Efficient VM Storage for Clouds Based on the High-Throughput BlobSeer BLOB Management System

Alexandra Carpen-Amarie, Tuan Viet Dinh, Gabriel Antoniu

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Efficient VM Storage for Clouds Based on the High-Throughput BlobSeer BLOB Management System

Alexandra Carpen-Amarie — Tuan Viet Dinh — Gabriel Antoniu

N° 7434

Septembre 2010

ISSN 0249-6399 ISRN INRIA/RR--7434--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

Efficient VM Storage for Clouds Based on the High-Throughput BlobSeer BLOB Management System

Alexandra Carpen-Amarie*, Tuan Viet Dinh* , Gabriel Antoniu*

Thème : Calcul distribué et applications à très haute performance
Équipe-Projet KerData

Rapport de recherche n° 7434 — Septembre 2010 — 12 pages

Abstract: Cloud computing has recently emerged as a new computing paradigm aiming at providing reliable, flexible, IT infrastructure and services based on externalized, virtual resources. In this context, users may upload Virtual Machine (VM) images into a Cloud storage service, from which they are propagated on demand to the physical nodes on which they are supposed to run. It is therefore important for the Cloud storage service to provide efficient support for VM storage in a context where a large number of clients may concurrently upload a large number of VMs, each of which may subsequently be needed by a large number of computing nodes. This paper addresses the problem of building such an efficient distributed repository for Cloud Virtual Machines. To meet this goal, our approach leverages BlobSeer, a system for efficient management of massive data concurrently accessed at a large-scale, as a storage back end for the Cloud VM repository. As a case study, we consider the Nimbus Cloud environment, whose repository currently relies on the GridFTP high-performance file transfer protocol. We have integrated BlobSeer as a back-end storage layer for GridFTP and evaluated our prototype on the Grid'5000 testbed.

Key-words: Distributed storage, Storage back end, Cloud storage service, Nimbus, GridFTP

* INRIA Rennes - Bretagne Atlantique/IRISA, Rennes, France
{alexandra.carpen-amarie,viet.dinh,gabriel.antoniu}@inria.fr

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Vers un système de stockage efficace pour les machines virtuelles dans les Clouds

Résumé : Le *Cloud computing* a récemment émergé comme un nouveau paradigme qui vise à fournir une infrastructure informatique et des services fiables et flexibles en s'appuyant sur des ressources virtuelles. Dans ce contexte, les utilisateurs peuvent télécharger des images de machines virtuelles vers un service de stockage *Cloud*, d'où elles sont propagées sur demande vers les noeuds physiques sur lesquels elles vont être exécutées. En consequence, il est essentiel que le service de gestion des données *Cloud* permette le stockage efficace des machines virtuelles quand de nombreux utilisateurs téléchargent des machines virtuelles simultanément.

Cet article porte sur le problème de la conception d'un tel service de stockage pour les machines virtuelles dans le *Cloud*. Notre approche s'appuie sur l'utilisation de BlobSeer, un système dédié à la gestion des données accedées par de nombreux clients à large échelle, comme back-end de stockage pour les machines virtuelles. Nous avons pris comme étude de cas la plate-forme Nimbus dont le système de stockage repose actuellement sur le protocole de transfert de fichiers GridFTP. Nous avons intégré BlobSeer comme couche de stockage pour GridFTP et nous avons évalué notre prototype sur la plate-forme Grid'5000.

Mots-clés : Stockage de données réparti, Service de stockage dans le Cloud, Nimbus, GridFTP

1 Introduction

Recently, important academic and industrial actors have started to investigate *Cloud computing*, an emerging paradigm for managing computing resources. The cloud computing model shifts the computation and data storage from the local data centers to a pool of virtualized resources hosted "in the Cloud" and the users pay only for effective resource usage. Among the various Cloud computing platforms that have been proposed, the Infrastructure-as-a-Service (IaaS) frameworks offer the largest flexibility, as they allow users to rent fully configurable virtual machines (VMs). Clients can typically upload their own VM images to a storage service, in order to enable access to a customized environment for their applications. The images are then deployed on the computing nodes rented by the client. In such a context, the Cloud storage services aim to provide the users with efficient VM repositories allowing for fast VM deployment.

This paper explains how such a storage service for the Nimbus [7, 15] Cloud environment could be built by leveraging BlobSeer [11], a BLOB management system designed for high-throughput data access under heavy concurrency. The contribution of this paper is to demonstrate how BlobSeer's concurrency-oriented features qualify this system to serve as a storage back end for the current Nimbus VM repository, which relies on GridFTP [1], a widely-used data transfer protocol.

The remainder of this paper is structured as follows. Section 2 presents the Nimbus environment and its storage service and identifies the scenarios that can benefit from an efficient distributed storage back end. Section 3 introduces BlobSeer and details our contribution. We evaluate our implementation on the Grid'5000 testbed: results are presented in Section 4. Finally, Section 5 draws conclusions and directions for future work.

2 Background

2.1 The Nimbus Cloud storage service

Nimbus [7, 15] is an open-source Infrastructure-as-a-Service Cloud framework, allowing the users to rent virtualized remote resources. It was designed to address the needs of the scientific community in multiple contexts, ranging from high-energy physics [8] to bioinformatics [10]. The architecture of a Nimbus cloud is based on four modular components, on top of which new modules are added to enable easy cluster configuration, interfaces to other IaaS clouds or optimized VM scheduling on physical resources.

The Cloud Client provides the users with the commands for launching and managing virtual resources.

The Workspace service is a standalone site VM manager and plays the role of the entry point for the Cloud. It handles client requests for virtualized resources and manages VM deployment.

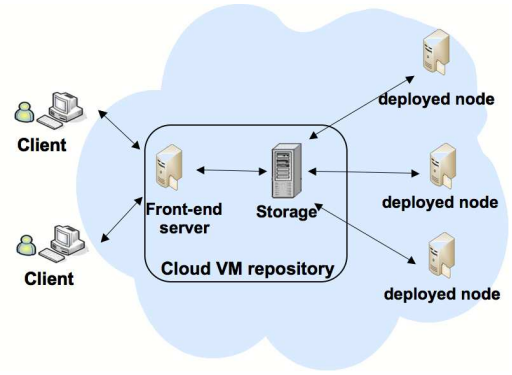


Figure 1: Usage scenario for a cloud VM repository

The Workspace Control is an agent running on each node, on which it handles VM deployment, management and configuration.

The storage service for VM images consists of a repository from which the images are deployed to the nodes allocated by the Workspace Service. When a Workspace control process has to deploy a new virtual machine image on its node, it directly accesses the repository and copies the image. In order to allow the users to use personalized environments, the system has to provide a means for them to upload or download their own virtual machine images. In Nimbus, VMs are stored on a *GridFTP* [1] server, which has the advantage of a standardized interface exposed to the users who need to upload specific VMs to the Cloud.

GridFTP is a widely-spread data transfer protocol implemented within the Globus Toolkit, providing high-performance data transfers for data-intensive scientific applications targeted for Grid environments. The GridFTP server [2] is designed to support different storage back ends. While exposing the same interface to GridFTP clients, it hides the details of how uploaded data is actually stored. It relies on an abstraction layer called the Data Storage Interface (DSI) [9], which is responsible for reading and writing data from/to the underlying storage back end. Several implementations have been proposed for this interface, relying on various back ends: POSIX-compliant file systems (used by default), HPSS (High Performance Storage System [19]) or SRB (Storage Resource Broker [4]). Finally, some DSI implementations, such as MAPFS [13, 17] and Hadoop File System [18, 5] aim at optimizing the cost of parallel data transfers.

2.2 Our approach: overview

Our goal is to improve the performance of the Nimbus Virtual Machines storage service, with respect to the following two scenarios (depicted on Figure 1):

Simultaneous deployment of VM images on multiple compute nodes. Currently, the storage service for VM images is implemented as a single physical

GridFTP server that stores all available images. When a Workspace Control daemon receives a request from the Workspace service to deploy a new VM, it has to copy it on its local file system from the repository. This approach has a major limitation when the system has to handle the deployment of a single virtual machine image on hundreds of nodes requested at the same time. The repository becomes a bottleneck that slows down the whole resource-leasing process and directly impacts on the response time to the clients. Therefore, the performance of the VM deployment process can be improved by replacing the current storage layer with a distributed storage system designed to provide a high-throughput data transfer under heavy concurrency. Moreover, the ideal candidate for the storage layer has to support partial-file transfers, as booting the VM images does not always require the whole image to be copied locally.

Concurrent uploading of VM images by multiple clients. Any IaaS Cloud storage service has to provide the users with a means to upload and download VM images. For this purpose, the Nimbus storage service employs a GridFTP server as a front end for the storage layer, as it enables the use of the widely-known GridFTP protocol for the data transfers from the clients. However, the storage back ends currently supported by GridFTP are limited and are not suitable for massively concurrent VM image uploading by a large set of clients. Coupling the GridFTP front end with a new concurrency-optimized storage layer addressing these particular needs is precisely our goal.

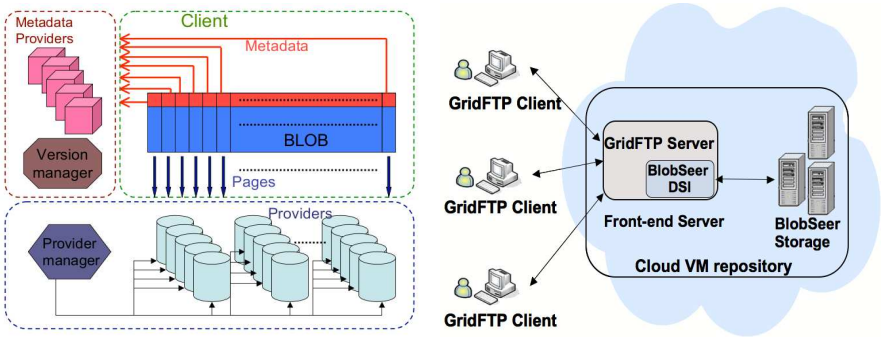
2.3 Related work

We focus on the challenges that a cloud storage service has to overcome, and in particular on the specific requirements for the systems that store the virtual machines images. All the main actors in the Infrastructure-as-a-Service cloud landscape, such as Amazon with its EC2 [3] system or Eucalyptus [14] typically propose their own storage service for user data and virtual machine images.

The Amazon Simple Storage Service (Amazon S3) [16] is a highly-scalable and reliable web-service that allows users to store data in Amazon data centers. Any client can use Amazon S3 to store and retrieve any amount of data from anywhere on the web, by accessing its web-services interface and paying for the data transfers and for the duration the data is kept. The system provides storage for flat data sequences called *objects*, which can be retrieved by their unique name within containers named *buckets*. Amazon S3 acts both as a VM repository and user-data keeper. Users can upload and download specific Amazon Machine Images (AMIs) that can be further deployed on compute nodes provided by the Amazon EC2.

Walrus [12] is a data-storage service designed for the open-source Eucalyptus system. It is interface compatible with Amazon S3, allowing users to store persistent data, organized as buckets and objects.

While the implementation of the Amazon S3 is proprietary, Walrus has the same default configuration as the Nimbus storage service, namely it runs on



(a) The architecture of the BlobSeer system (b) The BlobSeer DSI: architecture

Figure 2: Integrating BlobSeer with the GridFTP server

a single node that handles all the operations performed on the virtual machine images: upload/download, copy to the compute nodes. Therefore, both systems have the same limitations concerning the slow deployment of a large number of concurrent virtual machines and would benefit from having a distributed storage layer optimized for VM operations.

3 A concurrency-optimized distributed storage for GridFTP

3.1 The BlobSeer data management service

BlobSeer [11] is a data-management system designed to address some key requirements of large-scale data-intensive applications, such as scalability, transparency with respect to data location and the ability to sustain a high throughput under heavy access concurrency. The key features that make it suitable for a cloud storage system include the following:

Data striping. BlobSeer can handle large, unstructured sequences of bytes called *Binary Large Objects* (BLOBs), that can reach the order of TB. They are split into equally-sized chunks which are distributed among the storage nodes. This feature enables writing and reading data in parallel, and thus an increased throughput.

Distributed metadata management. BlobSeer employs a distributed metadata-management scheme, where the metadata associated with each data chunk are stored as a segment tree. The nodes that store the metadata trees are organized as a DHT that leverages the performance of concurrent accesses to metadata and eliminates the bottleneck of having a centralized metadata manager.

Lock-free versioning-based data accesses. A crucial design principle in BlobSeer is that data is never modified. Instead, each time an update is performed on a BLOB, the new patch is added to the system. Its metadata are weaved with the previous version's metadata tree, and a new version is created. Thus, no locking mechanism is needed, and the system can sustain a high throughput while many concurrent updates proceed in parallel on the same BLOB.

The architecture of BlobSeer relies on several distributed components, as shown in Figure 2(a). *Clients* create, read, write and append data from/to BLOBs. The system is built to deal with a large number of concurrent clients, that can access the same or different BLOBs. *Data providers* physically store the BLOB chunks generated by client operations, while *the metadata providers* store the metadata associated with each BLOB. *The provider manager* keeps track of all storage providers in the system and it implements a configurable placement strategy for the newly generated chunks on the providers. *The version manager* is responsible for the serialization of concurrent writes to the same BLOB.

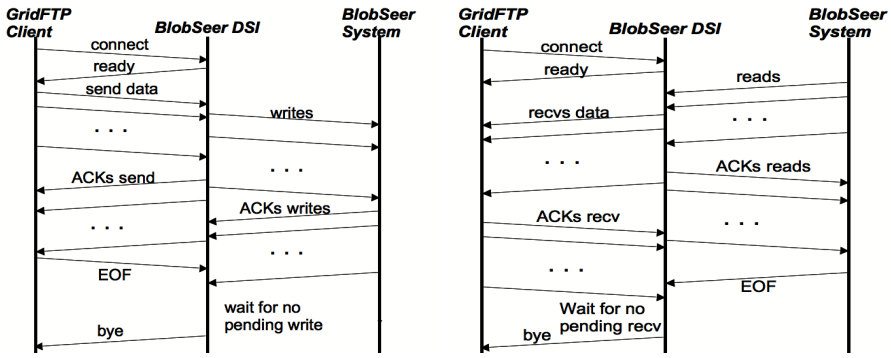
3.2 Integrating BlobSeer with GridFTP

To allow GridFTP to communicate with the BlobSeer system, we implemented the Data Storage Interface provided by GridFTP. The DSI consists in a set of function signatures associated with specific semantics that encapsulate the interaction with the storage layer. When the GridFTP has to perform an action that involves the storage layer, it passes a request to the DSI module. Each specific DSI implementation has to be able to service the request and notify the server when it has finished.

Our approach was to translate the requests dispatched by the GridFTP server into BlobSeer specific calls. Thus, the GridFTP server acts like a BlobSeer client that further accesses the BlobSeer components using its own API. As shown in Figure 2(b), the architecture we propose consists of two independent parts bridged by the BlobSeer DSI. First, the client-server communication is defined by the GridFTP protocol and is handled by the GridFTP server. Second, the server-storage communication is tackled by the BlobSeer client, which is embedded within the BlobSeer DSI.

We used the asynchronous event-handling framework provided by GridFTP, which allowed us to decouple the client-server data transfer and the transfer between the DSI and the data storage system. Each file transferred between a GridFTP client and a GridFTP server is split into equally-sized blocks. The blocks are then sent to the server, which in our implementation will store them into a queue, where the blocks wait until they can be submitted to the storage layer. The DSI is notified each time a block was inserted into the queue, and can communicate with the BlobSeer system independently from the transfers on the client-server side.

A fully functional DSI implementation has to handle the file-transfer operations, namely the file upload and download, depicted in Figure 3, and a set of commands that control the file namespace, such as make directory, delete directory, delete file or list directory. As an example, we detail one of the most



(a) The upload operation (b) The download operation

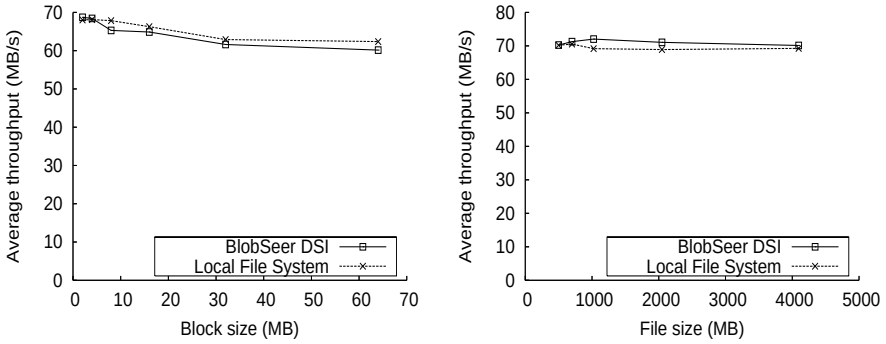
Figure 3: Data transfer diagrams

representative operations for our BlobSeer DSI: the upload of a file from the GridFTP client to the BlobSeer system:

1. The client sends a request to the GridFTP Server to store a file.
2. Before the file transfer, the GridFTP sever decides on a block size and an optimized number of parallel streams that concurrently transfer blocks of data between the client and the server. The server also issues a request for the storage system to create a new file associated with a new BLOB.
3. The GridFTP client splits the file to be sent into blocks that have a fixed size determined by the server. The blocks are then sent to the server together with their offsets within the original file, possibly in parallel if the number of parallel streams is greater than 1.
4. The GridFTP server continuously gets blocks of data from the data channel. Each block is stored in a new buffer which is added to the queue and the BlobSeer DSI is notified.
5. The BlobSeer DSI receives notifications when new blocks are added to the queue in an asynchronous fashion. The handler function extracts a block from the queue, and writes it in the BlobSeer system, at the specified offset.
6. When a write into the BlobSeer system is successfully completed, the BlobSeer DSI notifies the server, so that it can keep track of the number of successfully transferred blocks.
7. The upload operation is completed when the end of file is reached and all buffers are written into the storage system.

4 Evaluation

We evaluated our prototype through a series of experiments that assess the performance of the BlobSeer storage back end for GridFTP. We compared our



(a) Impact of the transfer block size on the throughput of the upload operation

(b) Throughput of the upload operation when the file size increases

Figure 4: Comparison between the local file system and the BlobSeer DSI as back ends for the GridFTP server

implementation with the local file system DSI, by measuring the throughput of data transfer operations when the file size varies and under concurrent accesses.

We performed our evaluations on the Grid'5000 [6] testbed, an experimental Grid platform gathering 9 sites geographically distributed in France. We used a cluster located in Bordeaux, which consists of nodes outfitted with Intel Xeon EM64T 3GHz and 2GB of RAM. Intra-cluster measured bandwidth is 110MB/s for TCP sockets with MTU set at 1500 B, latency is 0.1 ms.

In each experiment, we used one node for each GridFTP client and one node for the GridFTP server. The deployment configuration for the BlobSeer system was: 10 data providers, 2 metadata providers, one provider manager and one version manager. Each entity is deployed on a dedicated physical machine and the BlobSeer system is configured so as to store the data in the main memory of the provider nodes and not to use replication.

Impact of the transfer block size. The GridFTP protocol splits a file into equally-sized blocks whenever it has to be transferred to/from the GridFTP server. Our first set of tests aims to evaluate the influence of the block size on the throughput of the transfers. The test consists in uploading a 1 GB file from one GridFTP client to the GridFTP server. We repeat the test for several values of the block size and measure the throughput of the upload in each case. The results presented in Figure 4(a) show that the throughput obtained for the BlobSeer DSI is similar to the one obtained for the local file system. Therefore, the overhead of using BlobSeer as a storage system is not significant, and we can choose a block size optimized for the virtual machines deployment, without degrading the performance of the VM uploads.

Throughput of the upload operation when the file size increases In this test we deployed a GridFTP client that uploads a file on the GridFTP server. We fixed the block size at 4 MB and we used 2 parallel streams between the client

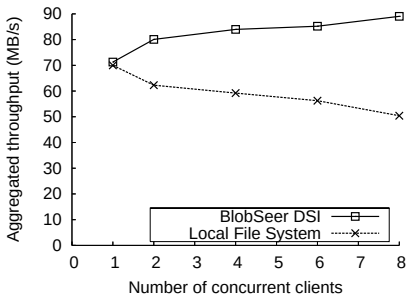


Figure 5: Impact of the number of concurrent clients on the aggregated throughput of the data transfers

and the server. We measure the average throughput of a file transfer for both the BlobSeer DSI and the local file system on the GridFTP server, when the size of the transferred file varies between 500 MB and 4096 MB. In this experiment, the BlobSeer storage layer behaves better than the typical disk storage when the size of the file increases, due to BlobSeer's ability to write data blocks in parallel to its data providers. As in the previous test, we can conclude that replacing the typical storage layer with BlobSeer does not impact on the performance of the client-server data transfers, as shown in Figure 4(b).

Performance evaluation under concurrency. We evaluated the impact of having multiple clients concurrently accessing the same GridFTP server, by measuring the aggregated throughput of all clients' transfers. Each client uploads a 1GB file on the same server that uses the local file system or the BlobSeer DSI as storage back ends. For this experiment too, we used the 4 MB block size and 2 parallel transfer streams. Figure 5 shows that the BlobSeer-based storage layer is more efficient than the default one, mainly because the BlobSeer system is optimized for sustaining a high throughput under heavy access concurrency.

5 Conclusion

In this paper, we investigated the problem of improving the performance of the VM images repository for Infrastructure-as-a-Service clouds. The main challenge for such a storage system is to be able to efficiently deploy virtual machine images to the nodes leased in a cloud environment, and therefore to sustain a high throughput while concurrently transferring data to a large number of nodes. Moreover, the system has to provide the clients with a means to upload their own virtual machine images and to support multiple simultaneous client operations. We used the Nimbus storage service as a case study. Our approach was to replace its default storage back end with BlobSeer, a distributed data-management system designed to efficiently store and transfer massive amounts of data at large scales. To achieve this goal, we integrated BlobSeer within the GridFTP framework, which is used as the front end for the Nimbus storage service. We relied on the Data Storage Interface, an abstraction layer that allows GridFTP to support different back-end storage systems. We

implemented the accesses to the BlobSeer system into this interface, by employing an asynchronous event-based approach, which allowed us to decouple the client-server transfers from the data transfers between the server and BlobSeer.

As future work, we plan to extend the BlobSeer DSI so as to support the striped configuration for the GridFTP server. This would enable us to transfer data between clusters that have a GridFTP server as a front end and to perform data transfers directly between storage nodes, while the front ends would handle only the control-related communication. Furthermore, we will compare the performance of the BlobSeer DSI with other GridFTP storage back ends, such as HDFS or MAPFS.

Acknowledgments

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see `http://www.grid5000.org/`).

References

[1] W. Allcock, J. Bester, J. Bresnahan, et al. Data Management and Transfer in High Performance Computational Grid Environments. *Parallel Computing Journal*, 28(5):749–771, 2002.

[2] W. Allcock, J. Bresnahan, R. Kettimuthu, et al. The Globus striped GridFTP framework and server. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 54, Washington, DC, USA, 2005. IEEE Computer Society.

[3] Amazon Elastic Compute Cloud (EC2). `http://aws.amazon.com/ec2/`.

[4] C. Baru, R. Moore, A. Rajasekar, and M. 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.

[5] Hadoop GridFTP. `https://twiki.grid.iu.edu/bin/view/Storage/HadoopGridFTP`.

[6] Y. Jégou, S. Lantéri, J. Leduc, et al. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *Intl. Journal of High Performance Comp. Applications*, 20(4):481–494, 2006.

[7] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa. Science Clouds: Early experiences in cloud computing for scientific applications. *In Cloud Computing and Its Application 2008 (CCA -08) Chicago*, October 2008.

[8] K. Keahey, T. Freeman, J. Lauret, and D. Olson. Virtual workspaces for scientific applications. *Journal of Physics: Conference Series*, 78(1):12–38, 2007.

[9] R. Kettimuthu, M. Link, J. Bresnahan, and W. Allcock. Globus Data Storage Interface (DSI) - enabling easy access to grid datasets. In *First DIALOGUE Workshop: Applications-Driven Issues in Data Grids*, Columbus, Ohio, USA, 2005.

[10] A. Matsunaga, M. Tsugawa, and J. Fortes. CloudBLAST: Combining MapReduce and virtualization on distributed resources for bioinformatics applications. In *ESCIENCE '08: Proceedings of the 2008 Fourth IEEE International Conference on eScience*, pages 222–229, Washington, DC, USA, 2008. IEEE Computer Society.

[11] B. Nicolae, G. Antoniu, and L. Bougé. BlobSeer: How to enable efficient versioning for large object storage under heavy access concurrency. In *2nd International Workshop on Data Management in Peer-to-peer systems (DAMAP)*, pages 18–25, St-Petersburg, Russia, 2009.

[12] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus open-source cloud-computing system. In *Proc. 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[13] M. Pérez, J. Carretero, F. Garía, J. M. Peña, and V. Robles. *MAPFS-Grid: A Flexible Architecture for Data-Intensive Grid Applications*, volume 2970/2004 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2004.

[14] The Eucalyptus Project. `http://open.eucaplytus.com`.

[15] The Nimbus Project. `http://www.nimbusproject.org/`.

[16] Amazon Simple Storage Service (S3). `http://aws.amazon.com/s3/`.

[17] A. Sánchez, M. Pérez, P. Gueant, et al. *A Parallel Data Storage Interface to GridFTP*. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2006.

[18] HDFS. The Hadoop Distributed File System. `http://hadoop.apache.org/common/docs/r0.20.1/hdfs_design.html`.

[19] D. Teaff, D. Watson, and B. Coyne. The architecture of the High Performance Storage System (HPSS). In *In Proceedings of the Goddard Conference on Mass Storage and Technologies*, pages 28–30, 1995.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399