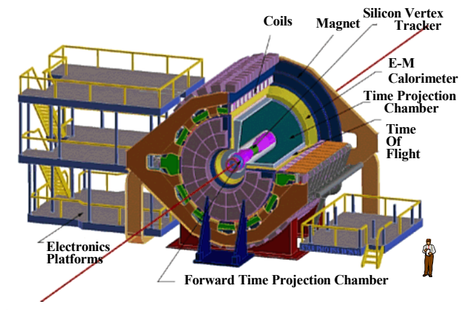
Grid data storage on widely distributed worker nodes using Scalla and SRM

Pavel Jakl for the STAR collaboration

Introduction

Facing the reality of storage economics, High Energy and Nuclear Physics (HENP) 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 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 explain the key features and approach on how to make access to mass storage (HPSS) possible in such a large deployment context. Furthermore, we give an overview of a fully "gridified" solution using the plug-and-play features of Scalla architecture, replacing standard storage access with grid middle-ware SRM (Storage resource manager) components designed for the space management.



STAR detector

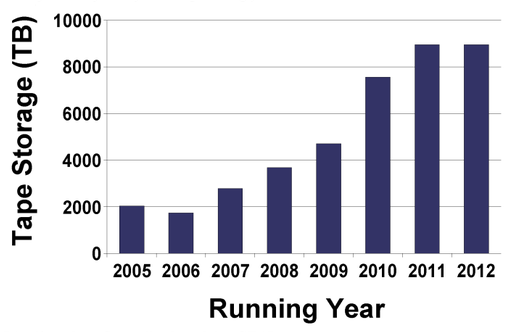
- Produce 1 PB of data per year and grows each year
- Already 15 PBs stored on tapes from previous years
- Over 15 Millions of files and grows

Distributed data management at STAR

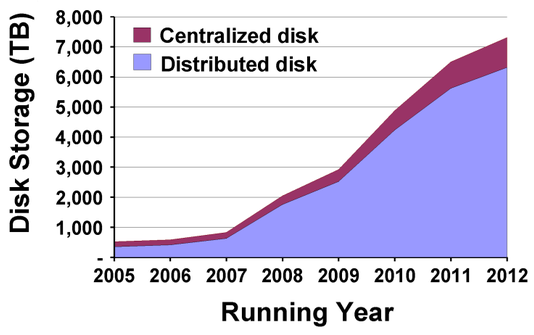
Several High Energy and Nuclear Physics experiments such as Solenodial Tracker at Relativistic Heavy Ion Collider (STAR RHIC) at Brookhaven National Laboratory produce PetaByte of data (raw and reconstructed) per year which bears deep puzzle to manage data over the normal data size storage in today's personal environment.

This challenge could in principle be resolved by using solutions involving standard centralized storage managed by NFS or instead using cheap disks attached to processing nodes called distributed storage. 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.
When considering distributed disk, the scalability and capacity linearly grows simultaneously with computing nodes, since the storage is attached. There is also 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.
On the other hand it brings 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 decided to move toward to a distributed storage model infrastructure as their primary storage solution.

Projection of data for the upcoming years



Distributed vs. Centralized model at STAR

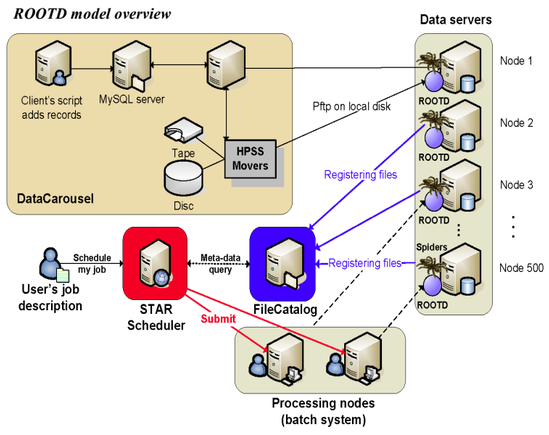


ROOTD distributed model

To overcome some of the mentioned limitations, STAR has used ROOTD [4] based model which provides a remote file access mechanism via TCP/IP-based data server daemon within the ROOT framework.

Any experiment facing Peta bytes scale problems are in need for a highly scalable hierarchical storage system to keep a permanent copy of the data. STAR uses a High Performance Storage System called ***HPSS***. Having a large archive is not sufficient of course as million of files would make the recovery of one file a needle in a hay stack nightmare. The second vital component is to arm the experiment with a robust and scalable catalog (***FileCatalog***), keeping the millions of files and potentially, an order of magnitude higher number of file replicas at reach (i.e where the data are located). The environment is composed of a large set of nodes with each node having from one to 3 local drives

ROOTD model overview



Since the data always has a primary copy deposited by the data reconstruction process into HPSS. Additional tools are needed to retrieve and populate the distributed disks in a pre-staged and static manner. To deal with this effort, the ***DataCarousel*** system was developed. All user data-intensive batch jobs read a file remotely via ROOTD, their jobs themselves are submitted according to the selection of data-sets. The ***STAR*** Unified Meta-***Scheduler*** (SUMS) would resolve user's meta-data-sets into logical files and identify particular physical locations of a file using the FileCatalog API. This abstraction layer makes the model viable as all files in this model would otherwise be strongly associated to server and storage that is, requires exact physical location knowledge which a user would hardly be able to keep track of the data-sets and their dynamic..

Real production scenario: optimizing access to the tape system

From our observation and usage at STAR, the average time to restore one file from the tape system was about ~ 21 minutes. By simple counting, when a user requests 1000 files, we get the time period of 350 hours being beyond any acceptable limit.

The first picture shows two most important key parameters of tape drive performance:

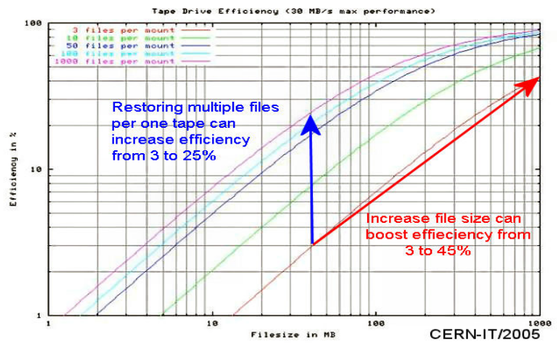
- the size of the file
- the number of files restored per one tape mount

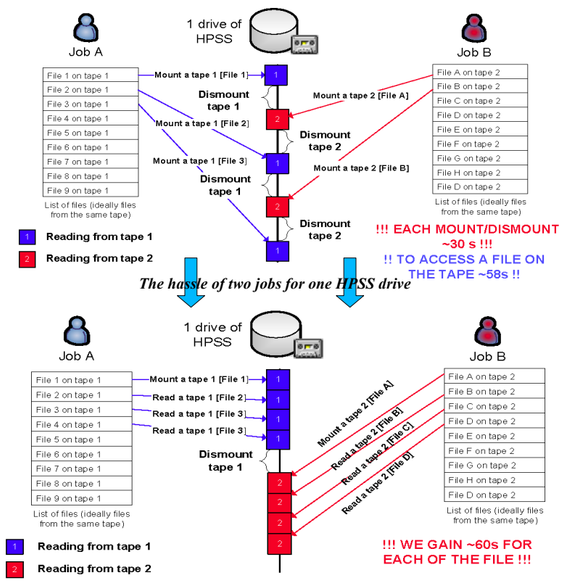
Files which are being used for analysis are MicroDST files with average size of ~88 MB. One can easily distinguish from the plot that by increasing the file size up to 1 GB, the performance efficiency gain is 40%. While increasing file size of already reconstructed and produced data is not an easy procedure and cannot be improved by any magic procedure within the Scalla/Xrootd system, the more interesting parameter is the second one.

The second parameter is directly affected by the access pattern of the application which requests files from/to tape system. However, the access pattern is far behind the application 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 above, 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 boost is more than 60%.

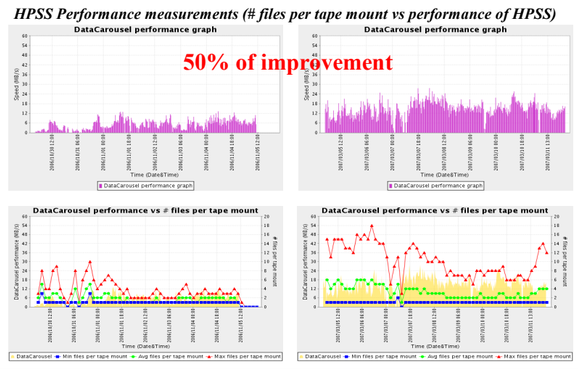
The two pictures shows the hassle of two jobs for one HPSS drive where the excessive mounting of the same tape is a consequence of the sequential processing, while the second one shows ideal state.
First *Job A* requests its first file from the list, the "*Tape 1*" is mounted for the "*File 1*" and file 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.

From the pattern, it is transparent that the same tape is mounted and dismounted constantly during a fixed period of time. Evidently, there is a solution of publishing the whole list of files to the system before starting to process them. This ensures the increment of files for efficient tape sorting and prevent the sequential processing defect on the tape system.. For this purpose, we have implemented new feature to Scalla system called **Pre-Staging.**





HPSS Performance measurements (# files per tape mount vs performance of HPSS)



Scalla/Xrootd as a next generation tool for distributed data access

But while it seems that ROOTD model [1] can achieve sophistication and faultless features at a first glance, the system still has its major flaws and deficiencies. The biggest is the lack of dynamic features as files are added and removed. ROOTD being by essence *Physical File Name (PFN)* oriented, it first needs constant cataloging and therefore the system lacks the flexibility of moving the data around without special handling.

Even though the files would be distributed at multiple places, physical file access requires exact reference at submission: by the time the job really starts, the entire load picture of the cluster may very well be different from what was used for the file access decision making process. Files placed on overloaded and not responding nodes could suddenly be requested and the scheduled job would die. This is inherent to the latency between a job dispatching and the time the work unit to really starts, this cannot be circumvented within a PFN model. In fact, another of those problems comes when a node suddenly re-appears but the disk holding the data was wiped-clean (maintenance downtime due to disk failure and replacement). In such cases, the registering of files does not only have little time to update its information but may not even exists since the system disk was wiped out. More obvious, the data population is relatively static: users could access only the data-sets already pre-populated in the system but never have a chance to access data-sets available on the mass storage. A dynamic system must therefore have the capability to hand shake with mass storage systems.

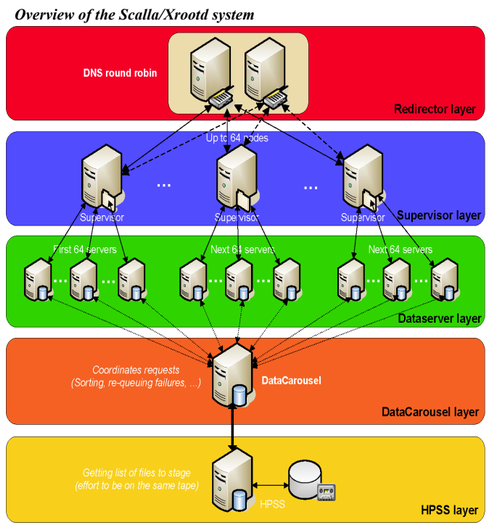
Also, such system should be self-adaptive, relying on its own coordination mechanism to balance load and access rather than relying on an external component providing mapping from meta-data or logical to physical name space.

The main and basic features such a system must accomplish are:

- ***Scalability:*** the performance of the system must scale with the number of clients and servers
- ***Fault tolerance:*** a high degree of fault tolerance at the user side is mandatory to minimize the number of jobs/applications failure after a transient or partial server side problem or any kind of network glitch or damaged files
- ***Security:*** allowing to run any security protocol
- ***Load balancing:*** a load balancing mechanism is needed, in order to efficiently distribute the load between clusters of servers and preventing hot spots in cluster
- ***Reliability:*** eliminate the single point of failure
- ***Replica management:*** determination of the location and multiplicity of data
- ***Single global unique name-space***: One mechanism that allows the name of a file to look the same on all computers is called a uniform name space
- **MSS integration:** accessing files from permanent storage (such as HPSS)
- ***Grid integration****:* the ability to connect to other instances located in different parts of the world

All requirements listed at the beginning of this section and additional requirement of external cataloging complies to the ***Scalla/Xrootd system***. Its architecture allows the construction of single server data access sites up to load balanced environments and structured peer-to-peer deployments, in which many servers cooperate to give an exported uniform name-space.

Overview of the Scalla/Xrootd system



Comparison of old and new solution

	ROOTD	Scalla
Scalability	Yes	Yes
Fault tolerance	No	Yes
Security	Yes	Yes
Load balancing	No	Yes
Reliability	No	Yes
Replica management	No	Yes
Unique namespace	Yes	Yes
MSS integration	No	Yes
Grid integration	No	Not yet

Grid access for the Scalla system using 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 import files from other space management systems (dCache, Castor) 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. 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) 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 .

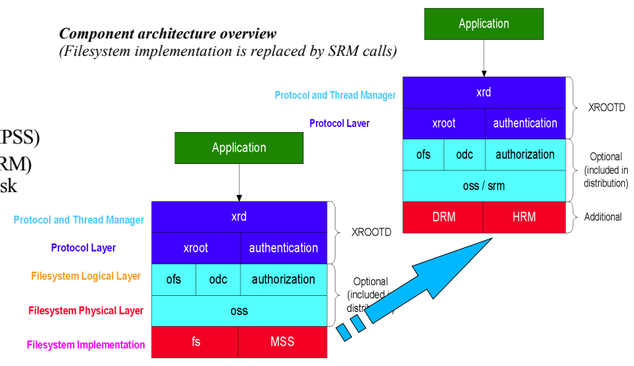
SRM comes in three flavors of storage resource managers:

- ***Disk Resource Manager (DRM)***
  - manages one or more disk resources
- ***Tape Resource Manager (TRM)***
  - manages the tertiary storage system (e.g. HPSS)
- ***Hierarchical Resource Manager*** (HRM=TRM+DRM)
  - stages files from tertiary storage into its disk cache and manage both resources

Scalla – SRM interaction:

- *Scalla* is responsible for managing the disk cluster (aggregation, load balancing etc.)
- *DRM* is responsible for managing the disk cache
- *HRM* is responsible for managing access to HPSS

Component architecture overview
(Filesystem implementation is replaced by SRM calls)



Summary

- Scalla/Xrootd is deployed on almost 500 nodes serving over 350 TBs of disk space [1]
- Load balancing and handshake with tape system make the system resilient to failures and it is used for **daily analysis**
- we developed monitoring toolkit to measure HPSS errors and ratio of HPSS requests over all requests as well as performance of HPSS with respect to 2 key performance parameters described in [2]
- we gathered and developed several performance measurements where the results are counted to applied load balancing optimizations well described in [2]
- Integration of Scalla/Xrootd with SRM is still ongoing where *ISSGC'07 attendance* is giving me wonderful opportunity to learn about several grid middle-ware components necessary for this exercise

References

[1] *P. Jakl, J. Lauret, A. Hanushevsky, A. Shoshani, A. Sim: From rootd to xrootd Prooceedings of CHEP'06, India, 2006*

[2] *P.Jakl et. al. : Managing widely distributed data-sets Research report, FNSPE CTU, Czech Republic, 2006*

[3] *P. Jakl et.al : Data access on widely distributed worker nodes ROOT workshop, CERN, 2007*

[4] *ROOT framework, http://root.cern.ch*

[5] *STAR, http://www.star.bnl.gov*