A New I/O Architecture for Improving the Performance in Large Scale Clusters

L. M. Sánchez García[1], Florin D. Isaila[1], Félix García Carballeira[1],
Jesús Carretero Pérez[1],
Rolf Rabenseifner[2], and Panagiotis Adamidis[2]

[1] Computer Science Department, Universidad Carlos III de Madrid,
Av. Universidad 30, 28911 Leganés, Madrid, Spain
{lmsan, florin, fgarcia, jcarrete}@arcos.inf.uc3m.es
http://www.arcos.inf.uc3m.es
[2] High Performance Computing Center Stuttgart (HLRS), Universität Stuttgart,
Nobelstrasse 19, 70569 Stuttgart, Germany
{rabenseifner, adamidis}@hlrs.de
http://www.hlrs.de

Abstract. The technology advances made in supercomputers and high performance computing clusters over the past few years have been tremendous. Clusters are the most common solution for high performance computing at the present time. In this kind of systems, an important subject is the parallel I/O subsystem design. Parallel file systems (GPFS, PVFS, Lustre, etc) have been the solution used to obtain high performance I/O. Parallel file systems increase performance by distributing data file across several I/O nodes. However, cluster's size is increasing continuously, specially for compute nodes, becoming the I/O nodes in a possible bottleneck of the system.
In this paper, we propose a new architecture that solves the problem pointed out before: new hierarchical I/O architecture based on parallel I/O proxies. Those I/O proxies execute on the compute nodes offering an intermediate parallel file system between the applications and the storage system of the cluster. That architecture reduces the load on the I/O nodes increasing the global performance. This paper shows the design of the proposed solution and a preliminary evaluation, using a cluster located in the Stuttgart HLRS center.

Key words: Parallel File Systems, MPI-IO, High Performance Computing, Flash-IO, clusters

1 Introduction

The technology advances made in supercomputers and high performance computing clusters over the past few years have been tremendous. Total performance of all systems on the Top500 has increased by a factor of 10 every four years [1]. The number of solutions based on clusters is growing, because they are relatively

inexpensive and can use commodity parts readily available from many suppliers. One of the most important design issues for clusters is I/O performance. There is an enormous interest on the development of high performance storage systems because the number of applications with high I/O requirements is increasing continuously.

A typical architecture for a high-performance computing cluster (HPCC) consists of compute nodes, network, and storage systems. The number of components is increasing continuously, and for large scale clusters there is a huge unbalance between the number of computing nodes and I/O nodes used by the storage system. For example, NEC Cacau cluster of HLRS center has 200 compute nodes and only 2 I/O nodes. Another example, MareNostrum of Barcelona Supercomputing Center has 2406 dual processors as compute nodes and only 20 I/O nodes. The IBM ASCI Purple has at least 1400 8-way processors as compute nodes and 128 I/O nodes. That can convert the I/O subsystem in a bottleneck, as shown in Figure 1. That Figure shows the performance (time in seconds) obtained testing Flash-IO benchmark [2] (described in the evaluation section), for different numbers of compute nodes and using the storage system of NEC Cacau cluster. As we can see, the I/O system does not scale with the number of compute nodes.

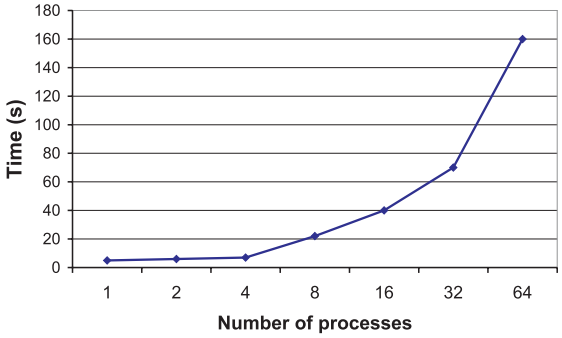

Fig. 1. Flash I/O results obtained with NEC cacau cluster using the cluster file system

This paper proposes a new I/O architecture for HPCC, based on hierarchical parallel *I/O proxies* that execute on computing nodes. Those parallel I/O proxies define an intermediate parallel file system between the applications and the storage system of the cluster. Our solution has two major goals: to increase data locality for applications, and reduce the number of I/O operations on the cluster storage system, in order to alleviate the possible bottleneck.

The paper is organized as follows: Section 2 describes the related work. Section 3 presents the new I/O architecture design. Section 4 describes the implementation of the system on NEC Cacau cluster. Performance evaluation is

presented in Section 5. Finally, Section 6 presents our conclusions and future works.

2 Related work

The use of parallelism in the file systems is based on the fact that a distributed and parallel system consists of several nodes with storage devices. The performance and bandwidth can be increased if data accesses are exploited in parallel [3]. Parallelism in file systems is obtained by using several independent server nodes supporting one or more secondary storage devices. Data are *striped* among those nodes and devices to allow parallel access to different files, and parallel access to the same file. Initially, this idea was proposed in [4] to increase the overall data throughput, striping data across several storage devices. This distribution technique is the basis for RAID systems [5].

Three different parallel I/O software architectures can be distinguished [6]: application libraries, parallel file systems, and intelligent I/O systems.

- *Application libraries* basically consist of a set of highly specialized I/O functions. Those functions provide a powerful development environment for experts with specific knowledge of the problem to model using this solution. Representative examples are MPI-IO [7], an I/O extension of the standardized message passing interface MPI.
- *Parallel file systems* operate independently from the applications, thus allowing more flexibility and generality. Examples of parallel file system are PVFS [8], Expand [9], GPFS [10].
- *An intelligent I/O system* hides the physical disk access to the application developer by providing a transparent logical I/O environment. The user describes what he wants and the system tries to optimize the I/O requests applying optimization techniques. This approach is used for example in Armada [11].

All these solutions are not enough for large scale clusters where the number of computing nodes is huge compared with the I/O nodes used by the storage system. For this kind of cluster, the current file systems for clusters can became the cluster I/O subsystem in a bottleneck.

There are other environments where the performance is improved with similar solutions as the one proposed here. *BAD-FS* [12] for GRID environment or *Scalable Lightweight Archival Storage Hierarchy (SLASH)* [13], and *High Performance Storage System (HPSS)* [14] for Mass Storage Systems propose the use of several disks as an intermediary cache system between the applications and the main storage systems. The main problem of those solutions is the difficulty to translate them to cluster environments.

3 Parallel I/O Proxies Architecture

Figure 2 shows the architecture of the proposed solution. The idea is to use the compute nodes for executing both applications and I/O proxies. An I/O proxy is

a file server that executes on a compute node and uses the local disk for storage. This solution is appropriated for clusters, because most clusters have compute nodes with several processors and local disks. We can define a virtual partition (VP) for each application by grouping several I/O proxies. As Figure 2 shows, a VP can be composed by nodes that could match or not the application ones, depending on the criteria used to form the VP.

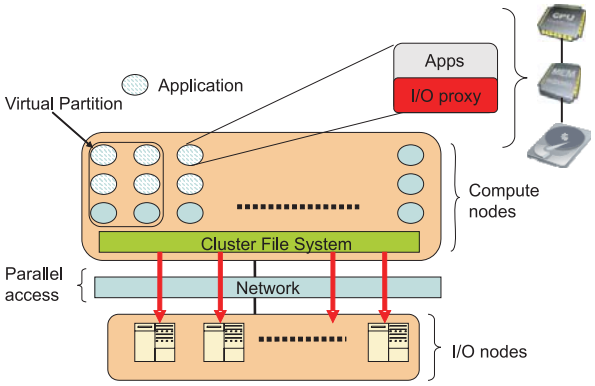

Fig. 2. Overview of the new proposed architecture

Let $C = \{c_1, c_2, \ldots, c_n\}$ the set of compute nodes. Let $A = \{a_1, a_2, \ldots, a_m\}$ the set of applications to run in C. Let $S = \{s_1, s_2, \ldots, s_p\}$ the set of cluster storage I/O nodes. We define $P = \{p_1, p_2, \ldots, p_q\}$ as the set of I/O proxies where p_i executes on c_i.

Formally, we define a virtual partition as a subset of proxy nodes, $VP \subseteq P$, strategically chosen to increase the data access parallelising degree for each application. The idea is to choose an optimal VP size, so that $\|S\| \ll \|VP\| \leq \|P\|$ and $\|S\| \ll \|VP\| \leq \|A\|$, in order to increase data locality for applications, and reduce the number of I/O operations on the cluster storage system.

When using the cluster file system, each file F of the application A is stored over S, and can be defined as $F = \{f_1, f_2, \ldots, f_q\}$, where f_i is the subfile stored in s_i. A subfile is the subset of the file data store at a particular I/O node. The system defines a function $f : F \Rightarrow S$ to map data to storage nodes. Our solution creates a VP for A (let's say VP_n), and stores F in another file F' into VP_n, using a new function $g : F' \Rightarrow VP_n$, where $F' = \{f'_1, f'_2, \ldots, f'_n\}$, and f'_j is the subfile stored at the p_j I/O proxy. As we have to store F in S, another function $h : VP_n \Rightarrow S$ is defined, so that $g \circ h : F \Rightarrow S$.

One important design aspect is to choose the VP size. Currently, we use an approach based on three parameters: the number of compute nodes (N) used by A, the size of each file F (F_s), and the size of local storage available (defined as $\frac{D}{k}$, where D the local disk size and k a constant defined by the system administrator or computed). The minimum number of I/O proxies used in VP

would be $nio = \frac{F_s}{\frac{D}{k}} = \frac{(F_s * k)}{D}$. The maximum would be N. We are still working on strategies to maximize the performance of $g \circ h$.

We introduce a restriction for the assignment of files to VPs. Two virtual partitions can not store the same file. So, the files can not be duplicated in different virtual partitions, avoiding possible coherence problems. It could be formulated as $VP(f') \cap VP'(f') = \emptyset$.

The virtual partition contributes in providing file service to applications. Applications access the files using the virtual partition as intermediate storage, combining different local disk for storage.

For example, MareNostrum has 20 I/O nodes with 140 TB and 2400 compute nodes with 40 GB of local disk. If we use a $k = 4$ of local disk for each I/O proxy, we could build a virtual partition with 20 TB of total storage.

Our solution has the following major features: transparent use of the I/O proxies, unique image of a file across the system, persistent storage on I/O proxies, independence of cluster file system, and adaptive virtual partition to application level like stripe size, data allocation, number of I/O proxies, etc. Figure 3 shows a case where the stripe size in the virtual partition is independent of the block size of the Cluster File System (CFS). Furthermore, the number of I/O proxies is larger than the number of I/O nodes of the CFS.

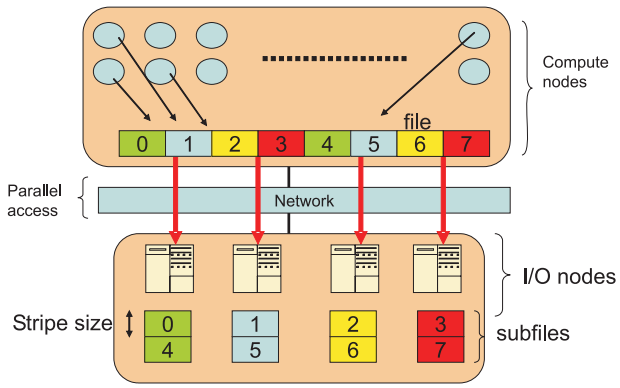

Fig. 3. Parallel files in I/O proxies are independent of the files stored in the CFS

3.1 I/O proxies

I/O proxies are responsible for managing and storing file data between the applications and the storage system of a cluster. We combine several I/O proxies for building a virtual partition where files are distributed. Main duties of an I/O proxy are: to serve data requests from compute nodes, to store data in their compute nodes until those data are stored in the CFS, and to provide to the compute nodes a fast access to the data.

I/O proxies use the local file system to store the data files, taking advantage of local cache, read ahead, and other optimizations of the local file system. That reduces the complexity of the I/O proxy and increases the performance. There are three major issues in I/O proxies: naming, metadata management, and I/O policies.

Our architecture assigns an internal id for the file F', by composing F, and the local partition used to store data. F' metadata is included as a small header at the beginning of each subfile, to provide fault tolerance. Metadata include the following information: file id, *stripe size*, local partition, number of I/O proxies, id of I/O proxies, and *base node*. The base node identifies the I/O proxy where the first block of the file resides and the *file distribution pattern* used. At the moment, we only use files with cyclic layout. We also have a unique *master node*, that can be different from the base node, to be used as primary node for metadata. This approach is similar to the mechanism used in the Vesta Parallel File System [15] and Expand [9]. To simplify the allocation and naming process, and to reduce potential bottlenecks, a virtual partition does not use any metadata manager, as in PVFS [8].

To obtain the master node of a file, the file name is hashed into the number of node: $hash(namefile) \Rightarrow I/Oproxy_i$

The hash function used in the current prototype is:

$$\left(\sum_{i=1}^{i=strlen(namefile)} namefile[i] \right) mod \ numProxies$$

The use of this simple approach offers a good distribution of masters. Table 1 shows the distribution (standard deviation) of masters between several I/O nodes. This results have been obtained using a real file system with 145,300 files. The results shown in this table demonstrate that this simple scheme allows to distribute the master nodes and the blocks between all NFS servers, balancing the use of all NFS servers and, hence, the I/O load.

Table 1. Distribution (standard deviation) of masters in different distributed partitions

Number of I/O nodes	**4**	**8**	**16**	**32**	**64**	**128**
Std. Dev.	0.43	0.56	0.39	0.23	0.15	0.11

I/O proxies ensures that data of a file are stored eventually in S. However, there are several I/O policies that can be applied to transfer data from VP to S. The system must decide when data are transferred from the I/O proxies to the storage system in case of write operations (or vice versa in case of read operations. The behaviour of the I/O proxy is different depending on the requested operation:

– *Reading:* if the file is not stored in the virtual partition, the data is transferred from the storage system to the virtual partition using one of the next policies: on demand (taking the data by demand), on application start (all the data is transferred from the storage system to the virtual partition) or when the file is open.
– *Writing:* the write operations use the data stored in a virtual partition to write them to the storage system. Data are transferred to the storage system using one of the next policies: write on close, write through, delayed write or flush on demand.

The storage of the I/O proxies is limited. When space is required for storing new files in VP, we use a LRU replacement algorithm.

4 Implementation

This section gives an overview of the implementation of the I/O proxy components and their deployment using the Expand Parallel File System over NEC Cacau cluster. The new architecture is implemented as a user-level component in Expand software architecture [16]. Expand is transparently linked with the user application and provides parallel I/O. As shown in Figure 4, a new abstract device interface has been implemented below Expand to communicate with the I/O proxies using TCP/IP or another communication protocol when available.

Expand communicates with the I/O proxies by using the g mapping function defined before. I/O proxies are user level processes located on the compute nodes. They use local file system to store data on the local disk and cluster file system primitives to communicate with the CFS.

The configuration of a virtual partition is defined on the file configuration of Expand. In this file configuration declares the following parameters: the number of I/O proxies, the logical name of each I/O proxy, the stripe size, etc.

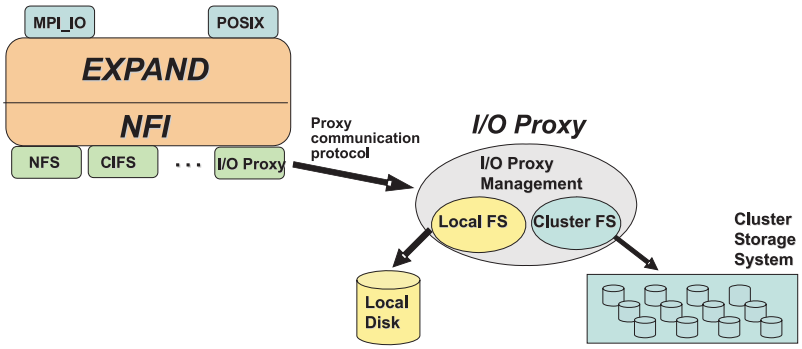

Fig. 4. Implementation of the I/O Proxy

In the current implementation, the transfer policies implemented are *delayed write* for write operations and *read at-begin* for read operations, without taking into account any hint of the I/O behaviour of the applications such as access pattern .

5 Evaluation

The existing prototype of this architecture was evaluated and compared with the I/O system of NEC Cacau cluster at the HLRS center by using FLASH-IO benchmark [2].This cluster has the following characteristics: 400 Intel Xeon EM64T CPU's, 160 nodes with 1 GigaByte of RAM memory and 40 nodes with 2 GigaByte of RAM memory, two frontend server to load balance, an Infiniband network interconnecting the compute nodes, a Gigabit Ethernet network for the communications between the compute nodes and the frontend server, and a Fiberchannel network to intercomunicate the frontend and the end-storage disks, NFS 2 protocol to access the disks of the end-storage and use a RAID 5 in the storage system.

FLASH-IO benchmark simulates I/O of FLASH Code. FLASH code is an adaptive mesh, parallel hydrodynamics code developed to simulate astrophysical thermonuclear flashes in two or three dimensioned space. FLASH-IO code produces a checkpoint file, a plotfile for centered data, and a plotfile for corner data. Those files are very different, because the first one is large and dense, as the last two ones are smaller and sparse.

The parameters used in the tests were the following: the number of I/O proxies is between 1 to 32 and the stripe size is between 1 KB to 2 MB.

Figure 5 shows time results for Flash-IO Benchmark using 32 processes. Each column shows the time spent to write the results of FLASH-IO to the CFS. The first column represents the time obtained using the CFS. The other columns represent the time obtained with our architecture and different configurations (tuning the different parameters of the Expand parallel file system cited in the before section, like I/O proxies (IOP), stripe size, etc). Time is obtained adding the time spent in the I/O proxies and the time consumed to flush data to CFS.

The results show that this new architecture obtained better performance than the current I/O system of the cluster. Notice that, for check pointing we get a speedup of at least 6 for all the configurations. For both plotfiles, we increase the performance by a factor of 20.

6 Conclusions

In this paper we have argued that the design of I/O systems for HPCC may create a bottleneck, because the storage and the compute nodes are unbalanced. We have described a new I/O architecture that increases the performance of the storage system of a HPCC, without hardware and system modifications. Using I/O proxies as intermediary between the application and the storage system of a HPCC and an abstraction named virtual partition composed of several I/O

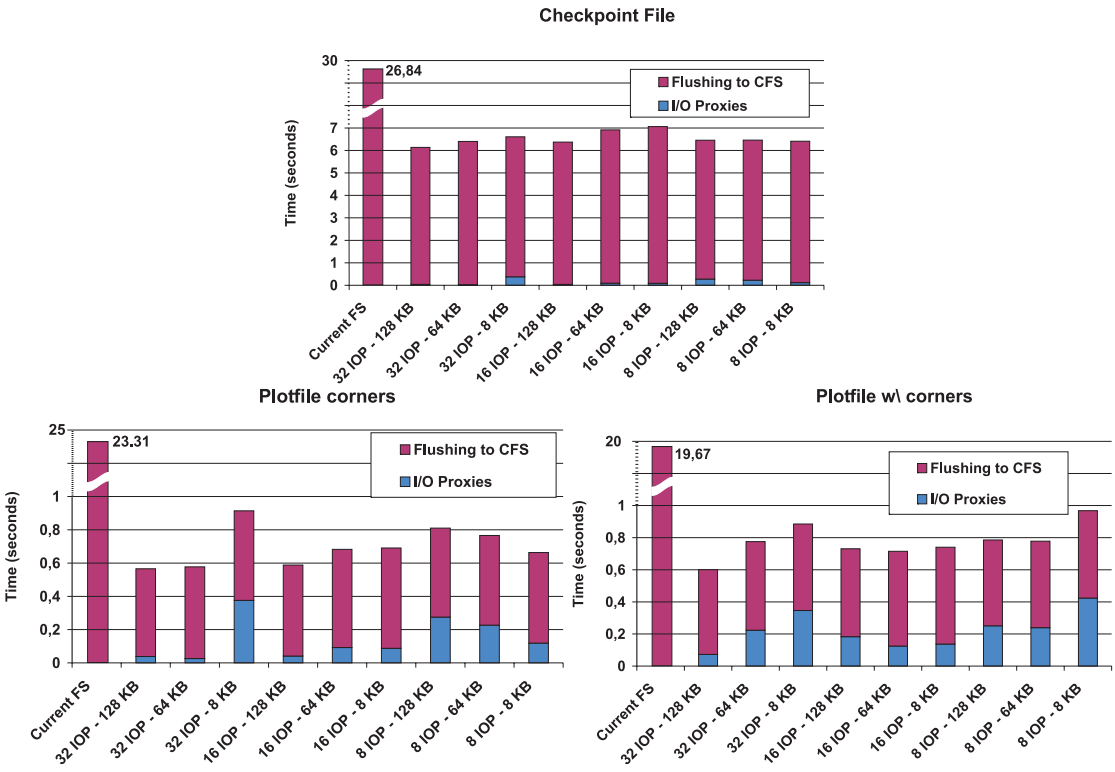

Fig. 5. Flash I/O Benchmark

proxies, most applications can obtain a better performance than if they use directly the storage system.

The evaluation performed in NEC Cacau cluster at the HLRS center, shows an important speedup (ranging 20 in some cases) for FLASH-IO benchmark. As this test is very demanding for I/O, we can expect even better results for more regular applications.

Some work is going on to test the system proposed in other HPCC and using more benchmarks. Moreover, we are working on virtual partition creation policies, I/O transfer policies, replacement policies, etc, to provide more flexibility and performance to the system.

Acknowledgements We would like to thank all people of HLRS center for their support on this project. This work was carried out under the HPC-EUROPA project (RII3-CT-2003- 506079), with the support of the European Community - Research Infrastructure Action under the FP6"Structuring the European Research Area" Programme). Also, this work has been supported by the Spanish Ministry of Education and Science under the TIN2004-02156 contract.

References

1. Vaughan-Nichols, S.J.: New trends revive supercomputing industry. **15**(2) (2004) 10–13
2. Fryxell, B., Olson, K., Ricker, P., Timmes, F.X., Zingale, M., Lamb, D.Q., MacNeice, P., Rosner, R., Truran, J.W., Tufo, H.: FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes. Volume 131. (2000) 273–334
3. del Rosario, J.M., Bordawekar, R., Choudhary, A.: Improved parallel i/o via a two-phase run-time access strategy. SIGARCH Comput. Archit. News **21**(5) (1993) 31–38
4. Salem, K., Garcia-Molina, H.: Disk striping. In: Proceedings of the Second International Conference on Data Engineering, February 5-7, 1986, Los Angeles, California, USA, IEEE Computer Society (1986) 336–342
5. Patterson, D.A., Gibson, G.A., Katz, R.H.: A case for redundant arrays of inexpensive disks (raid). In Boral, H., Larson, P.Å., eds.: Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, June 1-3, 1988, ACM Press (1988) 109–116
6. Schikuta, E., Wanek, H.: Parallel I/O. Volume 15. (2001) 162–168
7. MPI-Forum: Mpi-2: Extensions to the message-passing interface (1997)
8. Carns, P.H., Ligon III, W.B., Ross, R.B., Thakur, R.: PVFS: A parallel file system for linux clusters. In: Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, USENIX Association (2000) 317–327
9. Garcia-Carballeira, F., Calderon, A., Carretero, J., Fernandez, J., Perez, J.M.: The design of the Expand parallel file system. The International Journal of High Performance Computing Applications **17**(1) (2003) 21–38
10. Schmuck, F., Haskin, R.: GPFS: A shared-disk file system for large computing clusters. In: Proc. of the First Conference on File and Storage Technologies (FAST). (2002) 231–244
11. Oldfield, R., Kotz, D.: Armada: A parallel file system for computational grids. In: CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid, Washington, DC, USA, IEEE Computer Society (2001) 194
12. Bent, J., Thain, D., Arpaci-Dusseau, A., Arpaci-Dusseau, R.: Explicit control in a batch-aware distributed file system. (2004)
13. Nowoczynski, P., Stone, N., Sommerfield, J., Gill, B., Scott, J.R.: Slash - the scalable lightweight archival storage hierarchy. In: MSST. (2005) 245–252
14. Teaff, D., Watson, D., Coyne, B.: The architecture of the high performance storage system (hpss). In: Goddard Conference on Mass Storage and Technologies. (1995)
15. Corbett, P.F., Baylor, S.J., Feitelson, D.G.: Overview of the vesta parallel file system. SIGARCH Comput. Archit. News **21**(5) (1993) 7–14
16. Calderón, A., García, F., Carretero, J., Pérez, J.M., Fernández, J.: An implementation of mpi-io on expand: A parallel file system based on nfs servers. In: Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, London, UK, Springer-Verlag (2002) 306–313