Evaluating Cloud Storage Services for Tightly-Coupled Applications

Alexandra Carpen-Amarie[1], Kate Keahey[2], John Bresnahan[2], and Gabriel Antoniu[1]

[1] INRIA Rennes - Bretagne Atlantique / IRISA, France
[2] Argonne National Laboratory, USA

Abstract. The emergence of Cloud computing has given rise to numerous attempts to study the portability of scientific applications to this new paradigm. Tightly-coupled applications are a common class of scientific HPC applications, which exhibit specific requirements previously addressed by supercomputers. A key challenge towards the adoption of the Cloud paradigm for such applications is data management. In this paper, we argue that Cloud storage services represent a suitable data storage and sharing option for Cloud applications. We evaluate a distributed storage plugin for Cumulus, an S3-compatible open-source Cloud service, and we conduct a series of experiments with an atmospheric modeling application running in a private Cloud deployed on the Grid'5000 testbed. Our results, obtained on up to 144 parallel processes, show that the application is able to scale with the size of the data and the number of processes, while storing 50 GB of output data on a Cloud storage service.

Keywords: Cloud computing, data management, Cloud storage service, HPC applications, Nimbus, Cumulus.

1 Introduction

Important academic and industrial actors have recently started to investigate Cloud computing, a rapidly expanding paradigm for hosting and delivering on-demand services on a pay-per-use basis. The increasing popularity of the Cloud computing model has drawn the attention of the high-performance computing community, research in this area focusing on the usage of Cloud infrastructures as alternatives to the traditional dedicated supercomputers. Tightly-coupled scientific applications have a unique range of computing needs, from scalability and processing power to efficient parallel I/O and high-performance network interconnects. Several studies [7, 8, 11, 20] have analyzed the impact of porting such applications on Cloud environments, evaluating the tradeoff between the benefits of on-demand computing power and the Cloud cost model, and the performance overhead introduced when hosting HPC applications in the Cloud.

However, past research mainly focused on performance as a means to quantify the HPC capability of public [21] or private [7] Clouds. Another key challenge that can directly impact the effectiveness of using clouds for HPC workloads is data management. Many scientific applications generate large amounts of data which need to be persistently stored and made available for further analysis. Typically, supercomputers rely on parallel file systems specifically tuned for tightly-coupled application workloads for their storage needs. The versatility of Cloud platforms allows users to customize their virtual machines (VMs) with any file system, this solution being adopted by most of the existing studies. Although it allows users to recreate the original environment used for supercomputers, this approach comes with a performance penalty caused both by the time needed to deploy and configure their own storage mechanisms and the unreliability of VMs local disk storage.

In this paper we analyze an alternative solution: providing Cloud storage services for HPC applications executed on Clouds. We investigate the requirements of such a Cloud data management system and we evaluate a tightly-coupled scientific application with a customized open-source Cloud service. Whereas most studies [5,16,21] focus on public Cloud infrastructures such as Amazon's Elastic Compute Cloud (EC2) [2] and its Simple Storage Service (S3) [17], we perform our evaluations on top of an open-source Cloud that enables us to thoroughly control the physical infrastructure and the Cloud configuration. To this end, we rely on the Nimbus Cloud framework [13] and its S3-compatible storage service called Cumulus [3].

The reminder of this paper is structured as follows. Section 2 describes the motivation of our research. In Section 3 we introduce the applications we target and we detail Cloud Model 1 [6], an MPI-based applications that simulates atmospheric phenomena. Section 4 is dedicated to our Cloud storage service solution relying on Cumulus and BlobSeer [15]. An experimental evaluation of this system is presented in Section 5 and finally, Section 6 draws conclusions and ideas for future work.

2 Motivation

Infrastructure-as-a-Service Clouds typically provide the user with a set of virtual machine instances that can host applications. Such VMs are equipped with local disks the applications can use to store generated or input data. This storage solution, however, is not persistent, as the disk is wiped out each time a virtual machine lease ends. To cope with this issue, Amazon provides services such as the Elastic Block Store (EBS) [1]. Essentially, EBS allows users to attach a virtual disk to each of their VMs, which can then be backed up onto S3 to persistently store saved data. This solution has however a major drawback: each EBS disk corresponds to a specific virtual machine, and therefore the various VMs cannot share the stored data. Furthermore, as computing VMs carry out simulations and generate output data on local EBS disks, no application can access the whole final results without scanning each EBS disk and possibly copying the data onto a single disk to enable further processing. In contrast, uploading generated data directly into a Cloud storage system might overcome the limitations of this approach, enabling users to achieve not only persistency, but also the capability to have a globally shared view of their results.

The works that studied the performance of HPC applications in Cloud settings have typically entrusted data storage and management tasks to parallel file systems, in an attempt to recreate the original environments deployed on supercomputers. While this approach has the advantage of providing the application with a standard file system interface, aggregating the local storage space of virtual machines within a distributed file system does not guarantee data persistency. The computed results are either lost at the end of the VM lease, or the user has to manually save them into a persistent repository, such as Amazon S3, to make them available to higher-level applications. This operation increases the completion time and consequently, the cost of running the application in the Cloud. Additionally, VM failures can lead to data loss and require application re-execution.

Moreover, such applications typically generate several output datasets, one for each intermediate time step of the simulation. These results serve as an input for higher-level tools. For instance, data-mining and visualization tools, such as VisIt [19], may perform real-time data analysis, debugging or data aggregation for visualizing the output at each timestep. Replacing local storage with a Cloud-hosted service may enable real-time visualization and analysis for each dataset, as well as availability guarantees and standard interfaces to facilitate access to data.

3 Application Model

3.1 Tightly-Coupled Application Model

We target tightly-coupled, high-performance computing applications specific to the scientific community. Such applications exhibit a set of common features, discussed below.

Parallel processes. A wide range of HPC applications split the initial problem into a set of subproblems. Then, these smaller subproblems are spread across a fixed set of processes, which handle the data in parallel. Such applications typically rely on message-parsing systems (e.g., MPI) for inter-process communication and synchronization.

Compute-intensive simulations. We consider applications that simulate complex phenomena in various contexts, such as high-energy physics or atmospheric simulations. They usually require significant computing resources and spend more time to compute results than to perform I/O operations.

Massive output data. Real-life simulations involve large-sized output, as they compute a set of variables describing the evolution in time of the modeled phenomenon. They are typically designed to store results and additional application logs in a parallel file system, such as GPFS [18] or PVFS [14].

No concurrent access to files. Each process computes a subset of the problem output data. Concurrently dumping results into a single shared output file may lead to I/O bottlenecks prone to decrease the overall performance of the application. Therefore, we consider applications involving independent processes, which perform write operations in separate files.

3.2 Case Study: The CM1 Application

Cloud Model 1 (CM1) [6] is a three-dimensional, time-dependent numerical model designed for atmospheric research, in particular for modeling major phenomena such as thunderstorms. CM1 simulates a three-dimensional spatial domain defined by a grid of coordinates specified in a configuration file. For each spatial point, the application is designed to compute a set of problem-specific variables, including wind speed, humidity, pressure or temperature. A CM1 simulation involves computing the evolution in time of the parameter set associated with each grid point. To this end, the 3D domain is split along a two-dimensional grid and each obtained subdomain is assigned to its own process. For each time step, all processes compute the output corresponding to their subdomain, and then they exchange border values with the processes that handle neighboring subdomains. The computation phases alternate with I/O phases, when each process dumps the parameters describing its subdomain to the backend storage system. CM1 is implemented in Fortran 95 and the communication between processes relies on MPI [10].

4 Our Approach: A Scalable Cloud Storage Service

4.1 Background

Cumulus [3] is an open-source Cloud storage system that combines efficient data-transfer mechanisms and data-management techniques, aiming at providing data Cloud services in the context of scientific applications. It is designed to support swift data transfers using S3-compatible interfaces. Cumulus features a modular architecture that enables an efficient interaction with external modules. In particular, it is built on top of a storage interface that allows system administrators to customize their service according to their Cloud's requirements.The storage interface decouples the processing of client requests from the actual implementation of the storage backend, making Cumulus a versatile tool that can be adapted to various contexts. The default storage backend shipped with the Cumulus service implements the interaction with a POSIX-compliant file system. It provides support for interconnecting Cumulus with either local file systems or parallel file systems that expose a POSIX interface, such as NFS, PVFS or GPFS.

4.2 Towards a Cloud Storage Backend for Efficient Concurrent Data Transfers

To provide an efficient alternative for traditional file systems, a Cloud data service has to comply with several requirements specific for large-scale applications.

Support for massive files. Scientific applications typically process huge amounts of records, which cannot be hosted in separate, small files. To efficiently store such datasets, data services have to collect all the records into massive files that can reach Terabytes of data in size.

High-throughput concurrent data transfers. Parallel data processing is one of the crucial features that allow scientific applications to accommodate large amounts of data. To enable efficient support for such applications in the Cloud, data-management platforms have to sustain high-throughput data transfers, even under heavy access concurrency.

Fault tolerance. Reliability is a key requirement for Cloud storage, especially in public Cloud environments. It can be achieved by employing fault-tolerant storage backends. Moreover, *data versioning* is a desirable feature in such contexts, as it enables a transparent support for rolling back incorrect or malicious data modifications.

The aforementioned requirements represent the design principles of BlobSeer [15], a concurrency-optimized data-management system for data-intensive distributed applications. BlobSeer specifically targets applications that handle massive unstructured data,

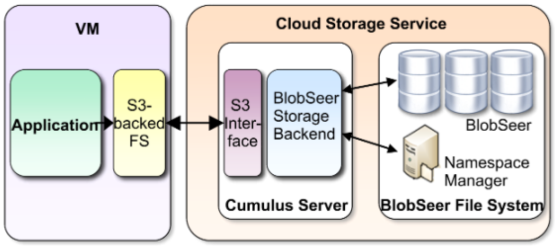


Fig. 1. The architecture of the BlobSeer-backed Cumulus storage service

called *blobs*, in the context of large-scale distributed environments. Its core operations rely on data striping, distributed metadata management and a versioning-based concurrency control mechanism to achieve efficient data transfers under highly-concurrent data accesses. The architecture of BlobSeer and its performance evaluation have been presented in detail in several works, such as [15].

To optimize Cumulus for large-scale, high-throughput concurrent data transfers, we designed and implemented a BlobSeer-based distributed storage backend for the Cumulus service. To this end, we enhanced BlobSeer with a file-system layer, enabling it to meet the requirements of the storage interface defined by Cumulus.

4.3 Design Overview

In order to run unmodified HPC applications in the Cloud and yet benefit from the Cloud storage solutions, we implemented two interface layers. First, we integrated BlobSeer as a backend for Cumulus, by designing a file-system interface for BlobSeer. Second, as typical Cloud storage services expose an S3 interface, we implemented a file-system module to run inside the VMs and stream file-oriented application data to any S3-compatible service. Thus, applications executed inside the VMs store their output data on local paths, assigning the file-system module the task of automatically transferring the data to the Cloud service, i.e. Cumulus in our case, and finally to the persistent storage backend (as shown in Figure 1).

The BlobSeer file-system interface. We designed a file-system layer on top of BlobSeer to enhance it with an easily-accessible and hierarchical file namespace, while preserving the efficient concurrent data operations built into the system. To this end, we equipped BlobSeer with a *namespace manager*, a centralized entity in charge of managing the file hierarchy and mapping files to *blobs*. The namespace manager implements the file system API, exposing the versioning interface of BlobSeer as well as standard file operations, such as create, open, read, write, close. However, the file system layer does not implement the POSIX semantics, preserving instead the original Blobseer primitives, along with their high-throughput data access guarantees.

The S3-backed file system layer. This module allows applications to interact with standard files that are transparently backed by an S3-compliant Cloud service. This interface layer includes a POSIX-compatible interface relying on FUSE (Filesystem in Userspace) [9]. Each write operation initiated by the application is translated into an upload to the S3 service. As a result, each output file is forwarded to the persistent

Cloud storage and at the same time it is made available to higher-level tools that can process it as the simulation continues. To optimize read operations, we introduced a prefetching mechanism that downloads the files from the S3 repository and stores them locally to improve read access time.

5 Experimental Evaluation

We conducted a set of experiments to analyze the performance and scalability of the Cumulus Cloud storage service in two different contexts. First, we investigated the behavior of the Cumulus service through synthetic benchmarks that assess its data transfer capabilities. Second, we focused on the CM1 application and the impact of the Cloud storage deployment on its performance.

5.1 Environmental Setup

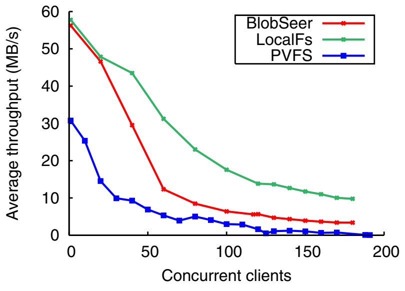
We carried out a set of experiments on Grid'5000 [12], an experimental testbed gathering 10 geographically-distributed sites in France. We used the Rennes cluster of Grid'5000. The nodes it provides are interconnected through a 1 Gbps Ethernet network, each node being equipped with at least 4 GB of memory.

We performed a comparison between several storage backends for Cumulus. First, we used the local disk of each Cumulus server as the storage backend, thus employing the storage space of all the Cumulus nodes. While this approach does not provide a global view of the data and therefore cannot be used for a real-life application, we included this evaluation as a baseline against which to assess the performance of the other backends. The second storage system employed is PVFS [14], a scalable parallel file system designed to provide high performance for HPC applications, by striping data and storing it across several data servers. Finally, we evaluated the BlobSeer-based Cumulus storage. Each experiment involves a distributed deployment configuration, where a set of replicated Cumulus servers use a common storage backend to serve concurrent data transfer requests.

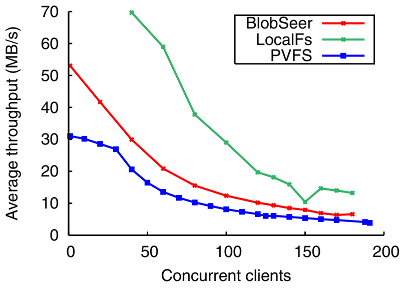
5.2 Cloud Storage Benchmark

To asses the impact of various storage backends on the scalability of the Cumulus service, we performed a set of experiments involving a large number of simultaneous data transfers, so as to simulate a typical HPC scenario when parallel processes concurrently generate output data.

In this experiment, both PVFS and BlobSeer were deployed in a similar configuration, that is 30 data storage nodes and 10 metadata nodes. Replicated Cumulus servers were co-deployed with the data storage nodes. For each execution we measured the average throughput achieved when multiple concurrent clients perform the same operation on the Cumulus service. The clients are launched simultaneously on dedicated machines. Each client performs an upload and a download for a single file of 1 GB. We increased the number of concurrent clients from 1 to 200 and we measured the average throughput of each operation. The results are shown in Figure 2. As expected,



(a) Upload average throughput for 30 Cumulus servers.



(b) Download average throughput for 30 Cumulus servers.

Fig. 2. Cumulus storage backend comparison under concurrent accesses

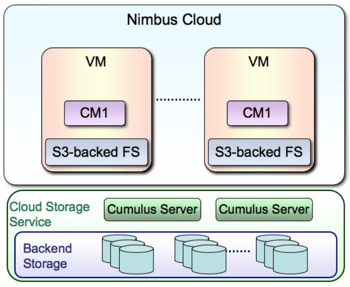
when each Cumulus server uses its own local disk for storage, the performance of the data transfers is better than when Cumulus is backed by a distributed file system. As the number of clients increases, the available network bandwidth is divided among the concurrent requests, resulting in lower average throughputs. As the number of clients increases to more than 100, the gap between the local disk backend and the distributed file systems is substantially reduced, due to the efficient data striping of the latter and the contention generated by the concurrent accesses when using the local disks. However, PVFS is not optimized for writing small blocks of data and its consistency mechanisms do not allow any client-side caching. As a result, PVFS cannot achieve its raw performance level when being used as a storage backend for the POSIX interface of Cumulus.

In contrast, the BlobSeer-based backend is specifically tuned to take full advantage of the storage system features. Despite the fact that the number of concurrent clients reaches 6 times the number of data storage nodes (which amounts to 30 nodes), the BlobSeer-backed system sustains an almost constant transfer rate for more than 60 clients and transferred data amounting to 180 GB. Thus, BlobSeer outperforms PVFS, maintaining a throughput approximatively 30% higher in the case of uploads and 60% higher for downloads.

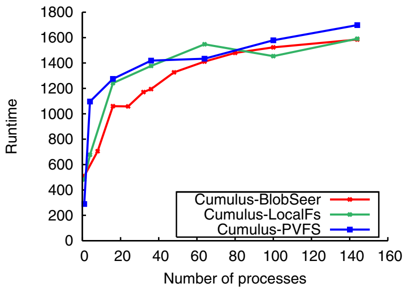
5.3 Evaluating Cumulus as a Cloud Service for CM1

To study the impact of storing HPC application data in a Cloud service, we relied on Nimbus to provide a customizable IaaS Cloud environment on Grid'5000. We employed 64 nodes to deploy Nimbus. For each experiment, the CM1 application was executed in a Nimbus virtual cluster of quadcore VMs with 4 GB of RAM. Each VM of such a cluster was equipped with the S3-backed file system, to enable CM1 to execute in parallel on the virtual cluster nodes, and to directly store its output data into Cumulus, as shown in Figure 3(a). We used 50 nodes to deploy the replicated Cumulus servers on top of the three storage backends: the local file system of each Cumulus node, BlobSeer and PVFS. Both BlobSeer and PVFS employ 50 *data providers* and 10 *metadata nodes*, each of them being deployed on dedicated machines.

CM1 is representative for a wide class of applications that simulate the evolution of a phenomenon in time. Each simulation is associated with a 3D domain and a time



(a) Experimental setup.



(b) Application runtime for 10 minutes of simulated time.

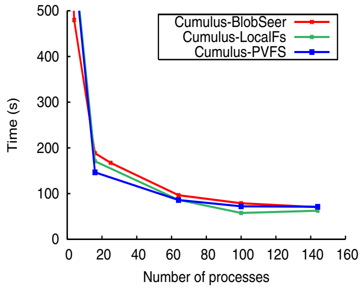
Fig. 3. Cloud storage evaluation for CM1 in a Nimbus Cloud environment

interval. A simulation consists in obtaining the values for a set of parameters for each point of the domain and for each time step. The initial domain is split among the processes and each of them is in charge of a particular subdomain. We used a 3D hurricane simulation described in [4]. The application was configured to use MPI, each process generating a set of output files for each time step.

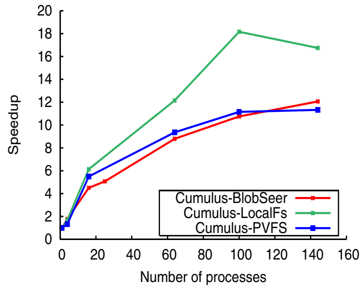
Completion time when increasing the pressure on the storage system. For the first experiment, we executed the application for 10 minutes of simulated time, with a time step of 20 seconds. We increased the number of MPI processes, maintaining constant the size of the simulated subdomain for each of them. We generated 4 intermediate output files, each of them of 85 MB in size, amounting to a total of 340 MB generated by each process per run. We deployed 4 MPI processes on each virtual machine (one for each core) and increased the number of processes from 1 to 144. The total size of the data generated for these simulations increases from 340 MB to 50 GB. Figure 3(b) shows the simulation completion time when increasing the number of processes and the output data is stored into Cumulus.

The results show a very steep increase of the completion time when the number of processes is small. However, for more than 20 processes, the curve flattens for all three storage solutions. The measured runtime trend can be explained by the increasing size of the simulated domain, which leads to a larger time spent in communication among the MPI processes that need to exchange more subdomain border values. In addition, the size of the output data increases along with the number of processes. The sustained performance for a large number of processes thus suggests the application is able to scale despite storing output data into the Cumulus external repository. The results also indicate the BlobSeer and the PVFS backends for the Cumulus servers do not lead to a performance drop for the application that stores data into Cumulus, when compared against the local file system backend. Moreover, the BlobSeer-based Cumulus version slightly outperforms the PVFS Cumulus backend. Similarly to the Cumulus benchmarks in the previous section, this result confirms the higher throughput delivered by BlobSeer in this context.

Application speedup. This experiment aims at evaluating the speedup obtained by scaling out the number of application processes for the same initial problem. For this



(a) Completion time.



(b) Speedup.

Fig. 4. Storage backend comparison for 30 minutes of simulated time with a 20 s time step.

evaluation, CM1 was executed for 30 minutes of simulation time, with a timestep of 20 seconds between consecutive computations. In contrast to the previous experiment, we increased the duration of the computation phase, so as to highlight the benefits of parallelizing the simulation on more processing nodes. For each point on the graph, Figure 4(a) depicts the time it takes the application to complete when the initial problem is divided among an increasing number of processes. Additionally, we have shown the corresponding speedup in Figure 4(b). The speedup for a specific number of processes is computed as the measured execution time of the application for a single process divided by the execution time when all the processes are employed.

As expected, as we decrease the size of the simulated spatial domain allocated to each process, the application completes its execution much faster. The drop in the execution time as we introduce new processes is a consequence of the smaller number of border values to be exchanged between them and the smaller size of the output data to be sent to the storage backend. The obtained performance of the three backends is similar, as most of the execution time accounts for computation. Furthermore, the two distributed backends achieve a comparable speedup. We however obtained a better speedup when using the local file system of each Cumulus server as a storage solution. This result is mainly due to the large execution time measured for the local file system in the case of only one process, when the all generated data had to be dumped on a single node's file system, whereas it was distributed among storage nodes for the two other backends.

6 Conclusions

In this paper we address the data management needs of tightly-coupled scientific applications executed in Cloud environments. In this context, we argue that Cloud storage services can meet some of the requirements of such applications and contribute to the adoption of the Cloud paradigm by the HPC community. The benefit of such an approach lies in the ability to provide globally-shared access to data generated by parallel processes and to enable simultaneous data analysis or visualization. We assess the performance of an open-source Cloud data service called Cumulus, backed by various storage solutions. Among them, we focused on BlobSeer, a data management system optimized for highly-concurrent accesses to large-scale distributed data. We proposed an interfacing mechanism that enables Cumulus to use BlobSeer as a storage backend and to take advantage of its efficient data transfer operations while exposing an S3-compatible interface to the users. Furthermore, we evaluated Cumulus for CM1, a real-life tightly-coupled simulator for atmospheric phenomena. We conducted experiments on a private Nimbus Cloud deployed on the Grid'5000 infrastructure, showing that a data service such as Cumulus allows the application to scale both with the number of processes and with the size of the generated output. We also examined various backends for Cumulus, the obtained results suggesting the backend choice has an impact on the completion time of the executed application, along with providing additional benefits for higher-level data analysis tools, such as data availability or standard interfaces.

As future work, we plan to perform more in-depth evaluations for various workloads and access patterns, as well as to study the advantages of leveraging Cumulus for online visualization of the generated simulation results.

Acknowledgments. This work has been supported by the ANR MapReduce grant (ANR-10-SEGI-001). The 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.

References

1. Amazon Elastic Block Store (EBS), http://aws.amazon.com/ebs/
2. Amazon Elastic Compute Cloud (EC2), http://aws.amazon.com/ec2/
3. Bresnahan, J., Keahey, K., LaBissoniere, D., et al.: Cumulus: an open source storage cloud for science. In: ScienceCloud 2011, pp. 25–32. ACM, New York (2011)
4. Bryan, G.H., Rotunno, R.: Evaluation of an analytical model for the maximum intensity of tropical cyclones. Journal of the Atmospheric Sciences 66(10), 3042–3060 (2009)
5. Carlyle, A.G., Harrell, S.L., Smith, P.M.: Cost-effective HPC: The community or the Cloud? In: CloudCom 2010, pp. 169–176 (December 2010)
6. Cloud Model 1, http://www.mmm.ucar.edu/people/bryan/cm1/
7. Ekanayake, J., Fox, G.: High Performance Parallel Computing with Clouds and Cloud Technologies. In: Avresky, D.R., Diaz, M., Bode, A., Ciciani, B., Dekel, E. (eds.) Cloudcomp 2009. LNICST, vol. 34, pp. 20–38. Springer, Heidelberg (2010)
8. El-Khamra, Y., Hyunjoo, K., Shantenu, J., et al.: Exploring the performance fluctuations of HPC workloads on clouds. In: CloudCom, pp. 383–387 (December 2010)
9. File System in UserspacE (FUSE), http://fuse.sourceforge.net
10. Gropp, W., Lusk, E., et al.: High-performance, portable implementation of the MPI Message Passing Interface Standard. Parallel Computing 22(6), 789–828 (1996)
11. He, Q., Zhou, S., Kobler, B., et al.: Case study for running HPC applications in public clouds. In: HPDC 2010, USA, pp. 395–401 (2010)
12. Jégou, Y., Lantéri, S., Leduc, J., et al.: Grid'5000: a large scale and highly reconfigurable experimental grid testbed. Intl. J. of HPC Applications 20(4), 481–494 (2006)
13. Keahey, K., Freeman, T.: Science Clouds: Early Experiences in Cloud Computing for Scientific Applications. In: Cloud Computing and Its Applications (CCA), USA (2008)
14. Ligon, W.B., Ross, R.B.: Implementation and performance of a parallel file system for high performance distributed applications. In: HPDC 1996, pp. 471–480. IEEE Computer Society, Washington, DC (1996)

15. Nicolae, B., Antoniu, G., Bougé, L., et al.: BlobSeer: Next generation data management for large scale infrastructures. J. Parallel and Distrib. Comput. 71(2), 168–184 (2011)
16. Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T., Epema, D.: A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing. In: Avresky, D.R., Diaz, M., Bode, A., Ciciani, B., Dekel, E. (eds.) Cloudcomp 2009. LNICST, vol. 34, pp. 115–131. Springer, Heidelberg (2010)
17. Amazon Simple Storage Service (S3), http://aws.amazon.com/s3/
18. Schmuck, F.B., Haskin, R.L.: GPFS: A shared-disk file system for large computing clusters. In: Conf. on File and Storage Technologies, FAST, pp. 231–244. USENIX (2002)
19. VisIt, https://wci.llnl.gov/codes/visit/
20. Younge, A.J., Henschel, R., Brown, J.T., et al.: Analysis of virtualization technologies for high performance computing environments. In: CLOUD, pp. 9–16 (July 2011)
21. Zhai, Y., Liu, M., Zhai, J., et al.: Cloud versus in-house cluster: Evaluating amazon cluster compute instances for running mpi applications. In: 2011 Intl. Conf. for HPC, Networking, Storage and Analysis, SC, pp. 1–10 (November 2011)