Towards the Integration of StoRM on Amazon Simple Storage Service (S3)

Sergio Andreozzi[1], **Luca Magnoni**[1], **Riccardo Zappi**[1]
[1] INFN-CNAF, Viale Carlo Berti Pichat, 6/2, 40127, Bologna, Italy
E-mail: {sergio.andreozzi, luca.magnoni, riccardo.zappi}@cnaf.infn.it

Abstract. In Grid systems, a core resource being shared among geographically-dispersed communities of users is the storage. For this resource, a standard interface specification (Storage Resource Management or SRM) was defined and is being evolved in the context of the Open Grid Forum. By implementing this interface, all storage resources part of a Grid could be managed in an homogenous fashion. In this work, we consider the extension of StoRM (STOrage Resource Manager, an implementation of SRM v2.2) in order to integrate a new type of storage resource: the Amazon Simple Storage Service (Amazon S3). Amazon S3 is a simple Web services interface offering access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of Web sites. By performing this integration, we offer to the Grid community the capability to manage and access an incredible amount of storage resources freeing them from considering the costs associated with server maintenance, or whether they have enough storage available. The characteristics of StoRM are suitable for a smooth integration with Amazon S3. In particular, StoRM is designed to be easily adapted to the underlying storage resource via a plug-in mechanism, therefore a new plugin for integration with the Amazon S3 Web Service will be written. As regards the access policies, StoRM translates the Grid authorization rules into the Amazon S3 ones and applies them to the Amazon Web Services identity.

1. Introduction

In Grid systems, a core resource being shared among geographically-dispersed communities of users is the storage. For this resource, a standard interface specification (Storage Resource Management or SRM) was defined and is being evolved in the context of the Open Grid Forum. All storage resources part of a Grid can be managed in an homogenous fashion if they are exposed via an abstraction layer implementing the SRM specification.

In this work, we consider the extension of StoRM (STOrage Resource Manager, an implementation of the SRM interface v2.2) in order to integrate a new type of storage resource: the Amazon Simple Storage Service (S3). Amazon S3 is a novel storage utility that aims to provide data storage as a low-cost, highly available service using a pay-per-use billing model.

By performing this integration, we offer to the Grid community the capability to manage and access an incredible amount of storage resources freeing them from considering the costs associated with server maintenance, or whether they have enough storage available.

2. Storage resources in Grid: a new approach

In Grid systems, a core resource being shared among geographically-dispersed communities of users is the storage. For this resource, a standard interface specification (the Storage Resource Management or SRM) was defined and is being evolved in the context of the Open Grid Forum Grid Storage Management working group. By implementing this interface, all storage resources part of a Grid could be managed in an homogeneous fashion. In this work, we consider the extension of StoRM (STOrage Resource Manager, an implementation of SRM v2.2) [1, 2] in order to integrate a new type of storage resource: the Amazon Simple Storage Service (Amazon S3) [3]. Amazon S3 is a simple Web services interface offering access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of Web sites. By performing this integration, we offer to the Grid community the capability to manage and access an incredible amount of storage resources freeing them from considering the costs associated with server maintenance, or whether they have enough storage available. The characteristics of StoRM are suitable for a smooth integration with Amazon S3. In particular, StoRM is designed to be easily adapted to the underlying storage resource via a plug-in mechanism, therefore a new plugin for integration with the Amazon S3 Web Service will be written. As regards the access policies, StoRM translates the Grid authorization rules into the Amazon S3 ones and applies them to the Amazon Web Services identity.

3. Storage Resource Manager service and StoRM

When dealing with storage, the main problem facing the scientist today is the need to interact with a variety of storage systems providing different interfaces and security mechanisms. The goal is to present to scientists the same interface regardless of systems being used, providing transparent storage management functionalities.

This is the main goal of a common storage service interface, that has to be flexible enough to be used to access storage systems based on different storage technologies. At the same time, its usage should be simple enough so that clients do not need to have any knowledge of the underlying storage back-end.

3.1. Storage Resource Manager (SRM)

The concept of Storage Resource Managers (SRMs) services was devised by the Open Grid Forum Grid Storage Management (OGF-GSM) working group to meet the requirement above. The OGF-GSM defines the standard SRM interface [4] to hide storage dependent characteristics and to allow interoperability between different storage systems. Storage resource manager services are nowadays a building-block of the storage requirements in the grid computing models, where heterogeneous storage resources coexist and the use of a standard interface is fundamental to gain real interoperability.

The SRM interface defines a set of functionalities to manage the dynamic use of space and files in a storage resource on the Grid. From one side a client can request to the SRM service to reserve space and to manage files and directories, from the other side an SRM can sometimes dynamically decide which files to keep in the storage space and which files to remove (e.g. when free space is needed). Therefore, files are no longer permanent entities on the storage, but dynamical ones that can appear or disappear according to the user specification. A detailed description of the SRM service and its specifications is given in references [5, 6].

3.2. Data naming in SRM

In a Grid environment the naming capability allows users to refer to specific data resources in a physical storage system using a high level logical identifier. There are several terms in a Grid environment to identify data, depending on the different resources involved. The *Logical File Name* (LFN) is a user friendly (and user defined) high level identifier, it is organized in a file

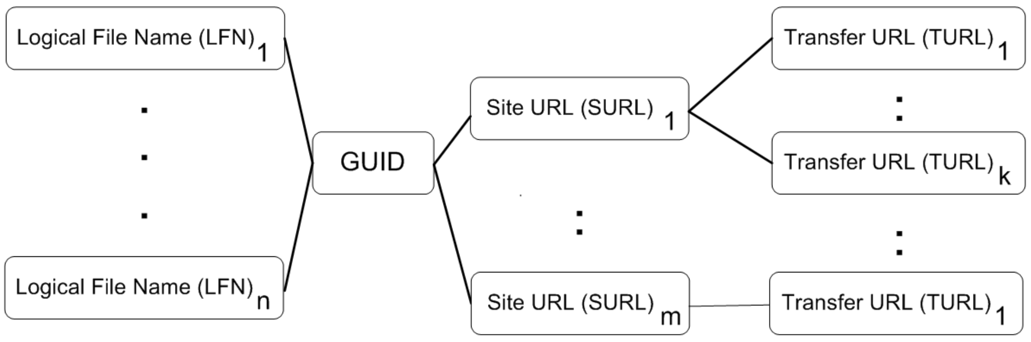

Figure 1. Relation between LFN, SURL and TURL.

system like structure, and there can be several LFNs for one file, similar to the concept of alias. A *Site URL* (SURL) is a logical identifier that provides an abstraction of the file namespace to address a file in a specific SRM service. The *Site URL* (SURL) points to the physical stored version of a file on a grid storage element. The *Transport URL* (TURL), or Transfer URL, is physical identifier provided by SRM services for data access. It depends on the access protocol specified by client (e.g. *GridFTP* or ***HTTP***, the one used for S3 integration) and on the physical location of data in the storage system. A specific file can be brought from tape to disk to provide direct access to the client. Finally the *file catalog* is the grid middleware service which maps LFNs to SURLs, i.e. tracks the real physical locations of the logical file names. Figure 1 depicts the relationships among the terms.

The structure of a SURL is:
`srm://host[:port]/[soap_end_point_path?SFN=]site_file_name`. A SURL can be expressed in normal form or in query form, depending on the presence or absence of the service endpoint path. For the integration with S3 we focus on SURLs in normal form. It is composed of the *prefix*, that is the `srm` string for all SRM services, followed by the *authority*, namely the service `hostname` and the optional `port`, and by the `Storage File Name` (stFN). The stFN is a logical identifier for the data, it has a file system like structure that can be different from the real location where the data is stored.

The structure of TURL is `prefix://[host[:port]]/physical_file_name`, it strongly depends on the transfer protocol the client specified, for example *gsiftp* or *http*. The *Physical File Name* (PFN) represents the real position of the data on the storage resources, depending on the access protocol.

To access a file through a SRM service, a SURL is provided as a parameter embedded in the specific SRM request. The basic method to interact with a SRM service to read and write data are the *srmPrepareToPut* and the *srmPrepareToGet* functionalities. The srmPrepareToPut is used by the client to write a new file onto storage, on this request the SRM service takes care of prepare the environment for user access, reserve space, performs garbage collection if needed and set appropriate permissions on data. An appropriate TURL built with user specified parameters is returned to client, for the write operation. The srmPrepareToGet is used by users to access data, the SRM service performs the necessary operations to make the file available for user access, for example staging the file from tape to disk and setting its permission. An appropriate TURL is returned to the client for the read operation. The SRM interface provides two classes of methods: asynchronous and synchronous ones. Asynchronous methods return a token corresponding to the request, and the corresponding action is performed by the SRM asynchronously. The client can retrieve at any time the status of the request by addressing it through such a token. This is the case for data management functionalities. Synchronous requests return at completion giving back the control to the client (*blocking calls*). This is the case of directory and file management (e.g. *srmLs*, *srmMkdir*, *srmRm* and *srmRmdir*) and space management functions (e.g. synchronous *srmReserveSpace*).

3.3. StoRM

Many SRM implementations exists, the work proposed in this article regards the Grid Storage Resource Manager StoRM [1]. StoRM has been developed with the specific aim of providing access to parallel file systems like GPFS and Lustre, but also standard POSIX file systems, from the Grid through a SRM interface. Moreover, StoRM is distinguished from the other SRM implementations because it uses a driver mechanism to bind to various file system. In this way, plugging into StoRM a new driver allows to support a new storage system or file system.

The StoRM project has been developed by a collaboration between the "Istituto Nazionale di Fisica Nucleare (INFN)" and the "the Abdus Salam International Centre for Theoretical Physics (ICTP)" in Trieste, the latter operating in the framework of the EGRID project.

The main idea behind StoRM [1, 2] is to create an SRM implementation to leverage on the advantages of high performance parallel file-systems in a Grid environment. It provides users and applications with the standard SRM v2.2 functionalities combined with the capability to perform secure and local accesses (the *file://* transfer protocol is supported) to the shared storage resources.

StoRM decouples the file system service from the SRM service, in the sense that the data-access functionality can be provided by any POSIX file system, while StoRM specifically provides just the SRM functionality. In the case that any advanced functionality peculiar to a given file system were available via specific APIs, they could be included into a StoRM driver bound to the file system and plugged in into StoRM. StoRM can be easily configured to run on the GPFS or Lustre parallel file systems, which provide high-performance POSIX access to the data and are widely employed commercial products. StoRM provides the advanced SRM functionalities to dynamically manage space and files according to the user requirements, to specify the desired lifetime and to allow for advance space reservation and a different quality of service than provided by the underlying storage system. Since it generalizes the file system access to the Grid, StoRM also takes advantage from the security mechanisms of the underlying file system to ensure an authenticated and authorized access to the data.

StoRM has a multi-tier architecture. The Front-End (FE) component exposes the service interface and manages client authentication. The Back-End (BE) component is the core of StoRM, it processes the SRM requests managing metadata and authorization aspects and interacting with other Grid services. It supports different file systems through a driver mechanism easy to expand and improve. Each component can be instanced and replicated on dedicated host, to satisfy the scalability and availability requirements. In the next section we present how the system can interact with the S3 storage.

4. Amazon S3

The Amazon Simple Storage Service (S3) is part of the Amazon Web Services (AWS) suite [7, 8], a set of services enabling Amazon to expose resources part of its infrastructure for easy integration into external applications. In the current view of the definition of Grid systems [9], such an approach falls into the category of data center Grids, where a single organization is involved in the complete dynamic life cycle of service deployment, provisioning, management and decommissioning as part of its normal operation.

Amazon S3 allows users to store any type of data at a cost of 15 cents per gigabyte per month, with bandwidth priced at between 10 and 18 cents per gigabyte transferred outside Amazon's network. Amazon S3 is intentionally built with a minimal feature set. S3 functionalities include write, read, and delete data objects pointed to a unique, developer-assigned key. Moreover, a

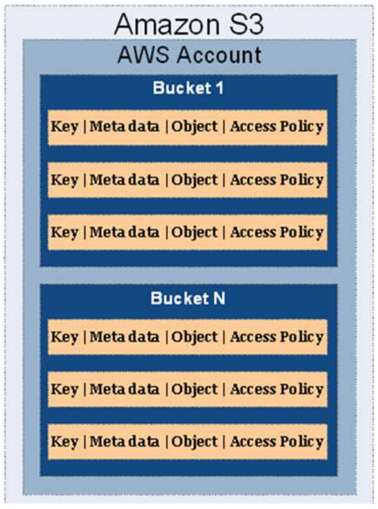
Amazon S3
AWS Account
Bucket 1
Key | Meta data | Object | Access Policy
Key | Meta data | Object | Access Policy
Key | Meta data | Object | Access Policy
Bucket N
Key | Meta data | Object | Access Policy
Key | Meta data | Object | Access Policy
Key | Meta data | Object | Access Policy

Figure 2. Schematic representation of objects and buckets concept in S3.

simple and effective authentication mechanism is provided to ensure that data is kept secure from unauthorized access. Objects can be made private or public, and rights can be granted to specific users. An S3 user can authorize other users to access his data. More information can be found in reference [7].

In Amazon S3, data can be organized in a two-level namespace, as shown in Fig. 2: S3 "objects" and S3 "buckets":

S3 Object An object is a labeled piece of data with some accompanying metadata. The object is composed of four parts:

(i) A name (S3 calls it the "key").
(ii) the "value": data stored in that object, that is any sequence of bytes up to 5 GB.
(iii) A reference to the parent bucket. An object always belongs to a bucket.
(iv) A set of metadata key-value pairs associated with the object; metadata can be divided in two categories: user and system, the latter being a small number of predefined HTTP metadata entries (e.g., Last-Modified).

S3 Bucket A bucket is a named container for objects. It represents the top-level namespace and the bucket identifier is globally unique; each S3 user is limited to 100 buckets and each bucket can contain an unlimited number of objects; nevertheless buckets can contain no further sub-buckets, so if a user wants a directory structure inside a bucket, he needs to simulate one by giving the objects names like "directory/subdirectory/file-object". Buckets serve several purposes: they are used to identify the corresponding accounts responsible for storage and data transfer charges, they play an important role in access control and they serve as the unit of aggregation for usage statistics reports.

4.1. Interface Styles

The Amazon S3 service is exposed through two different interface styles: `SOAP` and `REST`. The SOAP-based RPC-style API is described in WSDL and relies on the SOAP protocol for the communication. The HTTP-based REST-style API uses standard HTTP operations and follows the REST architectural style [10]. Within this approach, the operations are the HTTP operations and the exposed objects are identified by URIs.

The default download protocol is HTTP, but the BitTorrent(TM) protocol can also be used. In the remaining part of this section, we will focus on the REST-style API. Each URI identifying

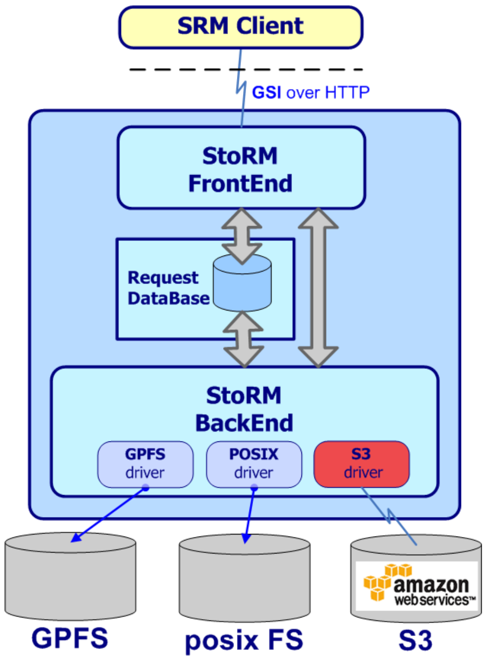

Figure 3. StoRM architecture with the AWS S3 integrated through a specific driver.

a resource, is not part of the resource representation, but can be constructed following a set of rules.

The REST-style S3 API provides three types of resources. They are presented here together with sample URLs:

- The list of buckets (https://s3.amazonaws.com/): there is only one resource of this type per user
- A particular bucket (https://s3.amazonaws.com/name-of-bucket/): there can be up to 100 resources of this type per user
- A particular S3 object inside a bucket (https://s3.amazonaws.com/name-of-bucket/name-of-object): there can be infinitely many resources of this type, in principle

4.2. Request Signing and Access Control

As regards authentication and access control, Amazon S3 considers two things when deciding whether or not the request should be allowed to proceed: the first is who is making the request; the second is the access control policy (ACL) on the bucket and object targeted by the request.

Authentication is based on user identity. When users register with Amazon's Web Services, they are assigned an identity, the "AWS Access Key ID". To prevent attacks based on creating a large number of identities by the same physical user (whitewashing attack), identities are linked to user credit cards. Together with their identity, users are assigned a private key: the "AWS Secret Access Key" generated by Amazon during registration. In this way the S3 clients are authenticated using a public/private key scheme and keyed-hash message authentication code (HMAC). Every performed request must be cryptographically signed with the user private key so that Amazon can identify the requestor (note that the "private key" is not totally private, since Amazon knows it).

The access control is specified using access control lists (ACL) at the granularity of buckets or objects. Each ACL can specify the access attributes for up to 100 identities. The access

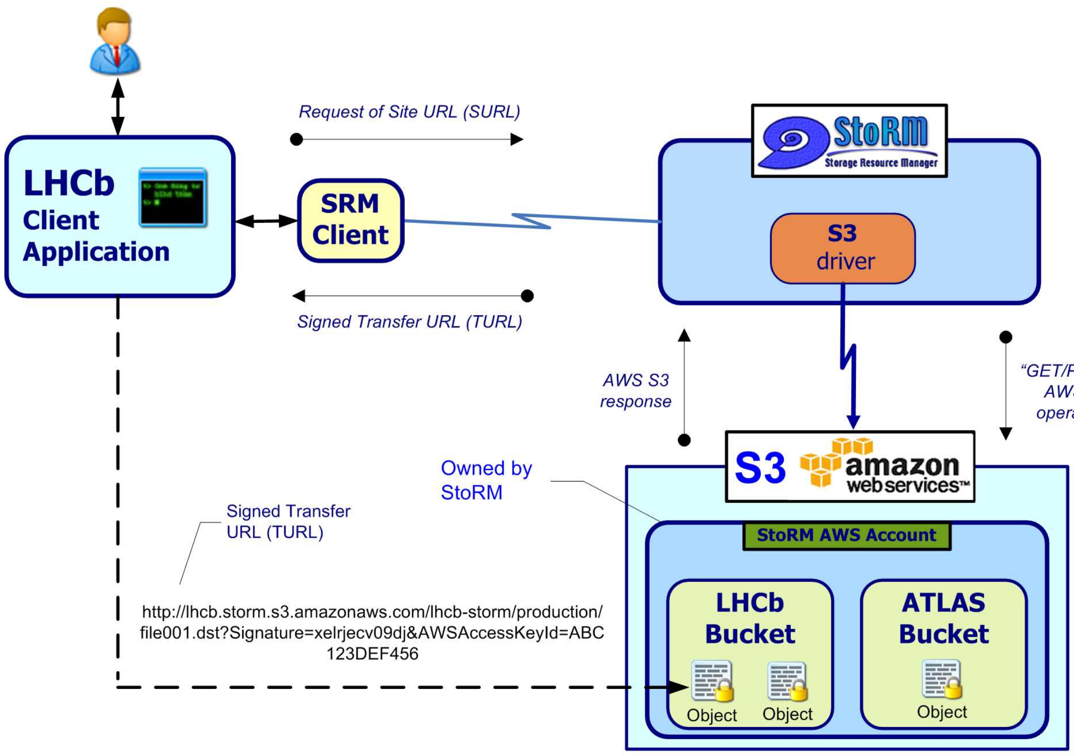
LHCb Client Application
SRM Client
Request of Site URL (SURL)
Signed Transfer URL (TURL)
StoRM Storage Resource Manager
S3 driver
AWS S3 response
"GET/PUT/… " AWS S3 operations
Owned by StoRM
Signed Transfer URL (TURL)
http://lhcb.storm.s3.amazonaws.com/lhcb-storm/production/file001.dst?Signature=xelrjecv09dj&AWSAccessKeyId=ABC123DEF456
S3 amazon webservices™
StoRM AWS Account
LHCb Bucket
ATLAS Bucket
Object
Object
Object

Figure 4. Sequence of operations from the client SRM request to the data access on the Amazon Simple Storage System (S3).

control attributes supported are:

- **fullcontrol**: the owner preserves full control over the object, having all permissions below;
- **read**: for buckets or objects, the user can read the object;
- **write**: for buckets only, the user can create or delete objects in the bucket;
- **readacl**: for buckets or objects, the user can read the ACL and the identity of the owner. Implicit for the owner;
- **writeacl**: the user can change the ACL; this is similar to full user control since the user can assign any right.

5. StoRM on Amazon S3

The characteristics of StoRM are suitable for a smooth integration with Amazon S3. In particular, StoRM is designed to be easily adapted to the underlying storage resource via a plug-in mechanism based on file system drivers, as shown in Fig. 3 StoRM translates the Grid authorization rules into the Amazon S3 ones and applies them to the Amazon Web Services identity.

StoRM interacts with file systems through a specific driver implementing a common internal interface to hide the storage dependent characteristics. The backend logic is decoupled from the specific file system in use, allowing a StoRM instance to work on different file-systems at the same time. At present there are two specific drivers for the GPFS and XFS file-systems that rely on proprietary APIs for the management of advanced functionalities, and a generic

POSIX driver which works for every file-system with POSIX semantics. A specific plug-in for the integration with the Amazon S3 Web Service will be written using the S3 API.

Every object stored in Amazon S3 is contained in a bucket. In the Worldwide LCG Grid Computing [11] use case StoRM creates a bucket for each Virtual Organization involved in the data access process, a use case being shown in Fig. 4. Within each bucket, StoRM replicates the user defined namespace to address the desired object. A specific key is assigned to each object for later retrieval. Amazon S3 decides which principal is making a request based on the authentication information provided with the request.

For each Virtual Organization (VO) StoRM uses at least two AWS accounts. An AWS account with write privilege will be used to build transfer URL able to create or delete objects and an unprivileged reader account for reading operations. Multiple accounts for write and read operations can be associated to VOs to allow more complex access policy; for example, AWS accounts can be related with VOMS attributes.

Each bucket is owned by the StoRM AWS account (identified by AWS Access Key ID). For each SRM data request, StoRM adds an ACL entry corresponding to the VO AWS account that want to store or access each file. StoRM builds the request including evidence that the principal knows the AWS Secret Access Key corresponding to the specified Access Key ID, for example by using the secret key to sign the request. The client can then use the URL returned by StoRM to perform the desired operation on data. When the request time to live expires, or when the client declares the end of data access operations (with dedicated SRM functionalities, as the *srmPutDone* or the *srmReleaseFile*), StoRM removes the ACL entry previously added. With this approach each Virtual Organization can relies on the accounting and billing capabilities provided by Amazon S3.

6. Conclusion

The main focus of this paper was to discuss how StoRM (STOrage Resource Manager), an implementation of SRM v2.2, can be extended to expose storage resources offered by the Amazon Simple Storage Service (S3) using a Grid standard interface.

We have shown how the modular design of StoRM based on a plug-in mechanism allows to easily integrate new types of underlying resources. The REST-style S3 API was selected to build such a plug-in and translation of the Grid authorization rules into the Amazon S3 ones were discussed.

By this work, we offer to Grid systems the capability of accessing a novel storage utility such as Amazon S3 without any impact on the service consumer side. Future work will be targeted at the development and testing of the extension presented in this paper.

References

[1] Corso E, Cozzini S, Donno F, Ghiselli A, Magnoni L, Mazzucato M, Murri R, Ricci P, Stockinger H, Terpin A, Vagnoni V and Zappi R 2006 StoRM, an SRM Implementation for LHC Analysis Farms, Computing in High Energy Physics http://indico.cern.ch/contributionDisplay.py?contribId=373&sessionId=13&confId=048

[2] Corso E, Cozzini S, Forti A, Ghiselli A, Magnoni L, Messina A, Nobile A, Terpin A, Vagnoni V and Zappi R 2006 StoRM: A SRM Solution on Disk Based Storage System URL `http://www.cyfronet.pl/cgw06/`

[3] Simple Storage Service (S3) A http://aws.amazon.com/s3

[4] 2007 The Storage Resource Manager Interface Specification, Version 2.2 `http://sdm.lbl.gov/srm-wg/doc/SRM.v2.2.html`

[5] Resource Managament Working Group S http://sdm.lbl.gov/srm-wg/

[6] Shoshani A, Sim A and Gu J 2004 *Grid Resource Management* (Norwell, MA, USA: Kluwer Academic Publishers) chap 20, pp 321–340

[7] Garfinkel S L 2007 An Evaluation of Amazons Grid Computing Services: EC2, S3 and SQS Tech. Rep. TR-08-07 Center for Research on Computation and Society, School for Engineering and Applied Sciences, Harvard University, Cambridge, MA

[8] Onibokun A and Palankar M 2006 Amazon S3: Black-Box Performance Evaluation Tech. rep. Computer Science and Engineering, University of South Florida
[9] C Smith D S 2007 Technical Strategy for the Open Grid Forum 2007-2010
[10] Fielding R T 2000 *Architectural styles and the design of network-based software architectures* Ph.D. thesis URL `http://portal.acm.org/citation.cfm?id=932295`
[11] Worldwide LHC Computing Grid 2007 http://lcg.web.cern.ch/lcg