Workshop on Interfaces and Abstractions for Scientific Data Storage (IASDS10), co-located with IEEE Int. Conference on Cluster Computing 2010 (Cluster10), Heraklion, Greece, September 2010.

pWalrus: Towards Better Integration of Parallel File Systems into Cloud Storage

Yoshihisa Abe and Garth Gibson
Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA, USA
{*yoshiabe, garth*}*@cs.cmu.edu*

***Abstract*—Amazon S3-style storage is an attractive option for clouds that provides data access over HTTP/HTTPS. At the same time, parallel file systems are an essential component in privately owned clusters that enable highly scalable data-intensive computing. In this work, we take advantage of both of those storage options, and propose pWalrus, a storage service layer that integrates parallel file systems effectively into cloud storage. Essentially, it exposes the mapping between S3 objects and backing files stored in an underlying parallel file system, and allows users to selectively use the S3 interface and direct access to the files. We describe the architecture of pWalrus, and present preliminary results showing its potential to exploit the performance and scalability of parallel file systems.**

***Keywords*-high performance computing; parallel and distributed file systems; cloud computing;**

I. Introduction

The Amazon S3 API [1] has gained popularity as the primary interface to public cloud storage. It provides cloud users with a convenient way of uploading and downloading data from persistent storage over HTTP/HTTPS. With a S3 storage service users have easy access to their data through this ubiquitous interface regardless of their personal computer's location. Once in the cloud, users' applications use the same HTTP interface to retrieve data from S3 before processing it. Finally, the same or other users outside the cloud can download results, or upload new data sets to be processed the same way. The simplicity and ubiquity of data access is a key advantage of S3 storage, while using other methods such as network or distributed file systems to transfer data usually involves administrative configuration for each end-user environment and personal computer.

Although the ubiquitous data accessibility provided by S3 services is also beneficial in private cloud computing environments, such as Eucalyptus [2], private cloud users are also likely to desire full use of the performance and capacity of purchased storage resources. In particular, parallel file systems, such as PVFS [3], PanFS [4], GPFS [5], and Lustre [6], play a fundamental role in cluster computing environments as storage capable of high performance and scalability. In order to accommodate applications with very high parallelism and performance, it is essential to ensure that they are able to exploit the scalability of such parallel file systems. If an S3 service is deployed in a cloud, it would occupy physical storage that could otherwise have been available to a parallel file system, reducing scalability as well as capacity. In addition, if the S3 service and parallel file system are separate storage options, users may end up duplicating their data in both, which leads to further inefficiency of storage capacity.

For this reason, private cloud administration demands good integration of the convenience and performance efficiency that S3 storage and parallel file systems provide, respectively. Table I summarizes characteristics of these two storage options. As the table shows, S3 storage and parallel file systems complement each other. S3 storage enhances the accessibility of data, making it available to users anywhere connected to the Internet, while data in a parallel file system is available only to users in environments properly configured and maintained by administrators to access it. On the other hand, parallel file systems are capable of handling high degrees of concurrency and allow direct and efficient reads and writes to partial files, while S3 hides it behind an interface that allows access only at the granularity of whole objects, even for access from the cloud computer nodes in the same data center. For the maximum usability of data storage in clouds, users should have the opportunity to selectively use those two types of access methods to their data.

In this paper, we propose pWalrus, a single storage model for cloud computing environments that takes advantage of the benefits of both S3-style storage and parallel file systems. In our model, an S3 service acts as a thin layer consisting of multiple servers, built on top of a parallel file system available in the underlying cluster environment. Users inside the cluster have two access paths to their data: S3 and direct access to the parallel file system. Thus, they may choose to use either interface for convenience or performance reasons. For example, if private cloud computing applications are written using the POSIX interface, they can obtain direct parallel file system access, while users outside the cloud still have the universal access to their data that S3 offers.

pWalrus is implemented as a modification to Walrus, the S3-compliant storage service of Eucalyptus [2]. pWalrus seeks to improve the original storage model of Walrus by (1) distributing workloads to multiple servers, (2) allowing direct file system access to data in addition to access over HTTP/HTTPS, (3) giving users the choice to read or write

Table I
CHARACTERISTICS OF S3 STORAGE AND PARALLEL FILE SYSTEMS

	Strengths	Restrictions
S3	- Facilitated access through a uniform interface - Universal accessibility regardless of user environments	- PUT/GET access to objects in their entirety
Parallel File Systems	- Scalable performance - POSIX interface with partial reads and writes	- Require administrative work to allow access

partial files, and (4) eliminating the process of copying S3 objects to file systems before working on them, which saves storage space as well as CPU time and networking bandwidth.

The remainder of the paper is organized as follows. In Section II, we summarize the background. We describe the design of pWalrus in Section III and show the benefits of providing a parallelized S3 service and allowing direct access to a parallel file system in Section IV. Finally, we summarize related work in Section V and conclude in Section VI.

II. BACKGROUND

A. Amazon Simple Storage Service

Amazon Simple Storage Service (Amazon S3) [1] is an Internet-based storage service with a simple set of functionalities that provides virtually unlimited storage space. Users periodically pay for the service based on the amount of data stored, data transferred, and access requests made. The main features of Amazon S3 are summarized below. We refer the reader to Amazon's online documentation of S3 for further details.

- Users create *buckets*, which are containers for data, and store in them *objects*, which are the base unit of user data.
- Users upload or download entire objects in a single operation, each up to 5 GB and with a unique key as its identifier, from their own buckets through REST and SOAP interfaces. The primary protocol for retrieving objects is HTTP.
- REST requests for accessing S3 buckets or objects are authenticated by Hash-based Message Authentication Code (HMAC) [7]. Each user has a pair, *Access Key ID* and *Secret Access Key*, and embeds the Access Key ID in each S3 request, appending the hash of the request content to it using the Secret Access Key to sign the hash. Upon receiving a request, the S3 service retrieves the Secret Access Key corresponding to the Access Key ID embedded in the request, computes the content hash using it, and compares the hash sent with the request against the computed hash. If the hashes match, the request is authenticated as that of the user who has been assigned the specified Access Key ID.
- Access control is done through an access control list (ACL) associated with each bucket and object. An ACL can specify types of access rights (READ, WRITE, READ_ACP, WRITE_ACP[1], FULL_CONTROL) granted to individual S3 users.
- Versioning is supported and can be enabled on a per-bucket basis. If versioning is enabled, there can be multiple instances of the same object identifier as the user overwrites it.
- The storage service is *eventually consistent*; users may not see the newest content of an object identifier immediately after it has been overwritten.

Using Amazon S3, user data can be stored in or nearby cloud computing environments such as Amazon Elastic Compute Cloud (EC2) [8].

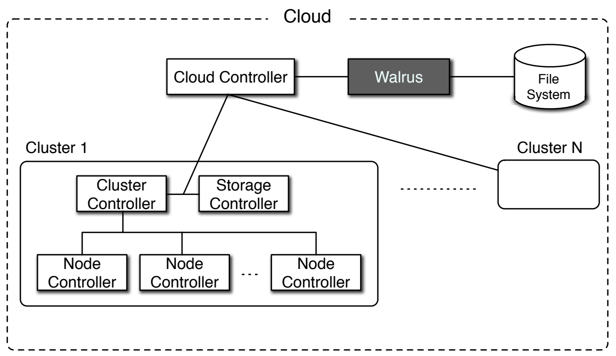

Figure 1. Eucalyptus architecture.

B. Eucalyptus and Walrus

Eucalyptus [2] is an open-source infrastructure for cloud computing. It follows the Amazon EC2 [8] model, in which users configure virtual machines in the cloud and run tasks on them. Figure 1 depicts the cloud model of Eucalyptus. The cloud controller is a top-level entity in charge of the entire cloud, managing resource allocation and user accounts, as well as providing a web interface for cloud management and EC2-compatible interfaces such as SOAP. The cluster controller performs per-cluster scheduling and networking, while the storage controller provides block store services similar to Amazon Elastic Block Service (EBS) [9]. The node controller controls the hypervisor on each compute node and sets up virtual machines on it.

[1]READ_ACP and WRITE_ACP are read and write permissions, respectively, for the access control list itself, as opposed to the bucket or object protected by it.

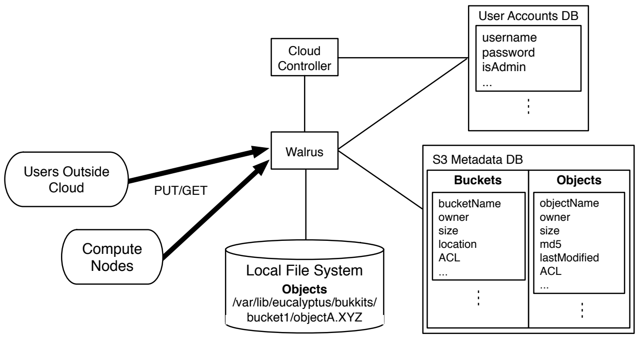

Figure 2. Architecture of Walrus (in Eucalyptus 1.6.2).

The S3 storage service implementation for a Eucalyptus cloud is called Walrus. It provides a bucket-based object store and exports interfaces compatible with Amazon S3. REST-based tools that work with Amazon S3, such as `s3curl` [10], are thus compatible with Walrus and work without modification. As of Eucalyptus 1.6.2, Walrus is a single instance per the entire cloud. As shown in Figure 2, it shares a Java-based database holding user account information with the cloud controller, and accesses metadata of S3 buckets and objects similarly stored in databases. The storage space for buckets and objects is mapped to a single, non-shared file system directory mounted on the machine on which Walrus runs. Both users outside the cloud and user applications running on compute nodes in the cloud upload and download data through the REST interface of Walrus.

III. pWalrus Design

A. Overview

pWalrus is structured as an array of S3 servers that are backed by the same parallel file system. All servers have the same view of S3 storage as user data. S3 system state is also stored in the shared file system. Thus, users can connect to any of the servers to access their data with the S3 interface, which creates the opportunity for load balancing using, for instance, DNS round-robin to map the URL for the S3 service to an IP address of a server. In addition, pWalrus allows for direct access to data in the parallel file system by exposing the mapping between S3 objects and the corresponding files used to store these objects, which in Walrus are managed exclusively and hidden by the S3 service.

Figure 3 illustrates the architecture of pWalrus. The solid arrows indicate the S3 interface to the storage service, and the dotted arrow direct access to the parallel file system. In order to make both interfaces available to users, pWalrus uses the following kinds of information in the parallel file system, most of which are stored in special files called ".walrus":

- User account information in a pWalrus management file.
- Proof by S3 users, in per-user special files, that they have access to the parallel file system as a certain cluster user.
- Metadata of S3 buckets in special files under each user's root S3 directory, which contains the directories backing buckets.
- Metadata of S3 objects in special files under each directory backing an S3 bucket.

In the pWalrus management file, pWalrus keeps information associated with each S3 user (which is identified by a Query ID, Eucalyptus' equivalent of Amazon S3's Access Key ID), including (1) authentication information like the Secret Key (Eucalyptus' equivalent of Amazon S3's Secret Access Key), (2) an associated cluster user account that is used to access the parallel file system and is specified by the user upon registration, and (3) the file system directory used to store buckets and objects for that user.

The per-user .walrus file is created by each S3 user[2] allowing direct file system access, and is referred to by pWalrus to confirm the validity of the claimed association between the S3 user and the cluster user account. After creating an account (Query ID) in pWalrus, each user stores his Query ID in the .walrus file under the home directory of his cluster user account. This .walrus file should be writable only by the user himself, and needs to be readable only by the user and the pWalrus service. pWalrus confirms the cluster user account association only if the Query ID stored in the .walrus file matches that of the S3 request passing the HMAC signature test. In this way, pWalrus effectively delegates the task of confirming the identity and authority of the user to cluster administration.

Finally, pWalrus uses other .walrus files in each bucket directory to store information about S3 objects that may not be supported by the parallel file system as file attributes. For example, access control lists used for S3 access and object content hashes (md5) are likely to be stored in .walrus files instead of in file attributes.

B. S3 Request Handling

pWalrus handles S3 requests as follows. First, upon receiving a request naming a specific Query ID, pWalrus looks up the corresponding Secret Key and uses it to confirm the validity of the S3 request by HMAC. Next, pWalrus retrieves from the pWalrus management file the cluster user account information corresponding to the Query ID embedded in the request, checks if that account has a .walrus file under the cluster user's home directory, and verifies that the contents of this .walrus file match the request's Query ID. If it does,

[2]Although users in our prototype author their per-user .walrus files themselves following a specific XML format, we recommend that .walrus files be prepared automatically through, for example, the web interface of Eucalyptus, to avoid formatting errors.

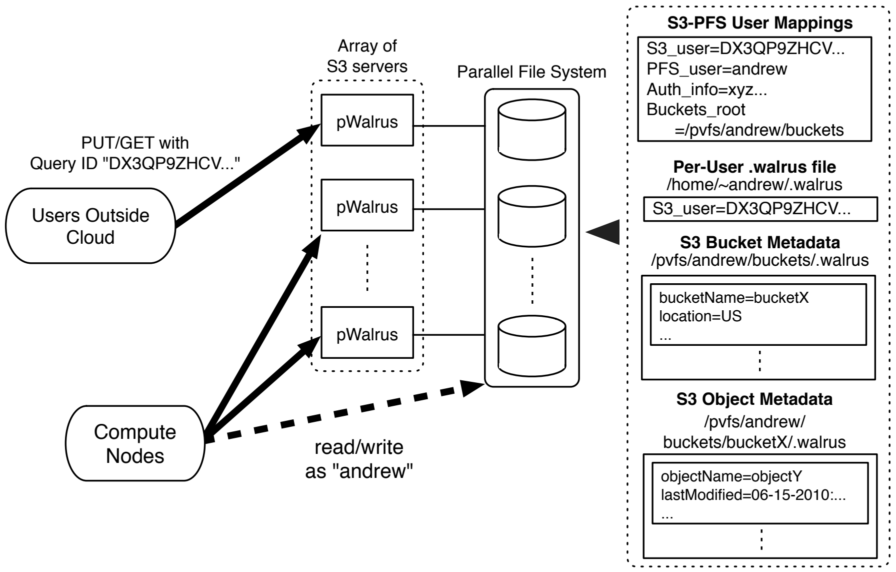

Figure 3. Architecture of pWalrus.

the request is authorized to access the parallel file system as the cluster user and pWalrus proceeds with the request.

If the request wants to store a new object, pWalrus creates a new file containing the request's data in the appropriate bucket directory and sets its owner to the corresponding cluster user. If the request asks to retrieve an existing object, pWalrus checks the access control lists of the bucket and object. If the requester has READ permission to both the bucket and object, the contents of the object's file are returned. Finally, if the request wants to read S3 metadata, such as a list of buckets owned by the user, pWalrus serves that request based on information contained in the appropriate .walrus files.

C. Reconciling S3 and Parallel File Systems

The storage model of pWalrus exposes to users two access paths to data and differences in the semantics of S3-style storage and the underlying parallel file system. As a result, two questions arise: (1) how we reconcile access semantics differences like atomic overwrites and concurrent seek-and-partial-write's, and (2) how we map file names to S3 object names, especially if versioning is supported. This is ongoing work[3]; we propose to address the former by providing a mechanism for users to explicitly expose (publish) their files to the S3 interface, and the latter by a file-naming convention.

To rephrase the first question, we need to ensure that S3 access to a certain file and direct access to that file in the parallel file system do not conflict with each other. Otherwise, it would be possible, for example, for pWalrus to overwrite an existing file (when versioning is not used) while the user is reading it directly through the file system. Our approach is to let the file system user have explicit control over what pWalrus may access. When a new file is created in the file system, it is not seen by pWalrus until the user executes a command to "export" it. Thus, he is free to read or write the file without worrying about possible access conflicts with pWalrus. Once he executes the command, the corresponding S3 metadata, including timestamp and content hash, is created in the appropriate .walrus file and pWalrus is allowed to access the file. If pWalrus reads a file whose timestamp or content hash has changed, it marks the corresponding object as corrupted by inappropriate external modification. If the user would like to update the file directly at a later time, he can explicitly "unexport" the object, removing the associated S3 metadata so that pWalrus no longer views the object as existing. There is less of a problem when a new file is created as a result of an S3 object overwrite request, on the other hand, because this always writes a new file. A newly written object's metadata is written by pWalrus so that the file has been "exported" upon the completion of the request.

The second challenge is defining the mapping between file and object names. A simple way is to use direct translation; a particular file name translates to the same object name available through the S3 interface. This approach is similar to FTP, which almost exposes the semantics of the underlying file system. However, if versioning is enabled in the S3 interface, there can be multiple files corresponding to the same object. We suggest that this kind of file-to-object mapping be done by a straightforward file-naming

[3]Our prototype, measured in Section IV, expects users to cope with semantic issues and employs a non-versioned direct mapping between file and object names.

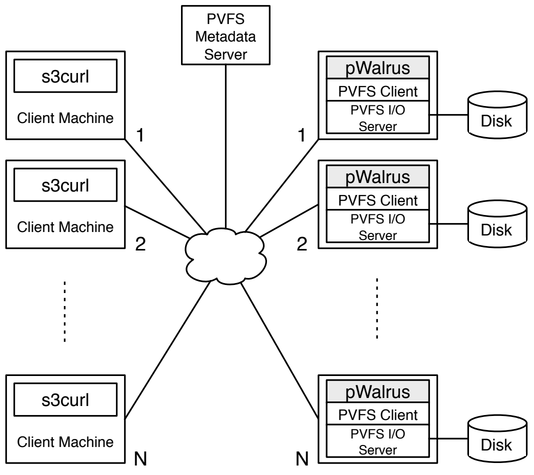

Figure 4. Experimental set-up.

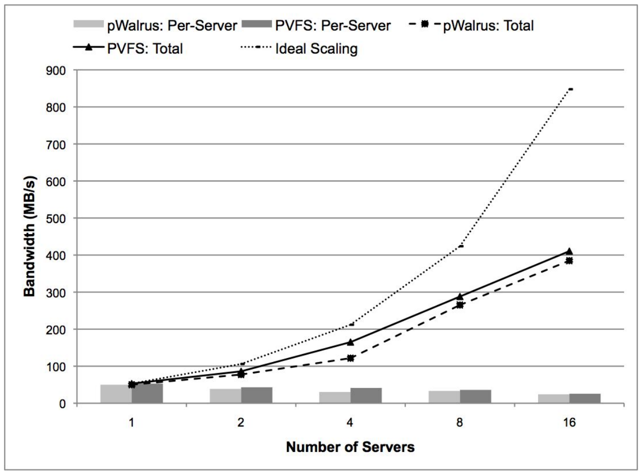

Figure 5. Results of downloading data with pWalrus and direct access to PVFS.

convention, such as "*objectName*.`pwalrus`.*version*," where *objectName* represents the name of the object, `pwalrus` indicates that the file can be used by pWalrus, and *version* is a unique number for distinguishing the file from the others corresponding to different versions of the same object.

IV. Preliminary Performance Results

We performed preliminary experiments to demonstrate the performance benefits of the parallel S3 access provided by pWalrus and of the direct access to the parallel file system underlying S3 storage. The objectives of these experiments are to confirm some of the advantages pWalrus offers: load distribution to multiple storage servers and efficient access to backing files rather than objects.

A. Experimental Set-up

Figure 4 shows our experimental set-up. We use PVFS 2.8.1 as the parallel file system that backs pWalrus, and dedicate a single machine to be the PVFS metadata server. There are N pWalrus servers running on distinct machines. The Eucalyptus packages built from our modified source code are installed on these server machines, and only the functionalities relevant to pWalrus are used. Also, each of the pWalrus server machines runs a PVFS I/O server exporting physical storage managed by Linux native `xfs` file systems. Also, on these server machines, we run PVFS client servers using the PVFS2 Linux kernel interface to mount PVFS as a POSIX file system. As a result, pWalrus servers transparently access the directories mapped to PVFS.

There are N (equivalent to the number of servers) clients that make concurrent REST S3 requests using `s3curl` to the pWalrus service, each downloading a 2 GB object containing text. The number of servers and clients, N, is varied from 1 to 16. The load on each server is evenly distributed, and thus the number of clients per server is always 1.

In addition to measuring S3 access bandwidth through pWalrus servers, we also measure the time it takes for the client machines to copy a 2 GB file each directly from the parallel file system to their local disk. In this case, the parallel file system client is mounted on the client machines using the PVFS2 Linux kernel interface. The `cp` command is used to copy each 2 GB file from a mounted PVFS directory to another directory contained in a local file system. Similar to the pWalrus measurements, the numbers of PVFS I/O servers and clients are kept equal, and varied from 1 to 16.

Both the server and client machines are equipped with two quad-core 2.83 or 3.0 GHz CPUs and 16 GB of memory (except two client machines having 32 GB of memory), and are connected through a 10 Gbps network. Each PVFS I/O server machine uses a SATA 7200 RPM hard disk with a 1 TB capacity as the physical storage for the local file system. All the machines run Linux kernel version 2.6.32. Before each measurement, the page, inode, and dentry caches are cleared on the server machines using the proc file system to ensure that the downloaded files are read from the disks.

B. Results

The results of our experiments are summarized in Figure 5. The x-axis shows the number of pWalrus servers for S3 access and that of PVFS I/O servers for direct file system access. The y-axis shows the throughput of downloading data from the servers. For each of the pWalrus and PVFS cases, the aggregate bandwidth of all servers (shown as plotted lines) and the bandwidth per server (which is equivalent to per disk and shown as bars) are shown. The numbers shown are the average of three measurements, which did not vary significantly.

The graph shows that, compared to the single pWalrus server case of 50 MB/s, arrays of multiple pWalrus servers are able to deliver increased aggregate bandwidth as a storage service, with 16 servers providing 385 MB/s. Overall, pWalrus bandwidth is only a little lower than the corresponding PVFS bandwidth. The results also show how fast we can retrieve data when we bypass the S3 interface and access PVFS directly. Starting from a single I/O server delivering 53 MB/s, PVFS achieved a maximum of 411 MB/s with 16 I/O servers. The bandwidth per server drops to around 25 MB/s as the number of servers increases. In these experiments both the numbers of servers and requests are scaled, and thus ideally the per-server bandwidth would remain constant. The limited scalability and performance in the graph may be caused by poor tuning of the PVFS configuration, or because we access PVFS through the kernel interface rather than a library binding, a choice we made for portability reasons.[4]

These results illustrate that the pWalrus storage model improves data access efficiency by two ways. First, an array of pWalrus servers can be used to distribute data writes and reads to different PVFS I/O servers, and has the potential to make available the scalability of the underlying parallel file system through the S3 interface. Second, the exposition of the files backing S3 objects to users allow them to bypass the S3 servers and directly benefit from the high performance of the underlying parallel file system.

V. Related Work

As cloud computing has gained increasing popularity, a number of infrastructures for it have emerged. Examples of such infrastructures, in addition to Amazon EC2 and Eucalyptus, include Tashi [11], vCloud [12], RightScale [13], and GoGrid [14]. Some of these, such as Tashi and vCloud, are used for hosting clouds in privately owned clusters, while others offer physical computer resources as part of virtualized computing environments as does Amazon EC2. As mentioned previously, pWalrus behaves as a functionally thin layer on top of a parallel file system, and exploits the file system for sharing management data between its servers. Therefore, pWalrus can be independent from other parts of the infrastructure being used, and thus is expected to work not only with Eucalyptus but also with other infrastructures for private cloud computing environments.

The implementation of pWalrus transparently accesses its physical storage through a regular mounted file system directory. Thus, it is expected to work without extra effort with a variety of parallel file systems that are commonly used in cluster environments. Examples of those file systems include PVFS [3], PanFS [4], GPFS [5], Lustre [6], HDFS [15], PLFS [16], and pNFS [17]. Through the S3 interface, pWalrus exposes the scalability of such file systems beyond the boundary of the cluster to users accessing their data from the outside.

[4]Also, there were a small number of Hadoop jobs running in the testbed during our measurements, which affected the results to some extent.

Finally, there exist a number of tools for facilitated data transfer over a network that are similar to pWalrus and S3. GridFTP [18] is an extension to FTP capable of exploiting higher parallelism at the TCP connection level compared to S3. Recent proposals in the NFS v4 working group [19] suggest support for pathless objects, which exist in a flat name space that is different from traditional directory hierarchies and closer to the S3 name space based on buckets and objects. WebDAV [20] is also a web-based file store, and has some similarities to pWalrus and S3 such as data content identified by URI and metadata represented in XML formats, although it has a different set of semantics from S3.

VI. Conclusion

In this paper, we proposed a storage model for cloud computing environments that enables users to access their data both through the S3 interface and through direct access to the parallel file system backing the S3 storage. For its simplicity and convenience, S3 storage has become a primary way of storing and accessing data in clouds. However, rather than view it as a full-fledged storage service that hides the physical storage system behind it and has users solely rely on it, we use it as an additional interface that enables users to further benefit from parallel file systems available in their clusters. This would be of particular interest to the administrators of private clouds, where the available storage resources are fixed and it is important to exploit as much of them as possible.

Acknowledgment

We would like to thank Rich Wolski and Neil Soman of Eucalyptus for their explanations and advice, and our anonymous reviewers for their useful comments. The work in this paper is based on research supported in part by the Betty and Gordon Moore Foundation, by the Department of Energy, under award number DE-FC02-06ER25767, by the Los Alamos National Laboratory, under contract number 54515-001-07, and by Google and Yahoo! research awards. We also thank the member companies of the PDL Consortium (including APC, DataDomain, EMC, Facebook, Google, Hewlett-Packard, Hitachi, IBM, Intel, LSI, Microsoft, NEC, NetApp, Oracle, Seagate, Sun, Symantec, and VMware) for their interest, insights, feedback, and support.

References

[1] "Amazon Simple Storage Service API reference, API version 2006-03-01," 2006.

[2] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus open-source cloud-computing system," in *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid.* Washington, DC, USA: IEEE Computer Society, 2009.

[3] I. F. Haddad, “PVFS: a parallel virtual file system for Linux clusters,” *Linux Journal*, 2000.

[4] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, “Scalable performance of the Panasas parallel file system,” in *FAST’08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*. Berkeley, CA, USA: USENIX Association, 2008.

[5] F. Schmuck and R. Haskin, “GPFS: a shared-disk file system for large computing clusters,” in *FAST ’02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*. Berkeley, CA, USA: USENIX Association, 2002.

[6] “Lustre File System,” http://www.lustre.org/.

[7] H. Krawczyk, M. Bellare, and R. Canetti, “HMAC: Keyed-hashing for message authentication,” 1997.

[8] “Amazon Elastic Compute Cloud (Amazon EC2),” http://aws.amazon.com/ec2/.

[9] “Elastic Block Storage,” http://aws.amazon.com/ebs/.

[10] “Amazon Web Services Developer Community : Amazon S3 Authentication Tool for Curl,” http://developer.amazonwebservices.com/connect/entry.jspa?externalID=128.

[11] M. A. Kozuch, M. P. Ryan, R. Gass, S. W. Schlosser, D. O’Hallaron, J. Cipar, E. Krevat, J. López, M. Stroucken, and G. R. Ganger, “Tashi: location-aware cluster management,” in *ACDC ’09: Proceedings of the 1st workshop on Automated control for datacenters and clouds*. New York, NY, USA: ACM, 2009.

[12] “Cloud Computing Services with VMware Virtualization - Cloud Infrastructure,” http://www.vmware.com/solutions/cloud-computing/.

[13] “Cloud Computing Management Platform by RightScale,” http://www.rightscale.com/.

[14] “Cloud Hosting, Cloud Computing, Hybrid Infrastructure from GoGrid,” http://gogrid.com/.

[15] “HDFS Architecture,” http://hadoop.apache.org/common/docs/current/hdfs_design.html.

[16] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate, “PLFS: a checkpoint filesystem for parallel applications,” in *SC ’09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. New York, NY, USA: ACM, 2009.

[17] “pNFS,” http://www.pnfs.com/.

[18] “GridFTP universal data transfer for the grid (the Globus project, white paper),” 2000.

[19] D. Roy, M. Eisler, and A. RN, “NFS pathless objects (internet-draft),” 2010.

[20] L. Dusseault, “HTTP extensions for web distributed authoring and versioning (WebDAV),” 2007.