# Data Access in HEP

**Fabrizio Furano**[*][†]
*CERN*
*E-mail:* `furano@cern.ch`

In this paper we pragmatically address some general aspects about massive data access in the HEP environment, starting to focus on the relationships that lie among the characteristics of the available technologies and the data access strategies which are consequently possible. The upcoming evolutions in the computing performance available also at the personal level will likely pose new challenges for the systems that have to feed the computations with data. We will introduce some ideas that may constitute the next steps in the evolution of this kind of systems, towards new levels of performance, interoperability and robustness. Efficiently running data-intensive applications can be very challenging in a single site, depending on the scale of the computations; running them in a worldwide distributed environment with chaotic user-related random access patterns needs a design which avoids all the pitfalls which could harm its efficiency at a major degree.

*13th International Workshop on Advanced Computing and Analysis Techniques in Physics Research*
*February 22-27, 2010*
*Jaipur, India*

[*]Speaker.
[†]Thanks to A.Hanushevsky and F.Carminati for their invaluable support.

## 1. INTRODUCTION

In this work we address the various solutions we developed in order to initiate a smooth transition from a high performance storage model composed by several nodes in different distant sites to a model where the nodes cooperate to give a unique file system-like coherent view of their content.

With the term "node" we refer to the deployment of a storage site of any size, from a small set of disks up to the most complex hierarchical storage [19] installation.

The task of promoting the maximum level of performance while making a potentially high number of storage nodes collaborate, is very challenging to achieve. The goal is to be able to effectively deploy a production-quality system, where the functionalities are effectively usable, and the data access performance is as close as possible to the one reachable by the hardware used to deploy this system. This can be considered as the major challenge in such systems, because requirements and expectations about performance and robustness are very high.

Moreover, the high latency typical of the wide area network (WAN) connections, and the intrinsically higher probability of temporary disconnections among clients and servers residing in different sites have historically been considered as very hard problems.

The idea behind our interest for WAN direct data access is not necessarily to force computations to be performed towards remote data. Hence, we do not believe that local storage clusters can be replaced at the present time, especially in the cases where a large computing farm is accessing the data. We believe that the "classical" use case where a farm can only access a storage element very "close" to it, is not the only choice anymore, as it is demonstrated, for example, in a production environment by the implementation of the ALICE computing model [13] [14] [15].

In a modern computing model dealing with data access for High Energy Physics experiments (but probably not only High Energy Physics), there are other very interesting use cases that can be easily accommodated with a well working WAN-wide data access, for example:

- Interactive data access, especially in the case of analysis applications that exploit the computing power of modern multicore architectures [11]. This would be even more precious for tasks like debugging an analysis or data-mining application without having to copy locally the data it accesses.
- Allow a batch analysis job to continue its execution if it landed in a farm whose local storage element lost the data files which were supposed to be present. They may instead be present in another site.
- Any other usage which could come to the mind of an user of the World Wide Web, which made interactivity available in a very easy way via WAN.

Hence, the objectives this kinds of systems are aiming for are:

- To be able to deploy a "unified worldwide storage", which transparently integrates functionalities at both site level and global level.

- To improve data access speed, transaction rate, scalability and efficiency.
- To allow dealing with Wide Area Networks, both for distributing the overall system and accessing its data.
- To implement fault tolerant communication policies.
- To implement fault tolerance through the “self healing” capabilities of the content of a Storage Element (SE).
- To give to remote SEs ways to remain synchronized with the metadata that describes their content.

These objectives also led us to address the issues related with the scalability of the whole design, and the fact that the focus on performance is a major requirement, since these aspects can make, in practice, the difference between being able to run data analysis applications and being not. The case study for these methods is the ALICE [13] [15] High Energy Physics experiment at CERN, where a system based on these principles has been incrementally deployed on an already-running production environment among many (about 60) sites spread all over the world, accessed concurrently by tens of thousands of data analysis jobs. However, given the generality of the discussed topics, the methods discussed in this paper can be extended to any other field with similar requirements.

### 1.1 The problem

High Energy Physics experiments, from the computing point of view, rely typically on statistics about events and on all the relative procedures to identify and classify them. Very often, in order to get sufficient statistics, huge amounts of data are analysed, in data stores which can reach sizes up to a few dozens of Petabytes, but which will likely grow much more in the future generations of experiments. Hence, the typical computing scenario deals with tens or hundreds million files in the repositories and tens of thousands of concurrent data analysing processes, often called *jobs*. Each job can open many files at once (e.g. about 100-150 in ALICE, up to 1000 in GLAST/Fermi), and keep them open for the time needed.
With these numbers, it is easy to understand why the file open transaction rate for the storage system can be very high. With respect to that, a rate of $O(10^3)$ file opens per second in a local cluster is not uncommon. In our experience the source of this traffic can be a combination of the local GRID site, a local batch system, a local PROOF [11] cluster, remote WAN-wide clients and occasional interactive users.

Historically, a common strategy, largely adopted in HEP, to handle very large amounts of data (on the order of the dozens of Petabytes per year) is to distribute the data among different computing centres, assigning a part of the data to each of them, possibly having some sites that are more important than others.

Among others, one of the consequences of this kind of architectures is that, for example, if a data processing job needs to access two data files, each one residing at a different site, an often

adopted solution is to make sure that a single site contains both of them, and the processing is started there.
Immediate consequences coming from the above scenario are, for instance, that a hypothetical data analysis job must wait for all of its data to be present at a site, before starting, or that the time to wait before starting can grow if some file transfers fail and have to be rescheduled. Moreover, the task of assigning files to sites (and replicating them) can easily degrade into a complex n-to-m matching problem, to be solved online, as new data comes and the old data is replicated in different places. An optimal solution is very difficult to achieve, given the typically very high complexity of such systems in the real deployments.

One could object that the complexity of an n-to-m online matching problem can be avoided in some cases by assigning to each site something like a "closed" subset of the data, i.e. a set of files whose processing does not need the presence of files residing elsewhere. This situation can theoretically evolve to one where the sites containing the so-called "trendy datasets" are overloaded, while others can be idle, or, alternatively, to a situation where the majority of the sites are manually forced to host the same data sets. Both situations, in our opinion, are undesirable.

### 1.2 File catalogues and metadata catalogues

Even if the discussed aspects are well known, however, it is in our opinion that the initiatives related to what we can call "transparent storage globalization" are not apparently progressing fast enough to allow a system designer to conceive a very large distributed production storage deployment using them in a transparent way. What happens in many cases is that those designs include some external component whose purpose is to take decisions about file placements, based on the knowledge of the supposed current global status of the storage system, i.e. the current (and/or desired) association of data to sites. Invariably, this kind of component takes the form of a simple-minded "file catalogue". Its deployment may represent a performance bottleneck in the architecture and also it can be a single point of failure, or a source of inconsistencies, as discussed later.

Here we must point out the subtle difference that lies, in our view, between a file catalogue and an application-related metadata catalogue. A file catalogue is a list of files, associated to other information, e.g. the sites/servers that host each particular file or the URLs to access them. When part of a common design, it often is the only way an application has to locate a file in a heavily distributed environment, hence it must be contacted for each file any application needs to access. A metadata catalogue, instead, contains information about the content of files, but not about where they can be accessed. Hence it can contain metadata about a non-existing file, or a file can have in principle no metadata entries. In our view, a worldwide storage system should shield the applications from its internals, like for example the tasks of locating files through sites or through servers.

Assigning the task of locating files to a comprehensive storage system, in our view, can be highly desirable for the intrinsic robustness of the concept. A file catalogue instead must deal with the fact that the content of the repository might change in an uncontrollable way (e.g. a broken media or other issues), opening new failure opportunities, which can also be due to mis-alignments between its content and the actual content of the repository.

To give a simpler example, we consider a completely different domain: the ID3 tags for music files. A technically very good example of a comprehensive metadata catalogue for them is e.g. the *www.musicbrainz.org* website. We could also think about creating a file catalogue, indexing all the music files owned by the world's population (and the scale would not be so different from the scale of the file catalogues of modern HEP experiments). The subtle difference among these two is that in the real world everybody is allowed not to own (or to loose or damage) a CD listed in this metadata catalogue. This would not be considered a fault by the maintainers of the metadata, and has no consequence on the information about that particular CD. In other words, this is just information about music releases which somebody could own and somebody not, and somebody else is unable to tell. Hence, an application willing to process a set of music files would only need to know which music files to process, and not how they can be accessed and in which data server in a worldwide deployment.
On the other hand, a "file catalogue"-like system aims at being a “perfect” representation of reality, and this adds some layers of complexity to the task, last but not least that it is computationally very difficult (if not impossible) just to verify that all the entries are correct and that all the CDs owned (and sold/exchanged) by the persons are listed and correctly attributed. Our point, when dealing with this kind of problems is that, from a general perspective, our proposal is not to promote a metadata catalogue to become a file catalogue, since finding or notifying the location of the requested resources is a task belonging to the storage system layer, not to the layer which handles the information bookkeeping. This might be considered a minor, or even subtle aspect, but it makes a very big difference from the architectural point of view when the scale of the system is very large and it is distributed worldwide. We tried to implement and deploy a system based on these concepts by using and contributing to the Xrootd/Scalla software suite for data access.

## 2. XROOTD AND THE SCALLA SOFTWARE SUITE

The basic system component of the Scalla software suite is a high-performance data server akin to a file server called *xrootd*. However, unlike NFS, the xrootd protocol includes a number of features like:

- Capability to accommodate up to several tens of thousands concurrent physical client connections per server;
- Capability to support several tens of thousands outstanding requests sharing the same physical connection;
- Communication optimizations, which allow clients to accelerate data transfers (e.g., overlapping requests, TCP multistreaming, vectored read requests etc.);
- An extensive fault recovery protocol that allows data transfers to be interrupted and continued at an alternate server;
- A comprehensive authentication/authorization framework;

- The possibility of interfacing a data server with external systems, like tape drives, in order to provide transparent hierarchical storage capabilities [19].
- Peer-to-peer elements that allow xrootd servers to be clustered together while still providing a uniform name space.

Xrootd's clustering is accomplished by the *cmsd* component of the system. This is a specialized server that can co-ordinate xrootd's activities and direct clients to appropriate servers in real-time. In essence, the system consists of:

- A logical data network (i.e., the xrootd servers);
- A logical control network (i.e., the cmsd servers), based on a proprietery message passing technology.

The control network is used to cluster servers while the data network is used to deliver actual data. We define a *node* as a server pairing of an xrootd with a cmsd.

A cmsd can assume multiple roles, depending on the nature of the task. In its manager role, the cmsd discovers the best server for a client file request and co-ordinates the organization of a cluster. In its server role, the cmsd provides sufficient information to its manager cmsd so that it can properly select a data server for a client request.

Hence, a server cmsd is essentially an agent running on a data server node. In its supervisor role, the cmsd assumes the duties of both manager and server. As a manager, it allows server cmsd's to cluster around it, it aggregates the information provided by the server cmsd's and forwards the information to its manager cmsd. When a request is made by its manager, the supervisor cmsd determines which server cmsd will be used to satisfy the request. This role parceling allows the formation of data server cells that cluster around a local supervisor, which, in turn, clusters around a manager cmsd.

We can now expand the definition of a node to encompass the role. A data server node consists of an xrootd coupled with a server cmsd, a supervisor node consists of an xrootd and a supervisor cmsd, and a manager node consists of an xrootd coupled with a manager cmsd. The term node is logical, since supervisors can execute on the same hardware used by data servers.

## 2.1 Cell-based organisation

To limit the amount of message traffic in the system, a cell consists of 1-to-64 server nodes. Cells then cluster, in groups of up to 64. Clusters can, in turn, form superclusters, as needed. As shown in Figure 1, the system is organized as a B-64 tree, with a manager cmsd sitting at the root of the tree. Since supervisors also function as managers, the term manager should subsequently be assumed to include supervisors.

A hierarchical organization provides a predictable message traffic pattern and is extremely well suited for conducting directed searches for file resources requested by a client. Additionally, its performance, which depends on the number of collaborating nodes, scales very quickly with

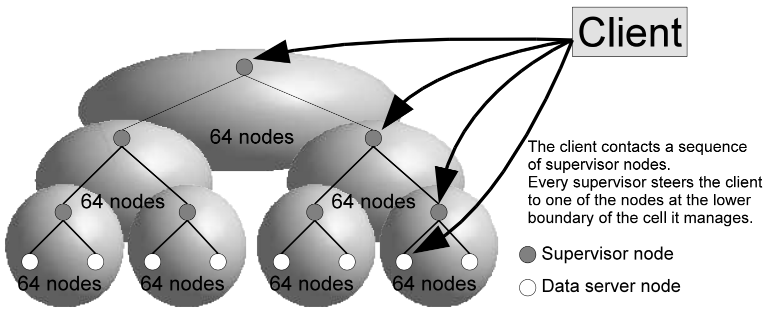
Client
64 nodes
64 nodes
64 nodes
64 nodes
64 nodes
64 nodes
64 nodes
The client contacts a sequence of supervisor nodes. Every supervisor steers the client to one of the nodes at the lower boundary of the cell it manages.
Supervisor node
Data server node

**Figure 1:** Cell-like organization of an xrootd cluster

only a small increase in messaging overhead. For instance, a two-level tree is able to cluster up to 262,144 data servers, with no data server being more than two hops away from the root node.

In order to provide enhanced fault-tolerance at the server side, manager nodes can be replicated. When nodes in the B-64 tree are replicated, the organization becomes a directed acyclic graph and maintains predictable message traffic. Thus, when the system is configured with *n* managers, there are at least *n* control paths from a manager to a data server node, allowing for a significant number of node failures before any part of the system becomes unreachable. Replicated manager nodes can be used as fail-over nodes or be asked to load balance the control traffic. Data server nodes are never replicated. Instead, file resources may be replicated across multiple data servers to provide the desired level of fault-tolerance as well as overall data access performance.

The main reason why we do not use the term “file system” when referring to this kind of system is the fact that some requirements of file systems have been relaxed, in order to make it easier to head for extreme robustness, performance and scalability. The general idea was that, for example, it is much easier to give up atomicity in distributed transactions than being unable to make the system grow if its performance/size is not sufficient. For example, right now there was no need to dedicate a serious effort in order to allow for atomic distributed transactions or distributed file locks. These functionalities, when related to extreme performance and scalability are still an open research problem, even more critical when dealing with high latency networks interconnecting nodes, clients and servers over a WAN. In HEP, in the vast majority of the cases, the data files are never updated, they are created and then accessed in read mode (and eventually deleted later), so these relaxations did not create problems to the computing models where this system was used up to now.

These kinds of simplifications were also related to the particular peer-to-peer like behaviour of the xrootd mechanism behind the file location functionalities. In the xrootd case, moreover, we can claim for instance that the fact that there is no database is not entirely true. In fact, *the database used to locate files is constituted by the message-passing-based aggregation of several hierarchical databases, which are the file systems of all the aggregated disk partitions.* Of course, these are extremely well optimised for their purpose and are up to date by definition; hence there is no need to replicate their content into a typically slow external database, just to locate files in sites or single servers.

Another characteristic of the xrootd daemon, which is extremely important, is the fact that each server hides its internal paths where the data is stored by means of a simple local string substitution in the file names. In our terminology this is called “localroot”. Doing this, each server in a cluster can export a namespace that is detached from the names of the mount points of a given server. Using this kind of feature, in a typical setup, all the servers are configured in order to export the same namespace, and hence the same file exported by more than one server will appear to have the same file name without the need of potentially complex external name translations.
This kind of internal filename prefix processing is also used to aggregate multiple mount points in the same server. Doing this, the processing applications are completely detached from the internal deployment details; all they see is a common coherent name space, and the computational overhead of this kind of local name translation is extremely low.

## 2.2 Fault tolerance

Other aspects, which we consider as very important, are those related to robustness and fault tolerance. From one side, a system designer has to be free to decide what the system should do if, for instance, a disk breaks, and its content is lost. Given the relative frequency of this kind of hardware problems, the manual recovery approach should be reduced to a minimum or not needed at all. The typical solutions exploited in fail-safe HEP data management designs are:

- The disk pool is just a disk-based cache of an external tape system, hence, if a disk breaks, its content will be (typically automatically) re-staged from the external units by some other server in the same cluster. Of course, the system must be able to work correctly even during this process.
- The files are replicated somewhere else. Hence, some system could schedule the creation of a new replica of the lost ones. The difficult part is that finding out which files were lost in a potentially large repository can be problematic.

The other aspect of fault tolerance is related to the client-server communication, typically based on TCP connections between clients and servers, and, in the xrootd case, also between server nodes. The operational principles that we applied rigorously are:

- A client must never return an error if a connection breaks for any reason (network glitches, etc), unless it failed after having retried for a certain number of times and/or failed to find an alternate location where to continue.
- The server can explicitly signal every potentially long operation, so that the client goes into a pause state from which it can be woken up later. This avoids having to deal too much with timeouts.
- Every inter-server connection is kept alive and eventually re-established if it seems to be broken.

## 3. MORE ABOUT STORAGE GLOBALISATION

The other interesting item related to the recent WAN-oriented features of the Scalla/xrootd suite is the possibility of building a unique repository, composed by several sub-clusters residing in different sites.

The xrootd architecture, based on a B-64 tree where servers are organised (or self-organise) into clusters [5], accommodates by construction the fact that servers can connect across a wide area network, provided that the communication protocol which connects them is robust enough. All this is accomplished with the introduction of a so-called *meta-manager* host, which the remote sites subscribe to. This new role of a server machine is the client entry point of a unique *meta-cluster*, which contains all the subscribed sites, potentially spread elsewhere.

Of course, one important requirement is that the remote sites are actually able to connect to this meta-manager and that each storage cluster is accessible from the other sites, but, an even more important one is that all the sub-clusters expose the same coherent name space. In other words, a given data file must be exported with the same file name everywhere, and, of course, the data servers must be accessible from outside.

With the possibility of accessing a very efficient coherent repository without having to know the location of a file in advance because the location task is handled by the storage system, we can implement many ideas. For instance, this mechanism is being used in the storage elements belonging to ALICE in order to quickly fetch a file that was supposed to be present on a storage element but for some reason is not accessible anymore. The next paragraph better explains the details of this kind of system and synthetically shows how this is performed in a way that is completely transparent to the client that requests the file.

## 4. THE VIRTUAL MASS STORAGE SYSTEM

As said, the set of Wide Area Network related features of the Scalla/Xrootd system played a major role in opening the possibility to evolve a storage system in the direction of being able to exploit Wide Area Networks in a productive way. The major design challenge was how to exploit this in order to augment the overall robustness and usefulness of a large distributed storage deployment.
The idea of letting data processing jobs access remote data was not new in the ALICE computing framework. For instance, this is the default way that the processing jobs have to access the so-called “conditions data”, which are stored in a Scalla/Xrootd-based storage cluster at CERN. Each processing job accesses 100-150 conditions data files (and 20,000 concurrent jobs are considered a normal activity): historically the WAN choice always proved itself a very good one, with a surprisingly good performance for that task, and a negligible failure rate.

On the other hand, the AliEn software [14], supported by its central services, already gives some sort of unique view of the entire repository, implementing namespace coherence across different sites by mapping the file names using a central relational database. Moving towards a model where the storage coherence is given by the storage system itself, and does not need external systems, would give an additional benefit to this key component of the ALICE computing. This advantage would be in the form of a performance enhancement, but also it would avoid the fact that an external system (like a relational database) cannot, by construction, keep easily track of the data files which can be lost due to malfunctioning disks. As already discussed, the form of a "metadata catalog" is by definition immune to this, and, in the recent times, the ALICE computing is slowly moving in this direction.

An additional consideration is that, from a functional point of view, we considered as a poor solution the fact that, once a missing file is detected, nothing is done by the storage system to fetch it. This is especially unfortunate when, for instance, the file that is missing in a site is available in a well-connected one, and pulling it immediately (and *quickly*) could be done without letting the requesting job fail.

A careful consideration of all these aspects led to thinking that there was a way to enhance the ALICE Scalla/Xrootd-based distributed storage in a way which automatically tries to fix this kind of problems, and at the same time does not disrupt the existing design, allowing for an incremental evolution of the storage design, distributed across several almost independent sites contributing to the project.

The generic solution to the problem has been to apply the following principles:

- All of the xrootd-based storage elements are aggregated into a unique worldwide meta-cluster, which exposes a unique and namespace-coherent view of their content;
- Each site that does not have tapes or similar mass storage systems considers the meta-cluster as its mass storage system. If an authorized request for a missing file comes to a storage cluster, then it tries to fetch it from one of the neighbour ones, as soon and as fast as possible.

The host which manages the meta-cluster (which in the Scalla/Xrootd terminology is called *meta-manager*) has been called *ALICE Global redirector*, and the idea for which a site sees the global namespace (to which it participates with its content) as its mass storage backend has been called *Virtual Mass Storage System* (VMSS).

In this context, we consider as very positive statements the facts that:

- In the xrootd architecture, locating a file is a very fast task (if the location is unknown to a manager server, finding it takes, in average, only a network round-trip time plus one local file system lookup);
- No central repositories of data or metadata are needed for the mechanism to work, everything is performed through real-time message exchanges;
- All of the features of the Scalla/Xrootd suite are preserved and enhanced (e.g. scalability, performance, etc.);

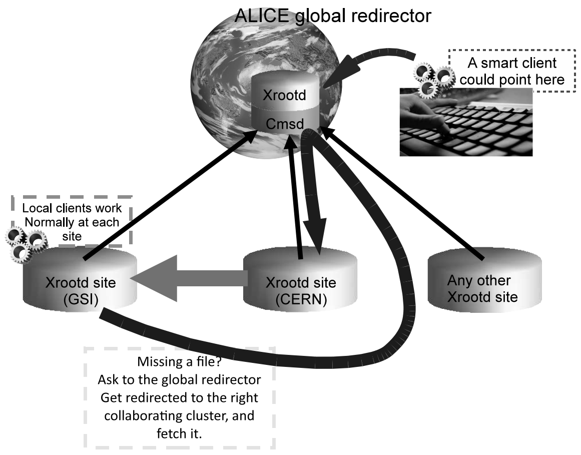
ALICE global redirector
Xrootd
Cmsd
A smart client could point here
Local clients work Normally at each site
Xrootd site (GSI)
Xrootd site (CERN)
Any other Xrootd site
Missing a file? Ask to the global redirector Get redirected to the right collaborating cluster, and fetch it.

**Figure 2:** An exemplification of the Virtual Mass Storage System.

- The design is completely backward compatible, e.g. in the ALICE case no changes were required to the complex software systems which constitute the computing infrastructure;

Hence, the system, as designed, represents a way to deploy and exercise a global real-time view of a unique multi-Petabyte distributed repository in an incremental way. This means that it is able to give a better service as more and more sites upgrade their storage clusters and join in.

Figure 4 shows a generic small schema of the discussed architecture. In the lower part of the figure we can see three cubes, representing sites with a Scalla/Xrootd storage cluster. If a client instantiated by a processing job (on the leftmost site labelled GSI) does not find the file it needs to open, then the GSI cluster (see picture) pauses the client and asks the global redirector to get the file, just requesting to copy the file from it. This copy request (which is just another normal xrootd client) is then redirected to the site that has the needed file, which is copied immediately to the leftmost site. The previously paused job can then continue its processing as if nothing particular had happened.

Hence, we could say that the general idea about the Virtual Mass Storage is that several distant sites provide a unique high performance file system-like data access structure; they constitute some sort of cloud of sites that collaborate in order to:

- Make some recovery actions automatic;
- Give an efficient way to move data between them, if this feature can be used by the overall system.

## 5. CONCLUSION AND CURRENT DIRECTIONS

By using the two mechanisms described (the possibility of efficiently accessing remote data and the possibility of creating inter-site meta-clusters) many things become possible, as these are

very generic mechanisms that encapsulate the technicalities but are not linked in any way to particular deployments.
For example, building a true and robust federation of sites now starts becoming much easier and efficient. For instance, one site could host the storage part, another one could host the worker nodes, without having to worry too much about the performance loss due to the latency of the link between the sites. Or both might have a part of the overall storage and appear as one, without having to deal with (generally less stable and very complex) "glue code" in order to build an artificial higher level view of the resources.
So far, all the tests and the production usages have been very successful, from the points of view of both performance and robustness. We believe that providing the possibilities of having a storage system able to work acrosss WANs properly and efficiently is a major accomplishment that will give many benefits, especially when dealing with user-level interactive data processing.

As discussed, the features that allow this kind of approach are quite generic, and it is foreseeable that other systems will follow a similar technological path.
With respect to the consistency between catalogues and the actual content of the remote site, other approaches are possible, for instance trying to update in real-time the content of the catalogues every time an inconsistency is detected. This approach could be a very interesting addition, that could also complement the VMSS approach, trying instead to fetch the file that was supposedly lost. In this case, both the catalogues and the storage systems would have a way to fix their content and/or propagate changes.

Other aspects worth facing are those related to the so-called "storage globalisation". For instance, a working implementation of some form of latency/location-aware load balancing would be a very interesting feature for data analysis applications in the need to access remote files efficiently. At the present time, the xrootd load balancing algorithm, indeed, only considers the load of the servers (network throughput and CPU consumption) as a metric in order to decide where to redirect a client. As always, the challenge would be not only to provide one more feature, but to provide it in a way which is compatible with the requirements for extreme performance and scalability.

## References

[1] Howard JH et al. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6, Feb, 1988.

[2] Bar-Noy A, Freund A, Naor JS. On-line Load Balancing in a Hierarchical Server Topology. *Proceedings of the 7th Annual European Symposium on Algorithms*, 1999.

[3] Lustre: a scalable, secure, robust, highly-available cluster file system. http://www.lustre.org/ [July 2009]

[4] The Scalla/xrootd Software Suite. http://savannah.cern.ch/projects/xrootd http://xrootd.slac.stanford.edu/ [July 2009]

[5] Furano F, Hanushevsky A. Managing commitments in a Multi Agent System using Passive Bids. iat,pp.698-701, *IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'05)*, 2005.

[6] Hanushevsky A, Weeks B. Designing high performance data access systems: invited talk abstract. *Proceedings of the 5th international Workshop on Software and Performance* (Palma, Illes Balears, Spain, July 12 - 14, 2005). WOSP '05. ACM, New York, NY, 267-267. DOI= http://doi.acm.org/10.1145/1071021.1071053.

[7] Dorigo A, Elmer P, Furano F, Hanushevsky A. Xrootd - A highly scalable architecture for data access. *WSEAS Transactions on Computers*, Apr. 2005.

[8] Hanushevsky A. Are SE architectures ready for LHC? *Proceedings of ACAT 2008: XII International Workshop on Advanced Computing and Analysis Techniques in Physics Research.* http://acat2008.cern.ch/ .

[9] Furano F, Hanushevsky A Data access performance through parallelization and vectored access. Some results. *CHEP07: Computing for High Energy Physics*. Journal of Physics: Conference Series 119 Volume 119 (2008) 072016 (9pp)

[10] ROOT: An Object-Oriented Data Analysis Framework
http://root.cern.ch [July 2009]

[11] Ballintijn M, Brun R, Rademakers F, Roland G. Distributed Parallel Analysis Framework with PROOF.
http://root.cern.ch/twiki/bin/view/ROOT/PROOF . [July 2009]

[12] The BABAR collaboration home page.
http://www.slac.stanford.edu/BFROOT . [July 2009]

[13] The ALICE home page at CERN
http://aliceinfo.cern.ch/ . [July 2009]

[14] ALICE: Technical Design Report of the Computing. June 2005. ISBN 92-9083-247-9.
http://aliceinfo.cern.ch/Collaboration/Documents/TDR/Computing.html

[15] Betev L, Carminati F, Furano F, Grigoras C, Saiz P. The ALICE computing model: an overview. *Third International Conference "Distributed Computing and Grid-technologies in Science and Education"*, GRID2008, http://grid2008.jinr.ru/

[16] Feichtinger D, Peters AJ. Authorization of Data Access in Distributed Storage Systems. *The 6th IEEE/ACM International Workshop on Grid Computing*, 2005.
http://ieeexplore.ieee.org/iel5/10354/32950/01542739.pdf?arnumber=1542739 [July 2009]

[17] the IEEE Computer Society's Storage System Standards Working Group.
http://ssswg.org/ [January 2009]

[18] Patterson RH, Gibson GA, Ginting E, Stodolsky D, Zelenka J. Informed prefetching and caching. *Proceedings of the 15th ACM Symposium on Operating Systems Principles.*, 1995.

[19] Hierarchical storage management, From Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/Hierarchical_storage_management [July 2009]

[20] ALICE Grid Monitoring with MonALISA. Realtime monitoring of the ALICE activities.
http://pcalimonitor.cern.ch/ [July 2009]