NorduGrid Storage – Overview and Design of a Service-Oriented Storage System

Jon K. Nilsen[1], Salman Toor[2], Zsombor Nagy[3], and Bjarte Mohn[4]

[1] University of Oslo, Dept. of Physics, P. O. Box 1048, Blindern, N-0316 Oslo, Norway
University of Oslo, Center for Information Technology, P. O. Box 1059, Blindern, N-0316 Oslo, Norway
j.k.nilsen@usit.uio.no
[2] Dept. Information Technology, Div. of Scientific Computing Uppsala University, Box 256, SE-751 05 Uppsala, Sweden
salman.toor@it.uu.se http://www.uu.se
[3] Institute of National Information and Infrastructure Development
NIIF/HUNGARNET, Victor Hugo 18-22, H-1132 Budapest, Hungary
zsombor@niif.hu
[4] Dept. of Physics and Astronomy, Div. of Nuclear and Particle Physics, Uppsala University, Box 535, SE-75121 Uppsala, Sweden
bjarte.mohn@fysast.uu.se

Abstract. There is an ever increasing need to utilize geographically distributed hardware resources, both in terms of CPU and in terms of storage. The service-oriented architecture provides a natural framework for managing these resources. The next generation Advanced Resource Connector (ARC) is a service-oriented Grid solution that will provide the middleware to represent distributed resources in one simple framework. In this paper, we will present an overview of ARC's novel storage system – the NorduGrid storage – itself a set of services providing a self-healing, flexible Grid storage solution. We will also present some first proof-of-concept test results from a deployment of the storage system distributed across three different countries.

1 Introduction

The challenge of building a reliable, self-healing, fault-tolerant, consistent data management system at the web-scale is an interesting task. Making the system work in a heterogeneous, distributed environment like the Grid is even more interesting. An increasing number of applications demand not only increased CPU power, but also vast amounts of storage space. The required storage space is not only restricted to the duration in which the application runs; the data should often be available for years afterwards, in certain cases even for decades. Nowadays, we can easily find single Grid jobs which produce gigabytes or even terabytes of data, ramping up the requirements of storage systems to the petabyte-scale

and beyond. To make the storage system useable to Grid users, the system must provide reliable and secure file transfer protocols, a cataloging system and secure storage of the data. Several projects and designs have emerged to address such challenges [1, 2].

In the advent of the next generation of the Advanced Resource Connector (ARC) Grid middleware [3] (new release due during fall 2009), we present the new NorduGrid storage [4]. This distributed storage system is designed to provide an easy to use, flexible and scalable system that can offer native storage and at the same time provide access to third-party solutions by using the same, uniform interface.

This paper is organized as follows. After describing general Grid storage systems in 2 and related work in Section 3, we give a bird's eye view of the next-generation Advanced Resource Connector (ARC) in Section 4. An overview of the NorduGrid storage is given in Section 5, while the architecture of the storage system is elaborated in Section 6. In Section 7 we give some early, proof-of-concept results, before concluding in Section 8.

2 Characteristics of Grid Storage

A Grid storage system provides its clients with access to data stored at remote storage systems. A traditional architecture (see e.g. [5, 6]) typically consists of an *indexing service*, indexing files from storage resources, a *replication service* for managing replica locations, and a (often centralized) *metadata catalog* imposing a global namespace on top of the resources. Data access can be handled either through a file transfer service or directly through the storage resource by quering the metadata catalog.

While this architecture has strongly improved the accessability of physically distributed data, common Grid storage solutions typically have some limitations: The metadata catalog, which is in most cases centralized can quickly become a bottleneck. In typical Grid storage systems, the storage resource is unaware of the state of its files, making consistency guarantees hard to give. Traditional Grid solutions often take for granted that the resource are dedicated for the Grid, and in some cases even require specific operating systems, thus restricting the amount of available hardware resources[5]. In addition, lack of standardization often makes interoperability between storage systems non-trivial. The NorduGrid Storage has as a goal to meet these limitations to provide a truly distributed, self-healing, flexible Grid storage solution, with a replicated metadata catalog, state aware storage resources and an operating system agnostic implementation.

3 Related Work

Many Grid projects have developed their own, aplication specific solutions. However, some general Grid storage solutions are widely adopted:

[5] It is often hard to convince a system administrator to install a specific operating system to be able to share her/his hardware resources in a Grid.

dCache [7] is a service oriented storage system which combines heterogeneous storage elements to collect several hundreds of terabytes in a single namespace. Originally designed to work on a local area network, dCache has proven to be usefull also in a Grid environment, with the NDGF (Nordic Data Grid Facillity) dCache installation [8] as the largest example. There, the core components, such as the metadata catalog, indexing service and protocol doors are run in a centralized manner, while the storage pools are distributed. The NorduGrid storage, designed to have multiple instances of all services running in a Grid environment, will not need a centralized set of core services. Additionally, dCache is relatively difficult to deploy and integrate with new applications. Being a more light-weight and flexible storage solution, the NorduGrid storage aims more towards new user groups less familiar with Grid solutions.

XTtreemFS [9] is an object-based file system specifically designed for the Grid environment of the operating system XTreemOS. Being object-based, XTreemFS have separated the metadata of files and the file content, or *objects* in the same fashion as the NorduGrid storage. Being a tightly integrated part of an operating system enables XTreemFS to implement functionality very similar to that of local file systems while being a distributed Grid storage. However, this also enforces a specific operating system, requiring dedicated hardware resources.

Based on the client-server model, the Storage Resource Broker (SRB) [10, 11] provides a flexible data grid management system. It allows uniform access to heterogeneous storage resources over a wide area network. Its functionality, with a uniform namespace for several Data Grid Managers and file systems, is quite similar to the functionality offered by our Gateway service (see Section 6.2). However, being built as a middleware on top of other major storage solutions, SRB does not offer its own storage solution.

Scalla is a widely used software suite consisting of an xrootd server for data access, and an olbd server for building scalable xrootd clusters [12]. Originally developed for use with the physics analysis tool root [13], xrootd offers data access both through the specialized xroot protocol and through other third-party protocols. The combination of the xrootd and olbd components offers a cluster storage designed for low latency, high bandwidth environments. In contrast, the NorduGrid storage is optimized for high latency and is more suitable for the Grid environment.

4 The Advanced Resource Connector

The next generation of Advanced Resource Connector (ARC) Grid middleware is developed by NorduGrid [14] and the EU KnowARC project [15]. It consists of a set of pluggable components. These components are the fundamental building blocks of the ARC services and clients. ARC services run inside a container called the Hosting Environment Daemon (HED) and there are four kinds of pluggable components with well defined tasks: Data Management Components are used to transfer the data using various protocols, Message Chain Components are responsible for the communication within clients and services as well as between

the clients and the services, ARC Client Components are plug-ins used by the clients to connect to different Grid flavors, and Policy Decision Components are responsible for the security model within the system.

5 The NorduGrid Storage

The NorduGrid storage consists of a set of SOAP based services residing within HED. Together, the services provide a self-healing, reliable, robust, scalable, resilient and consistent data storage system. Data is managed in a hierarchical global namespace with files and subcollections grouped into collections[6]. A dedicated root collection serves as a reference point for accessing the namespace. The hierarchy can then be referenced using Logical Names. The global namespace is accessed in the same manner as in local filesystems.

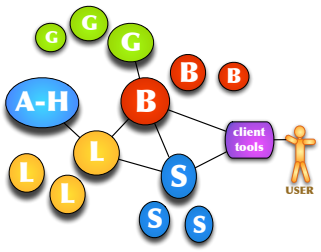

Fig. 1. Schematic of the NorduGrid storage architecture. The figure shows the main services of the NorduGrid storage and the communication channels. The B stands for Bartender, the S for Shepherd, the L for Librarian, the G for Gateway and the A-H stands for A-Hash. The straight lines denote the communication between services.

Being based on a service-oriented architecture, The NorduGrid storage consists of a set of services as shown in Fig. 1. The services are as follows: The Bartender (B) provides the high-level interface to the user; the Librarian (L) handles the entire storage namespace, using the A-Hash (A-H) as a metadatabase; the Gateway (G) provides access to third-party storage systems; and the Shepherd (S) is the frontend for the physical storage node. See Section 6 for a detailed discussion of the different services. The services communicate with each other through the Message Chain Components in HED. The communication channels are depicted by straight lines in Fig. 1.

[6] A concept very similar to files and directories in most common file systems.

6 Architecture of the NorduGrid Storage

In a service-oriented architecture the role of each service is well defined. Available objects[7] in the system are identified by unique global IDs. These IDs are categorized according to the object type:

- Each file and collection has a unique ID (GUID).
- Services are uniquely identified by a serviceID.
- The Shepherds in the system identify their files by a referenceID.

Each object in the NorduGrid storage has a globally unique ID. A collection contains files and other collections, and each of these entries has a name unique within the collection very much like entries in a standard directory on a local filesystem. Besides files and collections, the storage system has a third type of entry called mount-points, which are references to the third-party storages within the global namespace.

Replicas in a distributed storage system can have different states; they can be broken, deleted, partially uploaded, etc. In the NorduGrid storage, all replicas have assigned a state, some of which are '**alive**' (if the replica passed the checksum test, and the Shepherd reports that the storage node is healthy), '**invalid**' (if the replica has a wrong checksum, or the Shepherd claims it has no such replica), '**offline**' (if the Shepherd is down) and '**creating**' (if the replica is in the state of uploading).

In the following, the details of the architecture are presented in three parts: First we will discuss the details of the core components, second we will discuss some of the important features provided by the system, and third we will discuss the security model.

6.1 Core Components

The core components (Fig. 1) are described as follows:

- The **Bartender** provides a high-level interface for the storage system. Clients connect to the Bartender to create and remove files, collections and mount-points using their Logical Names. The Bartender communicates with the Librarian and Shepherd services to execute the clients' requests. However, the actual file data does not go through the Bartender, instead file transfers are directly performed between the storage nodes and the clients. There could be any number of independent Bartender services running in the system, providing high-availability and load-balancing.
- The **Librarian** manages the hierarchy and metadata of files, collections and mount points, as well as the health information of the Shepherd services. In addition, the Librarian handles the information about registered Shepherd services. The Librarian receives heartbeat messages from the Shepherds and changes replica states automatically if needed. The Librarian uses the

[7] Files, collections, mount points, etc.

A-Hash for consistently storing all metadata. This makes the Librarian a stateless service, thus enabling the system to have any number of independent Librarian services, again providing high-availability in the system.

- The **Shepherd** services runs as front-ends on storage nodes. A Shepherd service reports to a Librarian about the node's health state in terms of replicas. While the Bartender initiates file transfers, the actual transfers go directly between the storage node and the clients.
  When a new replica upload is initiated, the Shepherd generates a referenceID which refers to the replica within that Shepherd. Each Shepherd has a unique serviceID, so with these two IDs the replica can be unambiguously referenced. This is called a Location of the replica.
- The **A-Hash** is a hash table for consistently storing data in property-value pairs. All metadata about files, collections, mount point, Shepherd's health status, and so forth, is stored in the A-Hash. As the A-Hash is the service storing the entire state of the storage system, it is absolutely crucial for the sttorage system that the A-Hash is consistent. The distribution and replication of this service is therefore both necessary and challenging.

6.2 Features

- **Heartbeat monitoring:** In the proposed architecture, each Shepherd periodically sends heartbeats to a Librarian with information about replicas whose state changed since the last heartbeat, and the corresponding GUIDs. These heartbeats are then stored in the A-Hash, making them visible to all the Librarians in the system. If any of the Librarians notices that a Shepherd is late with its heartbeat, it will mark all the replicas in that Shepherd as offline.
- **Replication:** Shepherds periodically ask the Librarian if the file of the replica stored on its storage node has enough replicas. If the file does not have enough replicas, the Shepherd informs the Bartender and the Bartender initiates a put request that returns a Transfer URL to the Shepherd. The Shepherd finally uploads the new replica. The Shepherd which gets the new replica notifies the Librarian that the replica is alive. The Librarian then adds this to the corresponding file.
- **Deletion:** In a replicating storage system, there are several possible solutions for file deletion, since both the replicas and the metadata need to be removed. In our solution, we use the process of *lazy deletion* [16]. When a client requests that a file should be deleted (and the user has the proper permissions) the Bartender instructs the Librarian to remove the file's GUID from the A-Hash. When the Shepherd, periodically checking all its replicas, discovers that a replica has no file (and hence, no Location), it will automatically delete the replica.
- **Fault tolerance:** Due to the unpredictable nature of the Grid environment, it is essential to have some degree of automatic recovery system in case of unexpected failures. Fault tolerant behavior is required both at the level

of metadata and on the level of physical storage. While the work on fault-tolerant metadata is still in progress, the following two scenarios will explain the currently available recovery mechanism for physical storage:

- In the case of a file having an invalid checksum, the Shepherd immediately informs the Librarian and the Librarian changes the state of the given replica to '**invalid**'. To recover its replica, the Shepherd contacts a Bartender and asks for another replica of the file. The Bartender chooses a valid replica, initiates a file transfer from a Shepherd having the replica, and returns the TURL to the Shepherd with the invalid replica. When the Shepherd has received the replica and compared the checksum, it notifies the Librarian that the replica is alive again.
- In the case of a Shepherd going offline, a Librarian will, as mentioned earlier, notice the lack of heartbeats and invalidate all the replicas, initiating new replication for all the files stored in this storage node. However, if the Shepherd again comes online, there will evidently be more replicas than needed. The first Shepherd to notice this will set its replica's state to '**thirdwheel**', i.e. obsolete. At the next occasion, the Shepherd will remove the replica, if and only if it has the only '**thirdwheel**' replica of this file. If there are more replicas with this state, all replicas will be set back to '**alive**' and the process is repeated. This scenario will be discussed further in Section 7.

– **Client tools:** Being the only part a user will (and should) see from a storage system, the client tools are an important part of the NorduGrid storage. Currently ARC supports two ways of accessing the storage solution. The **Command-line Interface** (CLI) provides access to the storage through the methods `stat`, `makeCollection`, `unmakeCollection`, `putFile`, `getFile`, `delFile`, `list` and `move`. Methods for modifying access and ownership will be available in the near future. The **FUSE module** provides a high-level access to the storage system. Filesystem in Userspace (FUSE) [17] provides a simple library and a kernel-userspace interface. Using FUSE and the ARC Python interface, the FUSE module allows users to mount the storage namespace into the local namespace, enabling the use of graphical file browsers. It is worth mentioning that the client tools queries the storage system through the Bartender only. Currently upload and download is realized through HTTP(S), but there are plans to add support for other protocols, such as SRM and GridFTP.
– **Gateways:** Gateways are used to communicate with the external storage managers. While designing this service, care was taken to retain the transparency of the global namespace while using the external storage systems, and to develop a protocol-oriented service (i.e., the external storage managers which support a certain protocol should be handled using the corresponding gateway service). This approach provides flexibility while avoiding multiple Gateway services for different storage managers. Currently, the available Gateway service is based on the gridftp protocol.
The system stores the mount points which consists of URLs of external stores. In the case where a request is related to the downloading of files from

the external store, the Gateway service first checks the status of the file and then sends the Transfer URL (TURL) to the client via the Bartender. Using this TURL, the client can directly get the file from the external store.

6.3 Security Model

As is the case for all openly accessible web services, the security model is of crucial importance for the NorduGrid storage. The security architecture of the storage can be split into three parts; the inter-service authorization; the transfer-level authorization; and the high-level authorization:

- The **inter-service authorization** maintains the integrity of the internal communication between services. There are several communication paths between the services in the storage system. The Bartenders send requests to the Librarians and the Shepherds, the Shepherds communicate with the Librarians and the Librarians talk with the A-Hash. If any of these services is compromised or a new rogue service gets inserted in the system, the security of the entire system is compromised. To enable trust between the services, they need to know each other's Distinguished Names (DNs). This way, a rogue service would need to obtain a certificate with that exact DN from some trusted Certificate Authority (CA).
- The **transfer-level authorization** handles the authorization in the cases of uploading and downloading files. When a transfer is requested, the Shepherd will provide a one-time Transfer URL (TURL) to which the client can connect. In the current architecture, this TURL is world-accessible. This may not seem very secure at first. However, provided that the TURL has a very long, unguessable name, that it is transfered to the user in a secure way and that it can only be accessed once before it is deleted, the chance of being compromized is very low.
- The **high-level authorization** considers the access policies for the files and collections in the system. These policies are stored in A-Hash, in the metadata of the corresponding file or collection, providing a fine-grained security in the system.

The communication with and within the storage system is realized through HTTPS with standard X.509 authentication.

7 Testing and Discussion

Even though the NorduGrid storage is in a pre-prototype state, it is already possible to deploy and use the system for testing purposes. To properly run a proof-of-concept test, the resources need to be geographically distributed. In our test scenarios we utilized resources in three different countries.

In our test deployment, we used two nodes from Uppsala Multidisciplinary Center for Advanced Computational Science (UPPMAX), Sweden, three nodes from the Center for Information Technology (USIT) at the University of Oslo,

Norway, and one node from National Information Infrastructure Development Institute (NIIF), Hungary. The services were distributed as shown in Fig. 2:

- An A-Hash runs at UPPMAX.
- A Bartender runs at USIT.
- A Librarian runs at UPPMAX.
- In total five Shepherds were used for the tests: Three at USIT, having 100GB storage space each, one at UPPMAX, with 20GB, and one at NIIF, providing 16GB of storage space.

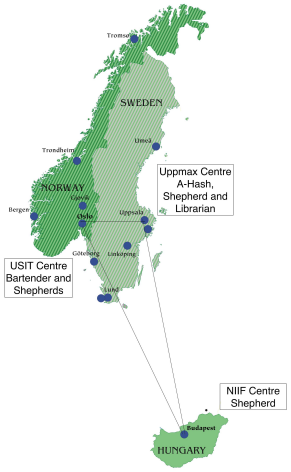

Fig. 2. The map shows the geographical distribution of the services in the test setup.

The following tests were carried out:

- **Test 1:** The distribution of data after uploading 10 large size (1 GB) files with one replica each.
- **Test 2:** The distribution of files after uploading 100 small size (1 kB) files with one replica each, and with simultanious uploading.
- **Test 3:** The distribution of data after repeating Tests 1 and 2 with three replicas.
- **Test 4:** System behavior while some of the Shepherds are offline.

In the tests we used two client machines: One ASUS Eee 901 on a wireless network in Uppsala, and one Dell PowerEdge 1425SC with 10Gb ethernet connection at USIT. All the tests were performed and worked as expected: All the files were both uploaded and downloaded with correct checksums and they all got the requested number of replicas within a reasonable time. However, some of the tests deserve extra attention.

Fig. 3 illustrates two test results, corresponding to Test 2 and Test 3: The orange (light grey) bars show the distribution after uploading 100 small files simultaneously, without replication. This gives an indication of how the system balances the load of many clients uploading at the same time. The green (dark grey) bars show the distribution with the same kind of uploading, but with the clients requesting three replicas for each file. The difference here is that the system simultaneously has to handle both replication and the clients requesting uploads. When the Bartender chooses a location for which to put a replica, it generates a list of Shepherds that (a) do not have a replica of the file, and (b) are not already in the process of uploading the replica. Next, it draws a Shepherd from the list at random, using a uniform random number generator. When uploading without asking for replicas we would therefore expect a relatively flat distribution of the files, as can be seen in Fig. 3.

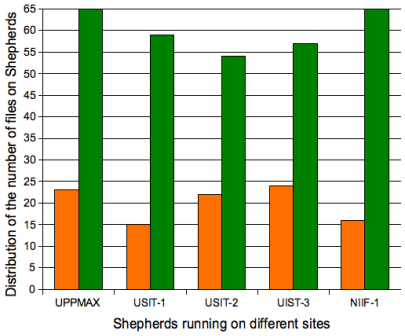

Fig. 3. The distribution of replicas over the geographically distributed storage nodes. Green bars show the distribution in the case where we upload 100 files requesting 3 replicas, whereas orange bars show the distribution after uploading 100 files without replication.

	UPPMAX	USIT-1	USIT-2	USIT-3	NIIF
Load(GB)	6.683	7.638	7.650	4.773	2.289

Table 1. Overall load distribution on the storage nodes after Tests 1, 2 and 3 were finished

Table 1 shows the disk usage after Tests 1, 2 and 3. It is worth noticing here that the bandwidth to NIIF was significantly lower than between UPPMAX and USIT. If the bandwidth is saturated and the storage node is busy, e.g., checksumming a large file, the heartbeat from this Shepherd may be delayed, causing the Bartender not to choose this Shepherd for the next replica. This may explain the relatively low storage load on the NIIF storage node.

	Before (GB)	During (GB)	After (GB)
UPPMAX	4.888	8.798	8.798
USIT-1	7.820	9.778	6.843
USIT-2	6.843	-	4.888
USIT-3	7.820	9.774	8.798
NIIF	3.320	2.344	1.367
Sum	30.69	30.69	30.69

Table 2. Storage load before, during and after a Shepherd outage.

A significant feature of the NorduGrid storage is its automatic self-healing. Test 4 addresses this feature by studying the effect of taking one Shepherd out of the system, and later reinserting it. Table 2 shows the storage distribution in three states: Before interrupting a Shepherd, after the USIT-2 Shepherd is interrupted and redistribution of replicas is finished, and alive, when the Shepherd is

restarted and the system again has stabilized. We can see that all files are properly distributed between the remaining Shepherds when one of them is disrupted and that the storage load evens out again when the Shepherd comes back online. When the system discovers that there are more replicas than needed, the first Shepherd noticing will set mark its replica as obsolete. Since the failing Shepherd didn't lose any replicas while being down, redistribution of replicas is just a matter of deleting obsolete replicas. We also see that the NIIF node actually got fewer files when USIT-2 went offline. If two Shepherds simultaneously start uploading and replicating a missing replica, there will be too many replicas. Here, a big replica on the NIIF node has randomly been chosen as obsolete. However, the total storage usage remains the same in all three cases.

8 Conclusion and Future Work

The proposed system is still in an early phase of development, but our test results demonstrate that the architecture is robust enough to handle the challenges for distributed large-scale storage. Much effort is required to make the system production ready. However, we believe that continuing in the same direction will enable us to provide a persistent and flexible storage system which can fulfill the needs of even the most demanding scientific community.

Some key areas need special attention and effort to make the proposed storage system even more stable, reliable and consistent:

- **The A-Hash**: This is currently centralized. To avoid a single point of failure in the system, and to improve the system performance, the A-Hash needs to be distributed.
- **Performance optimization:** To make a storage system ready for production, one needs to discover and improve on possible bottlenecks within the system. As soon as the A-Hash is distributed, other possible bottlenecks, such as load-balancing and self-healing mechanisms, will be investigated.
- **Protocols:** To ease the interoperability with third-party storage solutions and clients, the system needs to support storage protocols such as SRM and GridFTP. In addition the system will come with its own ARC protocol.

While the work in the above mentioned areas is still in progress, we have already shown that the NorduGrid storage already is in a state where initial real-life tests can be done. We have described a simple, yet strong architecture which we believe will benefit communities in need of a lightweight, yet distributed storage solution.

9 Acknowledgements

We wish to thank Mattias Ellert for helpful discussions and guidance on the ARC middleware, to Oxana Smirnova, Alex Read and David Cameron for vital comments and proof reading. In addition, we like to thank UPPMAX, NIIF and USIT for providing resources for running the storage tests.

The work has been supported by the European Commission through the KnowARC project (contract nr. 032691) and by the Nordunet3 programme through the NGIn project.

References

1. Hoschek, W., Jaen-Martinez, J., Samar, A., Stockinger, H., Stockinger, K.: Data management in an international data grid project, Springer-Verlag (2000) 77–90
2. Deng, Y., Wang, F.: A heterogeneous storage grid enabled by grid service. SIGOPS Oper. Syst. Rev. **41**(1) (2007) 7–13
3. Ellert, M., et al.: Advanced Resource Connector middleware for lightweight computational Grids. Future Gener. Comput. Syst. **23**(1) (2007) 219–240
4. Nagy, Z., Nilsen, J.K., Toor, S.: Documentation of the ARC storage system. NorduGrid. NORDUGRID-TECH-17.
5. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., Tuecke, S.: The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. Journal of Network and Computer Applications **23** (2001) 187–200
6. Stewart, G.A., Cameron, D., Cowan, G.A., McCance, G.: Storage and data management in egee. In: ACSW '07: Proceedings of the fifth Australasian symposium on ACSW frontiers, Darlinghurst, Australia, Australia, Australian Computer Society, Inc. (2007) 69–77
7. de Riese, M., Fuhrmann, P., Mkrtchyan, T., Ernst, M., Kulyavtsev, A., Podstavkov, V., Radicke, M., Sharma, N., Litvintsev, D., Perelmutov, T., Hesselroth, T.: dCache Book
8. Behrmann, G., Fuhrmann, P., Grønager, M., Kleist, J.: A Distributed Storage System with dCache. Journal of Physics: Conference Series 119 (2008) 062014
9. Hupfeld, F., Cortes, T., Kolbeck, B., Stender, J., Focht, E., Hess, M., Malo, J., Marti, J., Cesario, E.: The xtreemfs architecture—a case for object-based file systems in grids. Concurr. Comput. : Pract. Exper. **20**(17) (2008) 2049–2060
10. Baru, C., Moore, R., Rajasekar, A., Wan, M.: The SDSC storage resource broker. In: CASCON '98: Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research, IBM Press (1998) 5
11. Wan, M., Rajasekar, A., Moore, R., Andrews, P.: A Simple Mass Storage System for the SRB Data Grid. In: MSS '03: Proceedings of the 20 th IEEE/11 th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03), Washington, DC, USA, IEEE Computer Society (2003) 20
12. Boeheim, C., Hanushevsky, A., Leith, D., Melen, R., Mount, R., Pulliam, T., Weeks, B.: Scalla: Scalable Cluster Architecture for Low Latency Access Using xrootd and olbd Servers. Technical report, Stanford Linear Accelerator Center (2006)
13. Brun, R., Rademakers, F.: ROOT – An object oriented data analysis framework. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **389**(1-2) (April 1997) 81–86
14. : NorduGrid Collaboration.
15. : EU KnowARC project.
16. Celis, P., Franco, J.: The analysis of hashing with lazy deletions. Inf. Sci. **62**(1-2) (1992) 13–26
17. : Filesystem in Userspace.