic of hours, days and weeks. Following pictures 6 show time before and after releasing pre-staging. When looking on first 2 upper plots, there is visible 50% improvement on the performance. This is consequence of increase in average number of file per tape mount observed from the bottom plots.

7. Integrating Scalla/Xrootd with SRM

While the Scalla seems to perform extremely well and satisfy STAR's most immediate needs, such as a storage solution serving high-performance, scalable, fault-tolerant access to their physics data, it could itself be improved and extended.
For example, Scalla does not move files from one data-server to other data-server or even from one cache to other cache within one node, but always restore files from MSS. This may be slow and inefficient in comparison with transferring the file from other node or cache, not involving any tape mount or other delays intrinsic to MSS. Additionally, the system is not able to import files from other space management systems (as dCache, Castor [7]) or even across the grid. There are no advanced reservations of space, other users can collate the space in the meantime while the restore from MSS operation is still ongoing (in fact, we have observed failures related to the lack of space, likely related to such timing issues). There are no extended policies per users or role based giving *advanced* granting of permissions to a user. There is no concept of pinning the files, requested files can be evicted to release a space. This makes un-practical additional features such a pre-staging (essential for efficient co-scheduling of storage and computing cycles).

In addition, there are other middle-ware designed for the space management and only for the space management. Specifically, the grid middle-ware component called **Storage Resource Managers (SRMs)** [2] has for function to provide dynamic space allocation and file management on shared distributed storage systems. SRMs are designed to manage space, meaning designed to negotiate and handle the assignment of space for users and also manage lifetime of spaces. In addition of file management, they are responsible for managing files on behalf of user and provide advanced features such as pinning files in storage till they are released or also even manage lifetime of files that could be removed after specific time. SRMs also manage file sharing with configurable policies regulating what should reside on storage or what to evict. One of the powerful features of SRMs is ability of bringing the files from other SRMs, local or at remote locations including from other site and across the Grid . In fact, SRMs defines a fully specified protocol aims to handle and negotiate requests and movements. Note that SRMs themselves do not move files: they negotiate space and orchestrate file movements using standard transfer tools (gsiftp etc.) and it keeps a track of transfers and recover them from failures.
SRM has to be conceived as revolutionary protocol defining standard for communicating and sharing information over various storage systems (differential by architecture design, feature capabilities or even hardware structure etc.)
We understand the term "*SRM-aware storage solution*" as an software suite implementing particular version of SRM protocol. By this time, there are various implementations of SRM interface: **CASTOR**[7], **DPM**[8], **dCache** [9], **StoRM**[10], **BeStMan** and **DRM,TRM,HRM** - Berkeley implementation of SRM interface v1.1 [11].
The Berkeley implementations have been chosen as the implementation of SRM protocol fully satisfying requirement mentioned above in 7 and the most suitable implementation with exactly distinguish components of appropriated functionalities.
Berkeley implementation of SRM v1.1 comes with 3 types of storage resource managers: **Disk Resource Manager** (DRM) manages one or more disk resources, **Tape Resource Manager** (TRM) manages the tertiary storage system (e.g. HPSS) and **Hierarchical Resource**

Manager (HRM=TRM+DRM) stages files from tertiary storage into its disk cache. The implementation of SRM v2.2 is called **BeStMan** which is able to manage disk resources together with tertiary storage system.
On the other hand, while DRM, TRM, HRM or BeStMan do manage space efficiently and can talk to other SRM implementations (bringing for example files from other caches or SRM-aware tools), they know nothing of load balancing capabilities and they do not perform data aggregation or provide any global view of storage space, all of which was showed as a key advantage of Scalla/Xrootd system. We therefore proposed to leverage these technologies and integrate to Xrootd and SRM back-end for managing space.
Scalla/Xrootd system becomes responsible for managing the disk cluster seen on picture 7, creating and maintaining global namespace spread over several mutually independent nodes, making load balancing decisions in case of multiple replicas of a file or choosing a node in case of staging request from MSS, handling requests from client, queuing or recover them in case of failure etc.

Upon integration with SRM v1.1, the DRM responsibility relies on managing the disk cache as for example allocation of a space for newly incoming files, creating/releasing pin for a file being used/unused, making purging decisions in case of multiple possibilities and strategies etc. The HRM becomes responsible for managing the MSS such as coordinating requests to unique system, sorting requests according to multiple strategies (tape sorting, user priorities etc.), queuing requests and recover them in case of failure. The integration for SRM v2.2 follows the same picture where one node is picked up and configured with ability to manage tape resource.

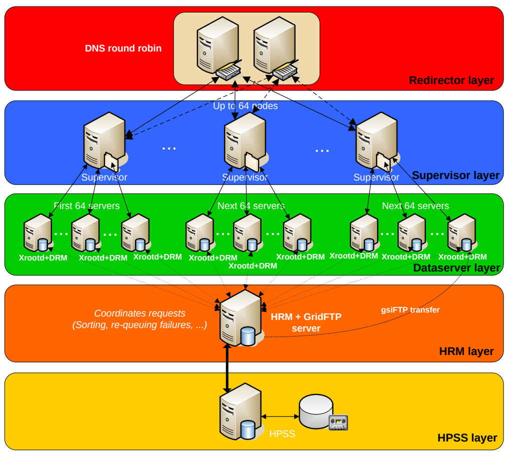
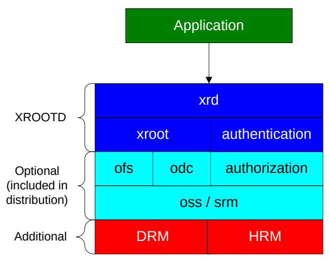


Figure 7. Global picture of Scalla and SRM v1.1 interaction

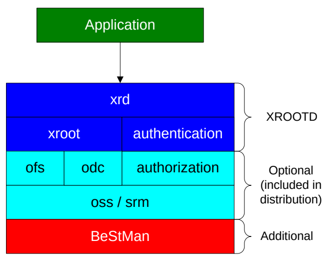
7.1. Binding systems together

Both systems have their own inner architecture and the task of integration lies on the question on how to bind them together. The Scalla system (exactly xrootd server) has its own internal structure shown in Fig. 8(a). It is composed of multiple components, each component serves a discreet task. Fortunately, this flexible layered architecture and interfaces [1] allows us to easily replace unwanted ones by another implementation incorporating SRM protocol has as it is showed in the Figure 8(b). Pictures show components architecture for both versions of SRM interface (v1.1 [12], v2.2 [13]) It is obvious from the Figure 8 that the most important component for the integration is the oss component responsible for providing access to the underlying file system. Therefore, the oss component was externalized as a plug-in offering easy overwriting the actual meta-data and data access calls (such as Open, Create, Stage, Close etc.). We have introduced new plug-in called **OssSrm** containing derived classes of the oss base classes, where methods contain direct calling of HRM or BeStMan client API methods. For instance, following examples demonstrate the logic:

- ◇ ***Create()*** - uses DRM or BeStMan to create a file (allocation, pinning the file)
- ◇ ***Stage()*** - uses DRM/HRM or BeStMan to retrieve a remote file from MSS
- ◇ ***Open()*** - informs DRM or BeStMan that the file is in use (i.e. pinning of the file)
- ◇ ***Close()*** - informs DRM or BeStMan that the file is no longer in use (i.e releasing the pin of the file)



(a) Components architecture of xrootd server with DRM/HRM implementation



(b) Xrootd server with BeStMan implementation

Figure 8. The overview of binding the Scalla and SRM

Conceptually, the xrootd server has several requirements for file management or data access calls summarized in following use cases:

- ◇ open a file already located in the cache
- ◇ open a remote file located in MSS
- ◇ create a new file and do not archive it to MSS
- ◇ create a new file and archive it to MSS
- ◇ create a new file and archive it to MSS with an specific delay
- ◇ modify an existing file located in the cache
- ◇ modify an existing file located in the cache and archive it to MSS
- ◇ modify an existing file located in the cache and archive it to MSS with the specific delay
- ◇ migrate an existing file from the cache to MSS

The reason for the need of **delayed archiving** in xrootd is that the application may want to wait till a bunch of files are generated before archiving, or it wishes to modify/append files before they are migrated to archive. In order to provide all functionalities without changing the SRM API, the "hint" parameter defined in API was used (for example: $\underline{MODIFY = true}$ or $\underline{COPY = true}$).

7.2. Status of integration
Mentioned use-cases were implemented for both versions of SRM interface. SRM v1.1 supports one physical cache partition per a node where the effort in additional development was moved into version 2.2. Scalla and SRM v2.2 is released in alpha version at the time of this paper supporting multiple cache partitions per a node. Future work will be focused on testing this solution in large scale and possibly further development to increase the stability and reliability of this solution.

8. Summary
During this paper, we gave brief overview on difficulties on managing huge amount of data and presented a software solution able to accomplish defined goals. Large amount of paper is dedicated to studying the performance of tape system with respect to this software solution. We have concluded that one of the most costly aspects of dealing with robotic tape systems is the time it takes to mount and dismount a tape. This is typically in the range of 60 - 80 seconds with current technology. Another latency problem is searching for a file on a tape, which can also take 20 - 40 seconds depending on the tape size and the search speed and also

size of the file consuming the tape. This time exponentially grows when the size of the files is close to 0. Avoiding these delays results in a large performance gain. To achieve this benefit, a scheduling system can be used that has the flexibility to stage files in tape-optimized order over many clients. The scheduling system must have information on file location on tapes (i.e., which tapes, and location on the tape). We have therefore presented such a system called "DataCarousel" having all these capabilities. However, the system does not have a global view of the files needed for an entire job. We have found a way of publishing the intend of entire job and makes the tape-optimization efficient enough.
Even if the whole system (Scalla + DataCarousel) is now able to handle the stress of requests to HPSS, it can be still improved in the way of cache management. The cache management has been proved very strongly by the grid-middleware component called "SRM" and we presented a plan on how to integrate the Scalla system with this grid middle-ware including components architecture as well as cluster architecture. Till this time, a alpha version of this exists and there still lots to finish such as large scale test and possible further development to increase the stability of this solution. We will continue on working on this front and hope to present real results soon.

9. Acknowledgment

This work was supported in part by the HENP Divisions of the Office of Science of the U.S. DOE; the U.S. NSF and by grants of GACR 202/07/0079 and LC07048.

References

[1] Hanushevsky A and Weeks B 2006 Scalla: Scalable cluster architecture for low latency access, using xrootd and olbd servers White paper URL `http://xrootd.slac.stanford.edu/papers/Scalla-Intro.htm`

[2] Shoshani A, Sim A and Gu J 2003 *Storage Resource Managers: Essential Components for the Grid* (Kluwer Academic Publishers) chap In Grid Resource Management: State of the Art and Future Trends, pp 321–340

[3] Dorigo A, Elmer P, Furano F and Hanushevsky A 2005 *Proc. WSEAS'05*

[4] PJakl, JLauret, Hanushevsky A, Shoshani A and Sim A 2006 *Proc. of Computing in High energy and nucler physics (CHEP'06)*

[5] Panzer-Steindl B 2005 Some calculations for the sizing of tape storage performance Tech. rep. CERN-IT

[6] FNAL Performance to/from tape http://www-isd.fnal.gov/UserPerformanceGuidlines.html

[7] Bcarring O, Couturier B, Durand J D, Knezo E and Ponce S 2004 *In Proceedings of the 12th NASA Goddard* vol 21st IEEE Conference on Mass Storage Systems and Technologies pp 345–359

[8] Stewart G A, Cameron D, Cowan G A and McCance G 2007 *Australasian Symposium on Grid Computing and Research* vol 68 (Ballarat, Australia)

[9] Fuhrmann P 2004 dCache, the overview White paper URL `http://www.dcache.org`

[10] Corso E, Cozzini S, Donno F, AGhiselli, LMagnonii, MMazzucato, Murri R, PPRicci, Terpin A, VVagnoni, RZappi and HStockinger 2006 *Proc. of Computing in High energy and nucler physics (CHEP'06)* (Mumbai,India)

[11] Sim A, Gu J, Shoshani A and Natarajan V 2004 *SSDBM* (IEEE Computer Society) pp 403– ISBN 0-7695-2146-0

[12] Bird I, Hess B, Kowalski A, Petravick D, Wellner R, Sim A and Shoshani A 2001 Common storage resource manager operations SRM collaboration internal documents SRM specification version 1.0

[13] Perelmutov T, Petravick D, Corso E, Magnoni L, Gu J, Badino P, Barring O, Baud J P, Donno F, Litmaath M, Witt S D, Jensen J, Haddox-Schatz M, Hess B, Kowalski A and Watson C 2007 The storage resource manager interface specification (version 2.2) SRM design version 2.2