Efficient Management of Complex Striped Files in Active Storage*

Juan Piernas[1,2] and Jarek Nieplocha[1]

[1] Pacific Northwest National Laboratory (USA)
[2] Universidad de Murcia (Spain)
piernas@ditec.um.es, jarek.nieplocha@pnl.gov

Abstract. Active Storage provides an opportunity for reducing the bandwidth requirements between the storage and compute elements of current supercomputing systems, and leveraging the processing power of the storage nodes used by some modern file systems. To achieve both objectives, Active Storage allows certain processing tasks to be performed directly on the storage nodes, near the data they manage. However, Active Storage must also support key requirements of scientific applications. In particular, Active Storage must be able to support striped files and files with complex formats (e.g., netCDF). In this paper, we describe how these important requirements can be addressed. The experimental results on a Lustre file system not only show that our proposal can reduce the network traffic to near zero and scale the performance with the number of storage nodes, but also that it provides an efficient treatment of striped files and can manage files with complex data structures.

1 Introduction

Recent improvements in storage technologies in terms of capacity as well as cost effectiveness, and the emergence of high-performance interconnects, have made it possible to build systems of unprecedented power by connecting thousands of compute and storage nodes. However, for large-scale scientific simulations that use these environments, the efficient management of enormous and increasing volumes of data remains a challenging problem.

On the other hand, several parallel/distributed file systems have been recently developed for high-performance and high-data volume computing systems. Some of these file systems, such as Lustre [1] and PVFS [2], use mainstream server computers as storage nodes, that is, computers that contain significant CPU and memory resources, several disks attached to them, and run a general-purpose operating system. Although the combined computing capacity of these nodes can

* This work was supported by the DoE, Office of Advanced Scientific Computing Research, at the Pacific Northwest National Laboratory (a multiprogram national laboratory operated by Battelle for the U.S. DoE under Contract DE-AC06-76RL01830), and by the Spanish MEC and European Comission FEDER funds under grants "Consolider Ingenio–2010 CSD2006–00046", and "TIN2006–15516–C04–03".

E. Luque, T. Margalef, and D. Benítez (Eds.): Euro-Par 2008, LNCS 5168, pp. 676–685, 2008.

be considerable in many high performance systems, it is exploited for storing and accessing but hardly ever for processing data.

One approach to reduce the bandwidth requirements between the compute and storage elements, and to leverage the processing power of the latter, is to move appropriate processing tasks to the storage nodes. We call this approach **Active Storage** in the context of parallel file systems [3][4]. Active Storage is similar to the *active disk* concept proposed for hard drives [5][6][7][8], but with two important differences: (1) storage devices are now full-fledged computers, and (2) they include a feature-rich environment provided typically by a Linux operating system. These two factors make it possible to run regular application codes on the storage nodes.

In this paper we show how we have enhanced our previous implementation of Active Storage [4] to support several key features which, until the current work, have been lacking in virtually all existing proposals of this technology. Specifically, we focus on striped files with either a simple format (e.g., a list of chunk-aligned records) and a more *complex* format as netCDF, which is very common for data exchange in some scientific applications. In the former case, our implementation provides a framework that makes it possible to transparently run unmodified programs to process the records in a striped file. In the latter case, the program to be run on the storage nodes must be aware of the format and striping of the input file. To address this requirement we enhanced Active Storage by introducing a new component, a *mapper*, to optimize the processing of the striped file.

We have evaluated our implementation of Active Storage by using the Lustre parallel file system and two application kernels. The experimental results show that our design achieves the two main objectives of Active Storage for both striped and non-striped files: it reduces the network traffic to near zero for test workloads, and can take advantage of the extra computing capacity offered by the storage nodes at the same time. Moreover, they prove that Active Storage can efficiently manage striped files, and that high performance can be achieved even for files with complex data structures.

The rest of the paper is organized as follows. Section 2 provides an overview of work related to the *active disks'* concept. Section 3 describes the architecture of our proposal for Active Storage, and the handling of non-striped files. The treatment of (complex) striped files is explained in Section 4. Experimental results are presented in Section 5. Conclusions appear in Section 6.

2 Related Work

The idea of *intelligent storage* was developed by several authors at the end of the 90's, with some similar ideas proposed even earlier in the 80's in the database world [9]. Riedel *et al.* [8] propose a system, called *Active Disks*, which takes advantage of the processing power on individual disk drives to run application-level code. Keeton *et al.* [7] present a computer architecture for decision support database servers that utilizes "intelligent" disks (IDISKs). Acharya *et al.* [5] evaluate the *active disk* architectures, and propose a stream-based programming

model for them. Lim *et al.* [10] propose an Active Disk File System (ADFS) where many of the responsibilities of a traditional central file server are offloaded to the active disks. Chiu *et al.* use the previous ideas of Active Disks and IDISKs to design a distributed smart disk architecture [6].

The *object-based storage device* (OSD) [11][12] is another concept which profits the processing power of the disk drives. Schlosser and Iren [13] suggest that, with more processing capability in the storage devices, it would be possible to delegate some database-specific tasks to the OSDs, in an active disk fashion. Du [14] merges the OSD and active disk concepts in order to build the *intelligent storage devices* one. An intelligent storage device is directly attached to the network, supports the OSD concept, and supports the active disk concept.

Felix *et al.* [3] present a first real implementation of *Active Storage* for the Lustre file system. They provides a kernel-space solution with the processing component parts implemented in the user space. Our recent implementation of Active Storage [4], however, offers a solution that is purely in user space. This makes our proposal more flexible, portable, and readily deployable, while it achieves the same or even better performance than Felix's implementation.

None of the previous papers have described a deployable solution that addresses the practical needs of many large scale scientific applications. For example, they have not provided means to deal with striped files. Moreover, the design principles of some approaches also hinder the development of possible enhancements to deal with striped files. The current paper, however, makes a key contribution to the field by showing that, in fact, Active Storage can be implemented efficiently for striped files even for complex data formats of the user data. This is possible by taking advantage of the user-space approach and filesystem-wide data view of our proposal.

3 Active Storage Overview

Figure 1(a) shows a cluster without Active Storage, whereas Figure 1(b) depicts our approach. Without Active Storage, many data-intensive processing tasks are performed on the compute nodes, producing a high network traffic and wasting the processing power of the storage node. Active Storage, however, allows some of those processing tasks to be performed on the storage nodes, reducing the network traffic, and profiting the CPU time of these nodes.

In our proposal, the compute and storage nodes are all clients of the parallel file system. Since the storage nodes are clients, they can access all the files in the file system and, specifically, the files stored locally. One of the clients will run `asmaster`, a program which receives a *rule* describing an Active Storage job and performs the actions contained in it. A rule is an XML file which can specify many actions to perform, but which basically contains the following information: a file pattern specifying the files to process, a program path, and program arguments (if any). In the storage nodes, there also exists an Active Storage Runtime Framework (ASRF), a set of programs which assist `asmaster` in executing a rule, i.e., an Active Storage job.

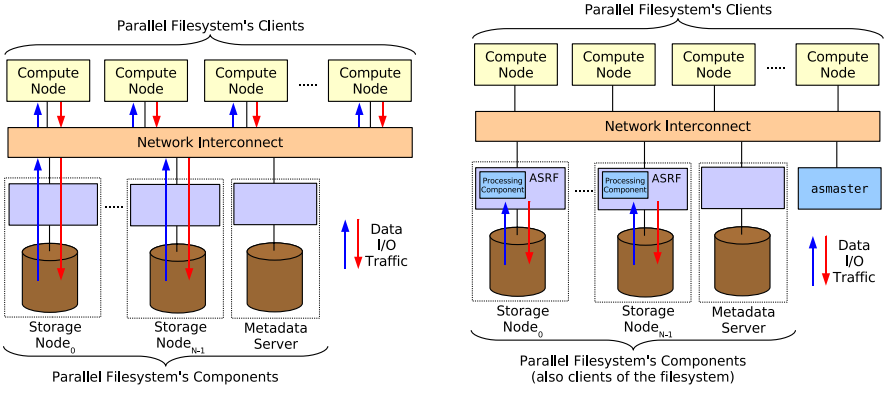

Fig. 1. (a) A traditional system without Active Storage. (b) Overview of the Active Storage architecture.

The files to be processed by an Active Storage job can already exist, or they can be created by a parallel application running on the compute nodes at the same time as the Active Storage job is active. In the latter case, the synchronization between the parallel application in the compute nodes, and the processing components in the storage nodes is important [4]. To simplify the implementation of the processing components, we have built a dynamic library, `libas`, which contains functions that hide the synchronization details. This library will also play a fundamental role in providing transparent access to striped files.

The processing of non-striped files is simple. For every matching file, `asmaster` will remotely run the given program as a processing component on the storage node where the file resides. If the processing components create output files, it is possible to instruct a rule to locally create those files. Also, if there are thousands of files which match the pattern, Active Storage will create thousands of processing components, one per file, at the same time.

Note that our Active Storage approach is quite different from running a parallel application on the storage nodes [4]. A major problem for a regular parallel application is that the application should be aware of the file distribution and the properties of the filesystem. Also, the number of files per storage node can change during the course of the time, and files can be unevenly distributed. Therefore, the number of "processing components" of the parallel application per storage node should be variable, and could change from run to run, or during the same run.

4 Management of Complex Striped Files

This section describes extensions to the Active Storage architectures to support striped files. This includes the management of striped files with a simple data format as well as files which have a complex data structure, such as netCDF.

4.1 Striped Files with Chunk-Aligned Records

In the baseline Active Storage model described in the previous section, we have assumed that files to be processed are not striped, i.e., each file is stored in only one storage node. Many parallel applications, however, stripe files across several storage nodes in order to benefit from the increased aggregate bandwidth.

To deal with striped files, our current approach launches a processing component per storage node used by the matching file, and makes every processing component process only the file chunks stored in its own node. If the processing components write to an output file, the output file must be created with the same striping pattern as the input file, and every processing component must write to only the file chunks stored in its corresponding node. Otherwise, the I/O operations would not be entirely local.

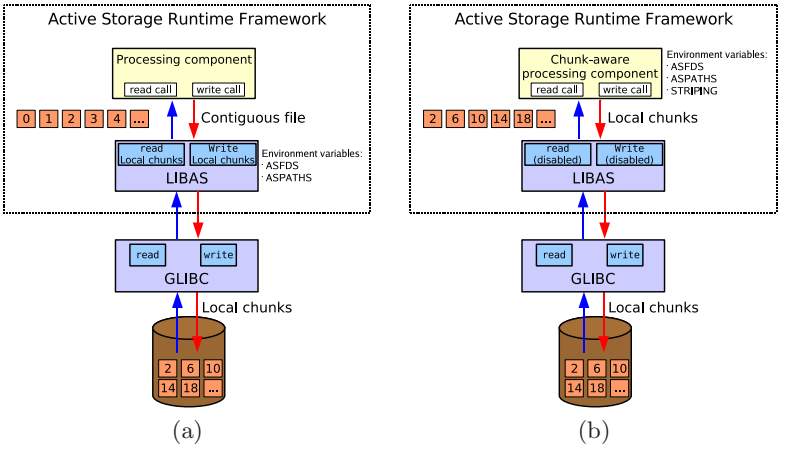

Fig. 2. The `libas` library. (a) Transparent access, and (b) non-transparent access to striped files.

With fixed-length, chunk-aligned records, Active Storage can provide transparent access to the striped files by exposing the different chunks of a file stored in the same node as a single, contiguous file. Active Storage uses the aforementioned `libas` dynamic library to provide that transparent access. The mechanism implemented is depicted in Figure 2(a). Using several environment variables, the Active Storage Runtime Framework passes to the `libas` library striping information about the striped files which must transparently be accessed. When a processing component opens a file, `libas` checks if the file is one of the files it must manage. If so, `libas` will intercept any operation on that file in order to only read and write the local chunks of the file.

The single, contiguous file's abstraction provided by Active Storage allows a user to run programs which access files sequentially as processing components, even when the input and output files are striped.

4.2 Striped Files with Unaligned Records

There are cases where the above requirement of fixed-length, chunk-aligned records is not met. In those cases, Active Storage cannot provide a transparent access to the striped files. Instead, our system passes striping information to the processing components (see Fig. 2(b)), which must decide by themselves which data in the local chunks, and other remote chunks, they have to access.

Our design provides a third option to deal with striped files which can not be transparently accessed. In some cases, the amount of output data is much greater or smaller than the amount of input data. Although the input and output files could use the same striping, the output records would not be typically written to the same storage node as their corresponding input records. In those cases, we can run a processing component per chunk of the input file. The output data can then be written to a local non-striped file whose name will have the absolute number of the input chunk. Certainly, there will not be a single "output file" but a set of sub-files, which will be accessed by a subsequent process in the order given by the sequence of chunk numbers. In this way, the write operations will be local, which is one of the primary design objectives for Active Storage.

4.3 Mapper Component for Processing of Complex Data Formats

Until now, we have assumed that all the data in the input files is processed. If Active Storage processing applies only to a part of the data in the files, it becomes necessary to augment the Active Storage architecture with a new element, the *mapper component*, that interacts with `asmaster`. This new component is a program which receives a file and its striping information as arguments, and return a list (a *map*) of the storage nodes which contain the data to process.

The mapper program also allows us to handle files with special data formats. For example, in our project we developed a mapper for netCDF [15] files to deal with climate applications. Our netCDF mapper receives as arguments a file, its striping information and the variable that the processing components will access; then, it reads the netCDF header of the file to locate the variable in the file, and uses the striping information to, finally, build the map of storage nodes. With this approach, the processing is optimized because only the relevant storage nodes run processing components (see Figure 3). Once `asmaster` launches the different processing components, these again read the netCDF header of the file and use its striping information to locate the part of the file that they must process.

5 Evaluation

Our system under test has 17 nodes: 8 compute nodes, 1 MDS/MGS server and 8 OSTs (the storage nodes in Lustre). All the nodes are identical, and their hardware elements are: two Dual Core Intel Xeon 5160 CPUs, 4 GB of RAM, a Dual Port MT25208 NIC, a Seagate ST3250624AS (250 GB) for the OS, and a Seagate ST3500630AS (500 GB) for Lustre. We also use a 24-ports MT47396 Infiniscale-III switch, Lustre 1.6.1 and Linux kernel 2.6.16.42.

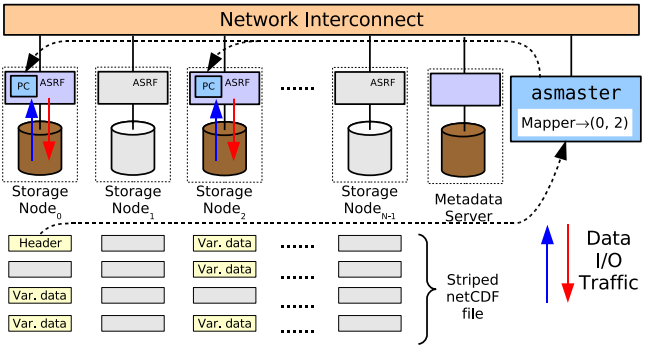

Fig. 3. The *mapper* application for netCDF files

We have compared the performance achieved by Active Storage with that obtained by a system without Active Storage. Our comparison uses two applications. The first ones, DSCAL, reads an input file of doubles in native format (8 bytes), and multiply every number by a scalar. The resulting doubles are written to an output file. In our experiments, this application has to process 16 input files of 1 GB each, and produce 16 output file, also of 1 GB each.

The second application, ClimStat (Climate Statistics), is used to compute the mean of a float variable in a netCDF file. The program receives the netCDF file, the variable and the striping of the file. With the striping information, the program determines which values of the variable are locally stored in its storage node, adds those values, and returns the sum and the number of added values (i.e., the output data is only a couple of numbers). In our experiments, ClimStat has to add the values of a record variable with 13,762,560 float elements (52.5 MB of data) split into 5 records which are spread across a 5.3 GB netCDF file. In total, there are 16 netCDF files and 840 MB of data to process. These files are the output of a Global Cloud Resolving Model simulation [16].

We have evaluated the performance of Active Storage for 1, 2, 4, and 8 storage nodes. Every file is striped across all the OSTs with a stripe size of 1MB. The achieved results have been compared with those obtained by a system without Active Storage and 1, 2, 4, and 8 compute/storage nodes With Active Storage, DSCAL and ClimStat run as processing components on the storage nodes. It is interesting to note that we use the same DSCAL code, and the same ClimStat code, for all the processing components; it does not matter the number of storage nodes. Without Active Storage, the applications run on the compute nodes.

5.1 Experimental Results for DSCAL

Figure 4(a) shows the overall execution time for DSCAL. This is the time to carry out the entire job, either in the compute or storage nodes, and includes the time taken to launch the different processing components, transfer data (when needed), and wait for the completion of the processes.

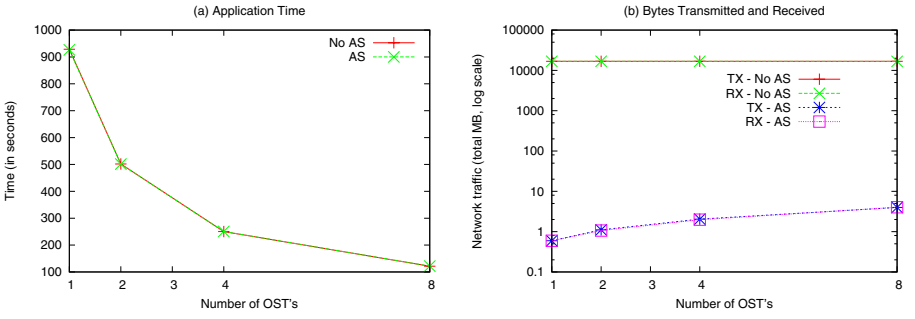

Fig. 4. Results for DSCAL in a non-saturated system

As we can see, there are no noticeable differences between running the application on the compute nodes and on the storage nodes. Since the number of nodes is small, compared to the disk speed our Infiniband interconnect does not become a bottleneck, hence the result. However, the absence of differences is important because it means that the overhead of the processing components on the storage nodes is small, and does not downgrade the performance of Lustre. Another important benefit, is that Active Storage allows to move the processing off the critical path: the application, after initiating the Active Storage processing, can continue instead of waiting for the time consuming I/O operations.

We can also see that the application time is divided by two when the number of OSTs doubles. This proves that Active Storage provides DSCAL with an environment where its performance scales with the number of storage nodes. This also proves that our proposal can handle stripes files efficiently.

Figure 4(b) displays the bytes transmitted and received in DSCAL. Unlike the application time, there is a huge difference between a system with Active Storage and another without it. With Active Storage, the network traffic is very small, while it can be very high without Active Storage, where the interconnect can become a bottleneck if there are hundreds or thousands of nodes. We can also see in the figure that the bytes transmitted and received are roughly the same. This result is expected because the application reads from and writes to files of the same size. Note that the network traffic smoothly increases with the number of nodes. This is mainly due to the meta-data operations (when processing a file, all the nodes have to obtain its meta-data information from the MDS server).

5.2 Experimental Results for ClimStat

In general, the results obtained by ClimStat are quite similar to those achieved by DSCAL, but there are some details to be explained. By coincidence, in our experiments all the storage nodes roughly have the same amount of data of the variable whose mean must be computed. Therefore, all of them take part in the task. With a larger number of storage nodes, other files or a different variable, this might not be true.

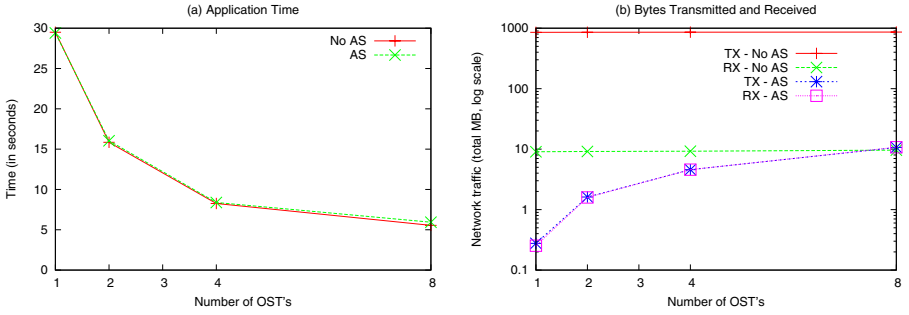

Fig. 5. Results for ClimStat in a non-saturated system

The application time for ClimStat is presented in Figure 5(a). In this case, the application times are much smaller than in DSCAL. However, similarly to DSCAL, there are no noticeable differences between running the application on the compute nodes and running the application on the storage nodes because the Infiniband interconnect does not become a bottleneck.

The network traffic is shown in Figure 5(b). As we can see, the amount of bytes received by every storage node is small, both with and without Active Storage. The reason is, as we have explained before that ClimStat produces only two numbers as output, which represent a few bytes. Therefore, we cannot expect any appreciable difference. The findings are different with respect to the number of bytes transmitted. In a system without Active Storage, the storage nodes have to send the data to the compute nodes. With Active Storage, however, this network traffic is very small.

Figure 5(b) also shows that, with Active Storage, the network traffic smoothly increases with the number of nodes. Like in DSCAL, this is due to the meta-data operations, but, for ClimStat, it is also due to the reading of the netCDF header. This header is small in our case, and is contained in only one storage node which has to send a copy to the other nodes. Without Active Storage, the header must also be sent to all the compute nodes, although its impact is small because of the high network traffic caused by the transmission of the temperature data.

The most important conclusion from the evaluation of the ClimStat application is not only that Active Storage is able to leverage the processing power of the storage nodes and considerably reduce the network traffic at the same time, but also that it can process data stored in files with a given (complex) format, like netCDF, and not evenly distributed among the storage nodes.

6 Conclusions

The current paper addresses two important requirements that were not considered in the previous work on Active Storage: striped files and complex scientific data formats such as netCDF. We have described how these important requirements can be addressed while achieving high performance. The experimental results on a Lustre file system not only show that our approach can reduce the

network traffic to near zero, and profit the extra computing capacity offered by the storage nodes at the same time, but also that it provides an efficient treatment of striped files and is able to manage files with complex data structures. We have also improved the usability aspects of Active Storage, providing a scientific-friendly environment to specify, in a simple way, the job to carry out.

References

1. Cluster File Systems Inc.: Lustre: A scalable, high-performance file system (2002), `http://www.lustre.org`
2. Carns, P.H., Ligon III, W.B., Ross, R.B., Thakur, R.: PVFS: a parallel file system for Linux clusters. In: Proc. of 4th Annual Linux Showcase and Con., pp. 317–327 (2000)
3. Felix, E.J., Fox, K., Regimbal, K., Nieplocha, J.: Active Storage processing in a parallel file system. In: Proc. of the 6th LCI International Conference on Linux Clusters: The HPC Revolution (2006)
4. Piernas, J., Nieplocha, J., Felix, E.J.: Evaluation of Active Storage strategies for the Lustre parallel file system. In: Proc. of 2007 Supercomp. Conf. (SC 2007) (2007)
5. Acharya, A., Uysal, M., Saltz, J.: Active disks: Programming model, algorithms and evaluation. In: Proc. of the ACM ASPLOS Conference, pp. 81–91 (1998)
6. Chiu, S.C., keng Liao, W., Choudhary, A.N.: Design and evaluation of distributed smart disk architecture for I/O-intensive workloads. In: Sloot, P.M.A., Abramson, D., Bogdanov, A.V., Gorbachev, Y.E., Dongarra, J., Zomaya, A.Y. (eds.) ICCS 2003. LNCS, vol. 2660, pp. 230–241. Springer, Heidelberg (2003)
7. Keeton, K., Patterson, D.A., Hellerstein, J.M.: A case for intelligent disks (IDISKs). SIGMOD Record 24(7), 42–52 (1998)
8. Riedel, E., Gibson, G., Faloutsos, C.: Active storage for large-scale data mining and multimedia. In: Proc. of the 24th Int. Conf. on Very Large Data Bases (VLDB), pp. 62–73 (1998)
9. DeWitt, D.J., Hawthorn, P.: A performance evaluation of database machine architectures. In: Proc. of the 7th Int. Conf. on Very Large Data Bases (VLDB), pp. 199–214 (1981)
10. Lim, H., Kapoor, V., Wighe, C., Du, D.H.: Active disk file system: A distributed, scalable file system. In: Proc. of the 18th IEEE Symposium on Mass Storage Systems and Technologies, San Diego, pp. 101–115 (2001)
11. Gibson, G.A., Nagle, D.F., Amiri, K., Chang, F.W., Feinberg, E.M., Gobioff, H., Lee, C., Ozceri, B., Riedel, E., Rochberg, D., Zelenka, J.: File server scaling with network-attached secure disks. In: Proc. of the 1997 ACM SIGMETRICS Intl. Conf. on Measurement and Modeling of Comp. Systems, pp. 272–284 (1997)
12. Mesnier, M., Ganger, G., Riedel, E.: Object-based storage. IEEE Communications Magazine 41(8), 84–90 (2005)
13. Schlosser, S.W., Iren, S.: Database storage management with object-based storage devices. In: Proc. of the First International Workshop on Data Management on New Hardware (DaMoN) (2005)
14. Du, D.H.: Intelligent storage for information retrieval. In: Proc. of the Intl. Conference on Next Generation Web Services Practices (NWeSP 2005), pp. 214–220 (2005)
15. Rew, R.K., Davis, G.P.: NetCDF: An interface for scientific data access. IEEE Computer Graphics and Applications 10(4), 76–82 (1990)
16. Schuchardt, K., Palmer, B., Daily, J., Elsethagen, T., Koontz, A.: IO strategies and data services for petascale data sets from a global cloud resolving model. Journal of Physics: Conference Series 78(012089) (2007)