FROM ROOTD TO XROOTD, FROM PHYSICAL TO LOGICAL FILES: EXPERIENCE ON ACCESSING AND MANAGING DISTRIBUTED DATA

P. Jakl, Nuclear Physics Institute, Prague, Czech Republic
J. Lauret, Brookhaven National Laboratory, Upton, NY 11973, USA
A. Hanushevsky, Stanford Linear Accelerator Center, Menlo Park, CA 94025, USA
A. Shoshani, A. Sim, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

Abstract

With its increasing data samples, the RHIC/STAR experiment has faced a challenging data management dilemma: solutions using cheap disks attached to processing nodes have rapidly become economically beneficial over standard centralized storage. While more difficult to manage, the STAR experiment moved to a storage attached to processing nodes that is, a widely distributed data model rendered viable by the introduction of a scalable replica catalog, and a home-brewed data replication and data management system. Access to the data was then provided via the rootd TNetfile API-based integrated in its framework.
However, the system has a few flaws and STAR has moved to an Xrootd based infrastructure, a tool initially used in the Babar experiment for its data management. We will report in this paper our model and configuration, explain the previous and current approach and the reasons for migration from one to the other as well as presenting our experience in deploying and testing Xrootd.
We will introduce our plan toward a full grid solution and the future incarnation of our approach: the merging of two technologies and the best of two worlds, Xrootd and SRM [4], [5], [6]. This will enable the dynamic management of disk storage at the xrootd nodes, as well as provide transparent access to remote storage nodes, including Mass Storage Systems.

INTRODUCTION

Driven by increasingly complex problems and propelled by increasingly powerful technology, today's science is as much based on computation, data analysis, and collaboration as on the efforts of individual experimentalists and theorists. But even as computer power, data storage, and communication continue to improve exponentially, computational and storage resources are failing to keep up with what scientists demand of them.
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 (see Fig. 1) which bears deep puzzle to manage data over the normal data size storage in today's personal environment.

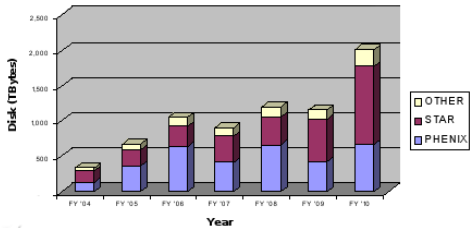

Figure 1: Raw data projection for the RHIC experiment. These numbers need to be multiplied by a factor of 2 for STAR to account for one pass of reconstructed data.

DISTRIBUTED VS CENTRALIZED DATA MANAGEMENT

This challenge could in principle be resolved by using solutions involving standard centralized storage managed by NFS [15], PanFS [16], Lustre [17], GPFS [18] or instead, using cheap disks attached to processing nodes which represents a greater economical benefit. While the two approaches seem a-priori radically different, they nevertheless share a common problem which is the ability to manage storage distributed among multiple servers or components (blade for PanFS, etc ...) but distributed storage pauses numerous additional challenges one would need to take into account. At the end, economic leaded STAR to choose a model where cheap and widely distributed storage will prevail over centralized solutions as illustrated in Fig. 2). In a centrally distributed data model, a naive ap-

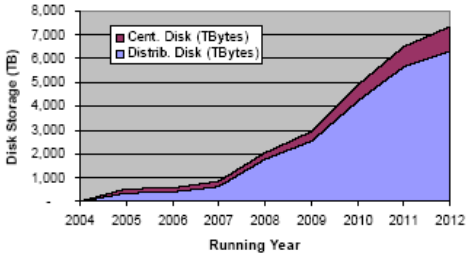

Figure 2: RHIC Tier-0 capacity profile

proach of using NFS-like solution is limited by the ability of the infrastructure and software to efficiently balance the load amongst the data servers (a few) available to the users or the data manager. However, a deeper requirements anal-

ysis shows that this kind of solution is not acceptable for many well-known reasons. Thousands of concurrent accesses from end users batch jobs that continuously analyze the data in a completely random way would for example greatly overcome the scalability of the basic NFS architecture. To achieve load balancing in such environment, one could imagine spreading the dataset randomly on all available storage but this would imply a complex organization of the data, a possibly dynamic re-organization of the data sets as demands changes in addition of an accurate file catalogue (and cataloguing capabilities). But even though one reverts to such techniques, NFS would still expose large pools of disk subject to equally massive data losses or corruption on crashes or hardware failures: most jobs are not much tolerant to such events and if a file simply do not exists, it is likely with every common framework to see the job dies without any possibility to search for other "copy" of a file. Solutions such as GPFS or PanFS resolve some of those load balancing issues by pushing down the handling of the load balancing and data replication to a lower hardware or driver level, presenting to the user a seamless NFS "look-and-feel" to the user without the burden of a learning curve. Perhaps encouraging solutions, costs remain extremely prohibitive.

To overcome some of these limitations, the natural alternative is building a "mesh" of loosely coupled data servers. Within this model, one would naturally fragment the storage to multiple smaller ones, the loss of one data server would not make a dramatic over impact providing one could still detect and recover from failures and dynamically re-populate lost data which otherwise would starve the data sets availability. From a server component aspect, ROOTD [12] offers the solution to share this load between many machines by keeping the files on the farm local disks. A ROOT client data access method could provide a way to reach the remote data transparently to the users of the framework. We will now discuss the engineering of such model and introduce its limitations.

ROOTD DISTRIBUTED DATA MODEL

Any experiments facing Petabytes scale problems are in need for a highly scalable hierarchical storage system to keep a permanent copy of the data off disks. The media are usually tape as to date; it has proven to be cheaper than any other solutions. STAR and all RHIC experiments use a High Performance Storage System called HPSS [14]. 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 catalogue, keeping the Millions of files and potentially, an order of magnitude higher number of replicas at reach. Here, we will not discuss the need or the differences between file, replica and meta-data catalogues but will assume that from a user perspective, queries to a meta-data catalogue should resolve into datasets, physical or logical.

To this aim, STAR had developed a scalable and reliable File catalog [13] that not only holds information about physical location of a file but also meta-data information of the file such as numbers of events, triggers etc. being used by its users on a daily basis to identify data sets.

Illustrated in Fig. 3, the distributed data STAR environment is a large set of nodes (320) with each node having from one to 3 local drives. 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. To deal with this effort, the DataCarousel [13] system was developed. Its main purpose is to organize the requests made by users as well as by data population requests made on behalf of the home-grown data management tool. Such cohesive data access and request throttling is handled by a set of compiled code interfacing with HPSS API and scripts implementing policies managing the entire system of requests therefore, preventing chaos. In this system, requests are stored in a MySQL database and a decision making, taking into account both usage policies and file relative location on one particular tape are taken into account to therefore restrict excessive and thus much slower restore due to mounting of robotic tapes. All requests to retrieve a file are handled asynchronously.

However, the choice of where to restore the data from all available nodes is not a decision made by the DataCarousel itself its purpose only being to coordinate restore requests from HPSS. To achieve this, an additional set of scripts relies on space monitoring information to determine the free storage and chose amongst the many node solely based on space availability criterion coupled with the minimal set of files per storage one would need for a viable operation. Additionally, Spiders [13] keeps track of newly appearing files on each storage element and add them to the replica catalog. The Spiders also detects data which disappear, the total time to update the entire catalog being of the order of minutes regardless of the number of nodes. Eventually,

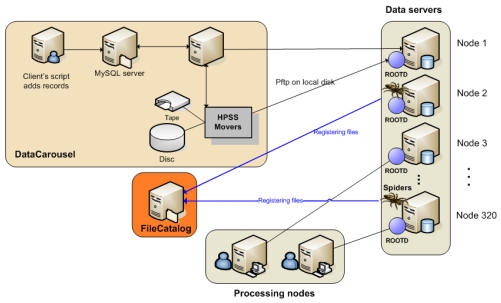

Figure 3: ROOTD distributed data model used in STAR experiment

and to complete the illustrative model of Fig. 3, all user data-intensive batch jobs read a file remotely via ROOTD, their jobs themselves are submitted according to the selection of data sets, a Meta-Scheduler front end would resolve from the STAR FileCatalog. 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. A user would hardly be able to keep track of the data-sets and their dynamic. The system has been extremely scalable when it comes to increasing the number of data servers, including its back-end catalogue and its relative accuracy over a ten million replicas size problem.
But while sophistication and faultless features could be achieved at a first glance, the system still has its major flaws and deficiencies. The biggest is the lack of dynamic features. ROOTD being by essence Physical File Name (PFN) oriented, it first needs constant cataloguing 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. Overloaded and not responding nodes could suddenly be requested and the scheduled job dies. This is inherent to the latency between a job dispatching and the work unit to really start. 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 Spider do not only have little time to update its information but the jobs was furthermore already scheduled with the previous knowledge of file present on that storage; this would be fatal to a job. 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 only. A dynamic system must therefore have the capability to hand shake with mass storage systems. Finally, a more subtle consideration, all users in rootd are trusted and no authorization mechanism exists in this system as there lacks write access and advanced authorization layers. 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.

XROOTD AS NEXT GENERATION TOOL FOR DISTRIBUTED DATA ACCESS

All mentioned issues led us to search for new generation tool for managing distributed data. Some of the needed characteristics are explained in references [1], [2]:

- Multiple servers have to cooperate with the purpose of handling huge amounts of distributed (and redundant if necessary) data without forcing the client to know which server to contact to access a particular file.
- One would benefit from having the server hide from the client applications its underlying file system types, including mass storage. In other words, access to any files in a given namespace should be transparently achieved regardless of the storage.
- In order to efficiently distribute the load between clusters of servers, a load balancing mechanism is needed.
- The system resources (sockets, memory, cache, disk accesses, CPU cycles, etc...) have to be used at the best, at both client and server sides.
- A high degree of fault tolerance at the client side is mandatory to minimize the number of jobs/applications which have to be restarted after a transient or partial server side problem or any kind of network glitch or damaged files

All of these requirements complies the eXtended rootd system also known as xrootd [1], [2], [3]. Its structure 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 namespace. If we compare side to side rootd and how xrootd system can solve our problems with its architecture and features, we arrive at the following conclusions. First, ROOTD knows only about PFN forcing a linear scaling of the catalogue as the number of data servers increase: XROOTD on the contrary knows about LFN, and refer to LFN for any data located within the xrootd system; there is no need for external additional cataloging procedure or Spiders. XROOTD load balancing mechanism determines which server is the best for client's request to open a file; nodes are selected based on reported information such as load, network I/O, memory usage and available space ensuring that the scenario of an on-responsive node would never occur. XROOTD additionally has fault tolerance features and load could be taken by other data-servers holding the data shall one data server be offline. But additionally, XROOTD implements a plugins to interact with mass storages. Missing data can be again restored from MSS within the user's job and the job startup latency is no longer an issue. If the file is not present by the time the job starts, it will be imported into the xrootd system space again. Within the same feature, one could imagine a completely dynamic data space population, no longer relying on pre-staged data but on a mechanism of "data on demand". As users request new data, it appears in the system unlike our rootd based system where data sets have to be judiciously chosen before hand. Finally, XROOTD has a plethora of Authorization plug-in which resolves the "trusted/untrusted" write access issue hinted earlier and open avenues to a finer access mechanism granularity.
On Fig. 4 we show a quick overview of the xrootd architecture [1] and how user's requests are handled within the xrootd cluster. Each request of a file is steered on node called Redirector which broadcasts messages to other nodes with a simple question "Do you have the file XX?" The whole structure consists of other layers allowing building a B64 tree structure and therefore used extremely fast 64-bits logical operations. When the question is answered,

the client is redirected to the particular node holding the data. This redirection prevents from overloading the head of the structure and also allows overspreading I/O load across the farm. The file is scheduled to be staged from HPSS when no node has a particular file or in case that all servers which serve the file are overloaded.

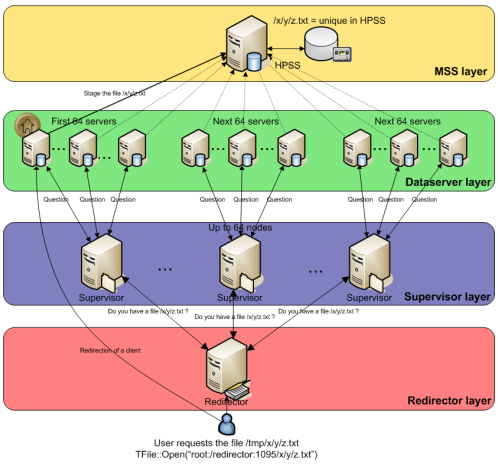

Figure 4: XROOTD architecture

XROOTD deployment/contribution

During the initial deployment, we had to contend with a few short comings such an initial limitation to 64 nodes per re-director. We reported this issue in February 2005 and a solution was available in April/May 2005. Within the activation of the authentication plugin we bumped on shaky, not well tested and documented part of xrootd which we contributed by providing bug fixes and possibility to authenticate as other user. One of the next contributions to the xrootd team was bug fixes of script measuring the load of servers: this script used pipe listing to measure actual I/O of a node. This worked well on Solaris platform but would hang on Linux platform due to a slightly different behavior of the netstat command. In addition, the computation of the IO was corrected. Finally, and to achive smooth transition from rootd(PFN) to xrootd (LFN only) we had to realize changes in the xrootd code allowing for support of both PFN and LFN. Further discussion lead to the design of a generic interface for the conversion of PFN ro LFN and vice-versa.
In our testing of the MSS plugin, we further observed that many data-serevrs may request data simultaneously, therefore falling into the uncoordinated request trap. This was exacerbated by the use of xrootd in such a large cluster and by many users: the net effect and event led to an HPSS collapse and downtime. While our plan is to go toward an SRM layer underneath as we will discuss in the next section, we decided in the interim to adapt our system by retrieving files from MSS using the DataCarousel framework.

XROOTD-SRM INTEGRATION

While the xrootd seems to satisfy our most immediate needs, it could itself be improved and extended. For example, Xrootd does not move files from data-servers to data-servers or cache to cache but restore files fro MSS. This may be slow and inefficient. Additionally, the system is not able import files from other space management systems or even across the grid (dCache, Castor ...). In a large scale pool of nodes, if ALL clients ask for a file restore from MSS, the system would exhibit a lack of coordination of accesses MSS resources as it lacks a request"queue". This advanced feature is needed for any coordinated requests and is especially important in a shared access environment where other tools, such as bulk data transfers to remote sites, may also perform MSS staging requests. No advanced reservation exists, no extended policies per users or role based giving advanced granting of permissions to a user.
In addition, there are other middle-ware designed for space management and only space management. Specifically, the grid middle-ware component called Storage Resource Managers (SRMs) [4], [5], [6] 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 remote locations including from other site.
SRMs have 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. On the other hand, while SRMs do manage space efficiently and can talk to other SRM (bringing for example files from other caches or SRM-aware tools), they are missing load balancing capabilities, a global data aggregator or a global view of storage space all of which was significantly showed as a great advantage of XROOTD. We therefore propose to leverage these technologies and integrate to Xrootd and SRM backend for managing space.

XROOTD-SRM components architecture

Both systems have their own inner architecture and the task of integration lies on the question on how to bind them together. Fortunately, Xrootd with its given layered architecture [1] allows us to re-use main elements and replace unwanted ones by SRM components has it is showed in

Fig. 5. Xrootd will remain responsible for managing disk cluster and the access to the global namespace, DRM will be accountable for managing disk cache and HRM will be responsible for staging files from MSS. In other words xrootd becomes a client of SRMs. While a functional implementation is not yet available at the time of this paper, basic class interface designed were integrated to the Xrootd to provide further hooks for this work to complete.

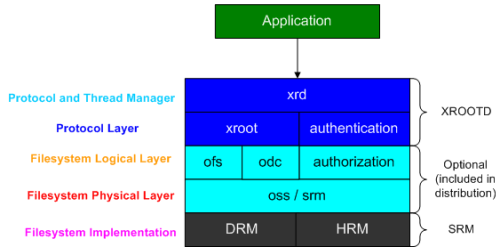

Figure 5: XROOTD-SRM components interaction

SUMMARY

We deployed xrootd system on 320 nodes at STAR Tier-0 center making this functional deployment the biggest xrootd deployment in the world. In addition to stability of the system, modulo few fixes in year 2005, the system appears to be stable and easily configurable [7], [8], [9], [10], [11]. From restart of data-servers "underneath" a running process to control file losses, we tested several fault-tolerant features which all performed beyond expectations (the clients even recovers from a global restart of all data-servers). The monitoring of XROOTD behavior by Ganglia toolkit [19] system in large scale and over long period of time haven't shown no significant impact on CPU or memory usage on nodes making the integration of such system even more palatable to reluctant system administrators, concerned of general impact a new system may introduce. Additionally, the flexible architecture of the Xrootd system appears to be suitable for a fusion of this middleware with SRM technologies which in turn, will bring to Xrootd Grid-aware capabilities.

ACKNOWLEDGEMENTS

The main author would like to express his thanks to Dr. Jérôme Lauret for leading him throughout this project and for providing advices and would to also like to thank Andy Hanushevsky, Arie Shoshani and Alex Sim for inspiring meetings and help to accomplish our goals.
This work was supported in part by the HENP Divisions of the Office of Science of the U.S. DOE; the U.S. NSF; IRP and GA of the Czech Republic.

REFERENCES

[1] A. Hanushevsky, A. Dorigo, F. Furano: The Next Generation Root File Server, Proc. CHEP2004, 2004

[2] A. Dorigo, P. Elmer, F. Furano, A. Hanushevsky: XROOTD - A highly scalable architecture for data access, Proc. WSEAS2005, 2005

[3] A. Hanushevsky, H. Stockinger: Proxy Service for the xrootd Data Server, Proc. SAG2004,

[4] A. Shoshani, A. Sim, J. Gu: Storage Resource Managers: Essential Components for the Grid, In Grid Resource Management: State of the Art and Future Trends, pp. 321-340, Kluwer Academic Publishers, 2003.

[5] L. Bernardo, A. Shoshani, A. Sim, H. Nordberg: "Access Coordination of Tertiary Storage for High energy Physics Applications," IEEE Symposium on Mass Storage Systems 2000, pp. 105-118, 2000

[6] A. Shoshani, A. Sim, J. Gu: Storage Resource Managers: Middleware Components for Grid Storage, IEEE Symposium on Mass Storage Systems, 2002 (MSS '02)

[7] A. Hanushevsky: XRD Configuration Reference, SLAC, 2005, http://xrootd.slac.stanford.edu/

[8] A. Hanushevsky: Open File System & Open Storage System Configuration Reference, SLAC, 2005, http://xrootd.slac.stanford.edu/

[9] A. Hanushevsky: Open Load Balancing Configuration Reference, SLAC, 2005, http://xrootd.slac.stanford.edu/

[10] A. Hanushevsky: GANIS, G.: Authentication & Access Control Configuration Reference, SLAC, 2005, http://xrootd.slac.stanford.edu/

[11] A. Hanushevsky: Cache File System Support MPS Reference, SLAC, 2004, http://xrootd.slac.stanford.edu/

[12] ROOTD, http://root.cern.ch/root/NetFile.html

[13] STAR Computing, http://www.star.bnl.gov/STAR/comp/

[14] High Performance Storage System, http://www.hpss-collaboration.org/hpss/index.jsp

[15] Network File System (NFS), http://www.tldp.org/LDP/nag/node140.html

[16] Panasas File System (PanFS), http://www.panasas.com/panfs.html

[17] Lustre cluster file system, http://www.lustre.org/documentation.html

[18] General Parallel FileSystem (GPFS), http://www-03.ibm.com/servers/eserver/clusters/software/gpfs.html

[19] Ganglia toolkit, http://ganglia.sourceforge.net/