Service Oriented ARC Storage; An Overview and Design

Jon K. Nilsen[†,‡] Salman Toor[*] Zsombor Nagy[¶]
Bjarte Mohn[§]
[*]Scientific Computing Division, IT Department, Uppsala University, Sweden
[†]Center for Information Technology, University of Oslo, Norway
[‡]Department of Physics, University of Oslo, Norway
[§]Department of Physics and Astronomy, Uppsala University, Sweden
[¶]National Information Infrastructure Development Institute, Hungary

j.k.nilsen@usit.uio.no salman.toor@it.uu.se zsombor@niif.hu
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 the ARC storage system, itself a set of services providing a self-healing, flexible Grid storage solution. We will also present some first proof-of-concept test results, deploying the storage system distributed between three different countries.**

I. Introduction

The challenge of building a reliable, self-healing, fault-tolerant, consistent data management system in a web scale is an interesting task. Making the system work in a heterogeneous, distributed environment like the Grid is even more interesting. An increasing amount of applications demands 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, producing gigabytes or even terabytes of data, ramping up the requirements of storage systems to the petabyte scale and beyond. Taking into account that hardware is inherently unstable[1], the need for replicating, self-healing storage is evident. 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 [16], [14].

In the advent of the next generation Advanced Resource Connector (ARC) Grid middleware [1] (due during Fall 2009), we present the ARC storage system [19]. 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 like dCache by using the same, uniform interface. Being part of ARC, the storage system is based on a service oriented architecture, in which each major component of the system runs as a separate service within the ARC Hosting Environment Daemon (HED) [10]. The HED service container gives the capability of flexible replacement of the components as well as the possibility to introduce modifications in the future.

This paper is organized as follows. After describing related work in Section II, we give the bird's eye view of the next generation Advanced Resource Connector (ARC) in Section III. An overview of the ARC Storage is given in Section IV, while the architecture of the storage system is elaborated in Section V. In Section VI we give some early, proof-of-concept results, before concluding in Section VII.

II. Related Work

dCache: dCache is a storage system [15] [7] which combines heterogeneous storage elements in the order of several hundreds of terabytes in a single namespace. It additionally supports standard and native protocols like gridftp, srm, dcap and gsidcap. dCache is a joint effort between Desy [2], Fermi National Laboratory [3], the Nordic Data Grid Facility (NDGF) [4] and several other collaborators. dCache has proven to be a very stable and scalable solution. However, it has a relatively high threshold for understanding and deployment. The ARC Storage, being a light-weight and flexible storage solution, aims more towards new user groups less familiar with Grid solutions.

BigTable: Bigtable is a distributed storage system managing structured data in the petabyte scale [13]. It is currently used within Google, in projects like web indexing, Google Earth and Google Financing. Bigtable has several

[1] E.g., if one hard drive has an uptime of 10,000 hours, one of 1,000 hard drives will fail after 10 hours.

interesting features, one of which is the distributed lock service, Chubby [9]. Chubby is a high-availability, distributed locking service sitting on top of the B^+-tree architecture of Bigtable. Chubby has several features similar to the distributed A-Hash service, among which the use of the Paxos algorithm [18], [12], is the most striking. A major caveat for the Grid community is that Bigtable is neither free, open-source, nor available to the public.

Storage Resource Broker: Based on the client-server model, The Storage Resource Broker (SRB) [6], [20] provides a flexible data grid management system. It allows uniform access for connecting 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. However, being built as a middleware on top of other major storage solutions, SRB does not offer its own storage solution.

Scalla: Scalla is a widely used software suite consisting of an xrootd server for data access, and an olbd server for building scalable xrootd clusters [8]. Originally developed for use with the physics analysis tool root, 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 ARC Storage is optimized for high latency, more suitable for the Grid environment.

III. The Advance Resource Connector

The next generation Advanced Resource Connector (ARC) Grid middleware [1] [5] consists of a set of pluggable components. These components are the fundamental building blocks of the ARC services and clients. ARC services are running inside a container called the Hosting Environment Daemon (HED). There are four kinds of pluggable components having 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.

To deliver the non-trivial quality of services required by the Grid, there are a number of services running inside the HED. For example, Grid job execution and management is handled by the A-REX service [17], policy decision is the task for the Charon service, the ISIS service is responsible for the information indexing, batch jobs submission is handled by the Sched service etc. In the later sections our discussion will focus on the architecture and the design of another major set of services, i.e., the ARC storage.

IV. The ARC Storage System

The ARC 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[2]. 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.

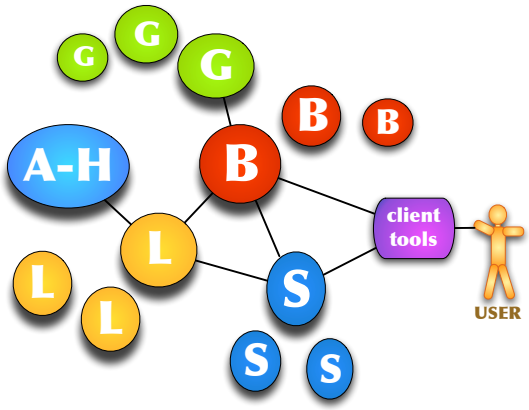

Fig. 1. Schematic of the ARC Storage architecture. The figure shows the main services of ARC Storage and how they communicate. See text for further details.

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

The system supports file transfer through several transfer protocols, with client side tools hiding technical details, such as specifying protocols, port numbers and so forth. To enable for fault-tolerance, the system implements automatic file replication, where the replicas of a file[3] are always stored on separate storage nodes[4]. To ensure a resilient, self-healing system, the Shepherd regularly sends heartbeats to one of the Librarians. If the Shepherd fails to send a heartbeat one of the Librarians will automatically initiate re-replication of the replicas between the other Shepherds.

In the default implementation the services will provide a full-featured and consistent data storage system using files as an atomic unit. The scope of the ARC data management system in the foreseen future is restricted to provide

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

[3] In this paper, a file denotes an entry in the global namespace, while replica denotes the physical file stored on a storage node.

[4] A storage node is a server with a Shepherd, a Shepherd backend and some storage service.

support for file-based data services.

The ARC Storage supports third-party storage services in two ways. Using the Gateway service, files already stored in some third-party storage can be accessed using the client tools provided by the ARC Storage. To make use of third-party services for storing new files, with the replication and fault-tolerance offered by the ARC Storage, third-party storage elements can also be used as backends for the Shepherd service.

External grid middleware components can access the ARC storage system by adapting and using ARC data service interfaces directly. ARC will provide interface components that communicate via standard protocols like SRM, which will give a single access point to the whole system.

V. Architecture of the ARC Storage

In a service oriented architecture the role of each service is well defined. Available objects[5] in the system are identified by unique global IDs. These IDs are categorized in three different categories according to the object type.

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

Each object in the ARC 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 ARC Storage has a third type of entries 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 ARC Storage, all replicas has 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 wrong checksum, or the Shepherd claims it has no such replica), '**creating**' (if the replica is in the state of uploading).

The details of the architecture are divided into two parts: First we will discuss the details of the core components, second we will discuss some of the important features provided by the system.

A. Core Components

A.1 Bartender

The Bartender service 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 accomplish 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.

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

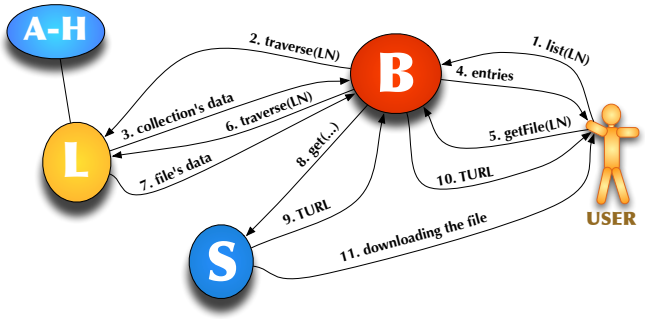

Fig. 2. Schematic showing the scenario of file download from the ARC Storage.

There could be any number of independent Bartender services running in the system, providing high-availability and load-balancing.

A.2 Librarian

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 change replica states automatically if needed. The Librarian uses the A-Hash for consistent storing. 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.

A.3 Shepherd

The Shepherd services run 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 goes directly between the Shepherd 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.

A.4 A-Hash

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, its absolutely crucial for the ARC Storage that the A-Hash is consistent. The distribution and replication of this service is therefore both challenging and needed.

B. Features

B.1 Heartbeat monitoring

In the proposed architecture, each Shepherd periodically send 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 invalidate all the replicas in that Shepherd.

B.2 Replication

Shepherds periodically ask the Librarian if the file of the replica stored on its storage node have 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 Shepherds which gets the new replica notifies the Librarian that the replica is alive. The Librarian then add this to the corresponding file.

B.3 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*, [11]. 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.

B.4 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 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, initiate file transfer from a Shepherd having the replica, and returns the TURL to the Shepherd with the invalid replica. When the Shepherd has recieved the replica and compared the checksum, it notifies the Librarian that the replica is alive again.
- In the case of a Shepherd becoming offline, a Librarian will as mentioned 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 VI.

B.5 Client tools

Being the only part a user will (and should) see from a storage system, the client tools is an important part of the ARC 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 CLI does assume a relatively high level of computer experience from the user. However, the CLI, being a stand-alone tool, can be used to access ARC Storage from any computer[6] that has network access and a Python installation.
- The **FUSE module** provides a high-level access to the ARC Storage. Filesystem in Userspace (FUSE) provides a simple library and a kernel-userspace interface. Using FUSE and the ARC Storage Python interface, the FUSE module allows users to mount the ARC Storage namespace into the local namespace. This way, the user can use his/her favorite file manager to access his/her files and collections. The FUSE module provides most of the features provided by the CLI, with the exception of modifying some non-posix metadata.

Its worth mentioning that the client tools access 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.

B.6 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.
- Develop a protocol oriented service i.e., all the external storage managers which support a certain protocol should be handled using the corresponding gateway service.

This approach provides the flexibility of avoiding multiple Gateway services for different storage managers. Currently the available Gateway service is based on the gridftp protocol.

When the user requests comes, the Librarian provides the metadata related with the request to the Bartender. Requests can be related to files, collections or external mount-points. In case of creating external mount-points,

[6] Also including Microsoft Windows PCs.

the Bartender contacts the Librarian to store the mount-point for later use. 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 send the Transfer URL (TURL) to the client via the Bartender. Using this TURL client can directly get the file from external store.

C. Security Model

As is the case for all openly accessible web services, the security model is of crucial importance for the ARC 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** takes care of the internal communication between services. There is a lot of communication between the services of the ARC Storage. The Bartenders send requests to the Librarians and the Shepherds, the Shepherds communicate with the Librarians, the Librarians talk with the A-Hash. If any of these services get 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) which the client can connect to. In the most 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 compromitation 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, offering a fine-grained security in the system.

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

VI. Testing and Discussion

Even though the ARC 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.

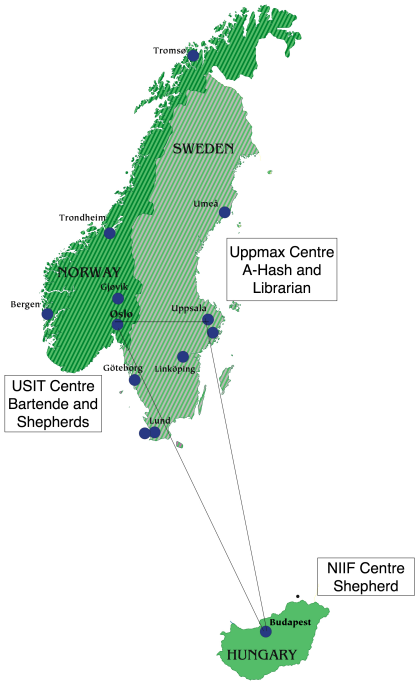

Fig. 3. The map shows the geographical distribution of the services in the test setup.

The services were distributed as shown in figure 3:

- An A-Hash runs at UPPMAX.
- A Bartender runs at USIT.
- A Librarian runs on UPPMAX.
- In total we are running five Shepherds: Three at USIT, having 100GB storage space each, one at UPPMAX, with 20GB, and one at NIIF, providing 16GB of storage space.

The following test where considered:

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

All the tests were performed and worked as expected. However, some of the tests deserve extra attention.

Figure 4 illustrate two different scenarios, corresponding to Test 2 and Test 3: The orange 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 bars show distribution with the same kind of uploading, but with the clients requesting three replicas each. The difference here is that the system simultaneously has

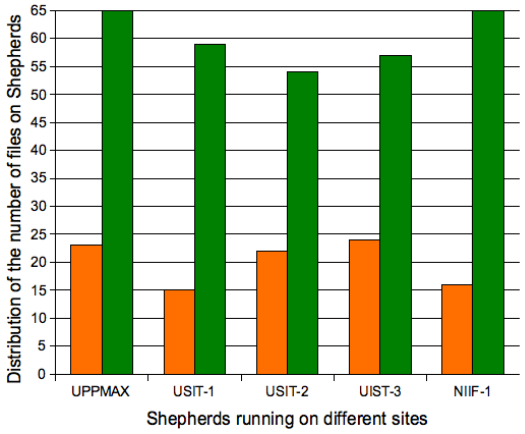

Fig. 4. The distribution of replicas over the geographically distributed storage nodes. Green bars shows the distribution in the case where we upload 100 files requesting 3 replicas where as orange bars show the distribution after uploading 100 files without replication.

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) is 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 figure 4.

	UPPMAX	USIT-1	USIT-2	USIT-3	NIIF
Load(MB)	6683	7638	7650	4773	2289

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

Table I shows the disk usage after Tests 1, 2 and 3. Worth noticing here is that the bandwidth NIIF was significantly lower than between UPPMAX and USIT. If the bandwith is saturated and the storage node is busy with, 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 (MB)	During (MB)	After (MB)
UPPMAX	4888	8799	8798
USIT-1	7820	9778	6843
USIT-2	6843	-	4888
USIT-3	7820	9774	8798
NIIF	3320	2344	1367
Sum	30691	30693	30694

TABLE II
Storage load before, during and after a Shepherd outage.

A significant feature of the ARC Storage is its automatic self healing. Test 4 addresses this feature by studying the effect of taking out one Shepherd from the system, and later reinsert it. Table II 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. Interestingly, the storage load automatically evens out 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. Interestingly, the NIIF node actually got fewer files when USIT-2 went offline. If two Shepherds simultaniously starts uploading 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.

VII. Conclusion and Future Work

The proposed system is still in an early phase of development, but our test results depict that the architecture is strong 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 the scientific community.

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

- **Security:** This is still under development. The design is more or less ready, but it still needs to be implemented and properly tested.
- **The A-Hash**: This is currently centralized. To avoid a singel 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.

Even though we have more work in front of us, we have shown that the ARC 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.

VIII. Acknowledgements

We wish to thank Mattias Ellert for helpful discussions and guidance on the ARC middleware. In addition, we like to thank UPPMAX, NIIF and USIT for providing resources for running the storage tests.

References

[1] ARC: http//:www.nordugrid.org/.
[2] dCache:http://www.dcache.org.
[3] FermiLab:http://www.fnal.gov.
[4] NDGF:http://www.ndgf.org/ndgfweb/home.html.
[5] KnowARC :http://www.knowarc.eu/.
[6] Chaitanya Baru, Reagan Moore, Arcot Rajasekar, and Michael Wan. The sdsc storage resource broker. In *CASCON '98: Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research*, page 5. IBM Press, 1998.
[7] G Behrmann, P Fuhrmann, M Grnager, and J Kleist. A distributed storage system with dcache. *Journal of Physics: Conference Series 119 (2008) 062014*.
[8] Chuck Boeheim, Andy Hanushevsky, David Leith, Randy Melen, Richard Mount, Teela Pulliam, and Bill Weeks. Scalla: Scalable cluster architecture for low latency access using xrootd and olbd servers. Technical report, Stanford Linear Accelerator Center, 2006.
[9] Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, pages 335–350, Berkeley, CA, USA, 2006. USENIX Association.
[10] D. Cameron, M. Ellert, J. Jönemo, A. Konstantinov, I. Marton, B. Mohn, J. K. Nilsen, M. NÓrden, W. Qiang, G. Roczei, F. Szalai, and A. Wäänänen. The hosting environment of the advanced resource connector middleware. Technical report, Nordugrid.
[11] Pedro Celis and John Franco. The analysis of hashing with lazy deletions. *Inf. Sci.*, 62(1-2):13–26, 1992.
[12] Tushar D. Chandra, Robert Griesemer, and Joshua Redstone. Paxos made live: an engineering perspective. In *PODC '07: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 398–407, New York, NY, USA, 2007. ACM.
[13] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, pages 205–218, Berkeley, CA, USA, 2006. USENIX Association.
[14] Yuhui Deng and Frank Wang. A heterogeneous storage grid enabled by grid service. *SIGOPS Oper. Syst. Rev.*, 41(1):7–13, 2007.
[15] Patrick Fuhrmann. dCache, the commodity cache. For dCache team.
[16] Wolfgang Hoschek, Javier Jaen-martinez, Asad Samar, Heinz Stockinger, and Kurt Stockinger. Data management in an international data grid project. pages 77–90. Springer-Verlag, 2000.
[17] A. Konstantinov. The hosting environment of the advanced resource connector middleware. Technical report, Nordugrid.
[18] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.
[19] Z. Nagy, J.K. Nilsen, and S. Toor. Documentation of the arc storage system. Technical report, Nordugrid.
[20] Michael Wan, Arcot Rajasekar, Reagan Moore, and Phil Andrews. 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)*, page 20, Washington, DC, USA, 2003. IEEE Computer Society.