# Evaluating the Performance of the dNFSP File System

Rodrigo Kassick[1,*], Caciano Machado[1,†], Everton Hermann[1,‡],
Rafael Ávila[1,2,‡], Philippe Navaux[1], Yves Denneulin[2]

[1] Instituto de Informática/UFRGS
Caixa Postal 15064
91501-970 Porto Alegre – Brazil
Email: {avila,navaux}@inf.ufrgs.br

[2] Laboratoire ID/IMAG
51, avenue Jean Kuntzmann
38330 Montbonnot-Saint Martin – France
Email: *First.Last*@imag.fr

## Abstract

*Parallel I/O in cluster computing is one of the most important issues to be tackled as clusters grow larger and larger. Many solutions have been proposed for the problem and, while effective in terms of performance, they usually represent a considerable amount of hacking into a "traditional" Beowulf cluster installation. In this paper, we investigate a parallel solution based on NFS, which reduces the level of intrusion in the file server installation, keeps the client side untouched, and still provides an improved level of performance and scalability for parallel applications. We compare our proposal to other existing file systems using known benchmarks, and demonstrate that it is a valid alternative for general-purpose cluster computing.*

## 1. Introduction

High performance file systems are nowadays a special source of attention for researchers in cluster computing, since technology and prices allow one to build machines that are each time larger and larger in size and capacity. In the November 2004 TOP500 list[1], 294 parallel machines are classified as clusters, and many of them feature more than 1000 processors. This trend has led researchers and vendors around the world to adapt existing (mostly commercial) high-performance I/O solutions as well as to design and implement new parallel file systems, which might be better suited to the "commodity-of-the-shelf" approach of Beowulf cluster computing [14].

In any case, solutions for the performance and scalability of cluster file systems generally present a significant amount of *intrusion* in a standard cluster installation, meaning that new tools, daemons and/or kernel drivers must be compiled, installed and configured. Traditional Beowulf software like the GNU/Linux system, TCP/IP, NFS and the like are well-known to system administrators and management tools, representing a good share of confidence and stability in a typical cluster installation. In this way, we believe that a solution for parallel I/O that could be built up from such established systems might be of considerable interest to cluster computing facilities.

Following this approach, in this paper we present a performance evaluation of *dNFSP* [1], an extension of the standard NFS file server intended for an improved level of performance and scalability on clusters while still maintaining compatibility with the standard NFS clients available on every Unix system. Our main goal is to evaluate the feasability of dNFSP as an alternative for cluster file systems.

We begin with a brief introduction to dNFSP and its principles, as well as its relation to other solutions for parallel I/O. Section 3 then describes the benchmarks and criteria used to evaluate dNFSP, followed by Section 4 that presents and discusses the obtained results. Finally we present in Section 5 our conclusions and future activities.

## 2. dNFSP – Distributing NFS

NFSP [8] is a project started at the *Laboratoire Informatique et Distribution*, in Grenoble, France, with the goal of providing an improved level of performance to a standard NFS [3] server. By following this ap-

∗ PIBIC/CNPq research assistant
† Dell/UFRGS research assistant
‡ Work supported by HP Brazil
1 http://www.top500.org

**Figure 1. NFSP architecture**

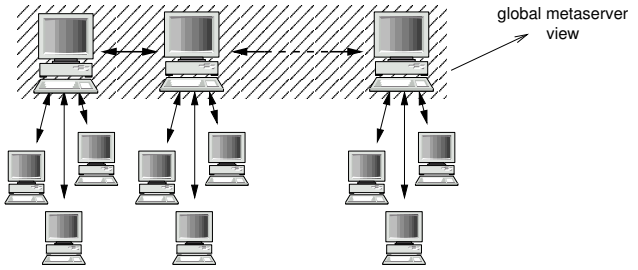


**Figure 2. Meta-servers are replicated in dNFSP**

proach, the project expects to reach a level of performance and scalability suitable for many parallel applications, and at the same time keep the intrusion level in relation to a traditional cluster installation to a minimum.

### 2.1. Design of NFSP

The principle behind NFSP is inspired in PVFS [4]. The functionality of the NFS server is split into two parts: a set of I/O nodes, or *iods* for short, which are responsible for storing and retrieving the data blocks that result from file striping, and a *meta-server*, which plays the main role in NFSP: it appears as the "normal" NFS server for the clients but, upon receiving a request, instead of reading/writing the data on its local file system, it forwards the request to the appropriate iods, which then respond to the clients as needed. Once several requests are received by the meta-server, they are forwarded to the iods, which can work in parallel, thus improving performance. Figure 1 illustrates one possible scenario. Iods can be run on client machines without any restriction. This allows one to easily benefit from the (usually forgotten) disk space on the compute nodes.

Several variations of NFSP have been implemented so far [9, 10], being based on both the user- and (Linux) kernel-level implementations of the standard NFS v2. One common characteristic among them is that the meta-server is one single process run on one of the nodes. This design allows for increased performance in the case of read operations, because most of the data involved in the complete operation are sent directly from the iods to the clients; however, in the case of writes, the data must be sent from the clients to the meta-server, and thus the original bottleneck remains the same.

### 2.2. dNFSP

In order to alleviate this problem, *dNFSP* [1] has been proposed as a variation of NFSP in which the meta-server is replicated onto several compute nodes. Figure 2 illustrates the new design.

In this approach, each replica of the meta-server works exactly as in the original model, receiving requests from the clients and forwarding them to the iods. However, one single replica serves only a subset of the clients, which see it as the only NFS server in the system (i.e. the client side is still unchanged). For example, if there are 4 meta-server replicas and 20 clients, each replica can be bound to 5 clients in order to balance the system. Now if all the clients need to access the server at the same time, several meta-server entry points (4, in the example) will be used instead of just one, thus allowing for an increased overall bandwidth also for write operations. Each replica is still capable of accessing the whole set of iods.

As a side-effect of meta-server replication, however, a new problem arises: to keep meta-data consistency among the several replicas. For example, if a client creates a new file, this file will exist on the meta-server that client is bound to, but not on the others, since meta-data are stored on each meta-server's local file system. This means that the new file will be visible for every client bound to the same meta-server, but not for the others.

In order to maintain meta-data coherence among the meta-server replicas without incurring in too much overhead, a mechanism based on LRC (Lazy Release Consistency [7]) is used. In this mechanism, meta-data are made consistent (which means to copy some information from one remote meta-server to another) only when effectively needed by the clients. This situation is basically detected when a client tries to access a file that *apparently* does not exist (*lookup* function in NFS). At this moment the meta-server in ques-

tion starts searching for the replica that contains the file, and copies it when found. We rely on the fact that accesses to really inexistent files should not occur (at least not often) in a parallel application. More details on the model are presented on another paper [1].

## 3. Related Work

The problem of efficient I/O in parallel computing arises every time the number of compute nodes grows beyond a few nodes (naturally, given that the application needs it). In this way many solutions for high performance storage in clusters and parallel machines in general have been proposed. Two main approaches seem to exist. The first one achieves improved performance by making use of dedicated hardware like high speed data links (e.g. fiber optics), redundant storage, non-volatile RAM, and several combinations thereof. This approach is mostly used by the file systems of commercial parallel machines, such as IBM GPFS [12] and Sistina GFS [13, 11]. Following another direction, file systems such as PVFS [4] and Lustre [5] try to obtain better performance by distributing the file system functionalities among the compute nodes. Since this approach generally does not require the use of any special kind of hardware, it is better suited to the philosophy behind cluster computing.

dNFSP belongs in the second group. The main difference in relation to other distributed file systems is the replicated meta-server design and its LRC-based coherence mechanism, which allows for reduced overhead operation in applications with low meta-data profile. Another aspect that distinguishes NFSP in general is the NFS compatibility. By building the system upon the traditional NFS foundation, we aim at obtaining a system with well-known configuration and management procedures, thus reducing the impact of introducing a new technology, while being able to keep with a good level of both performance and scalability which are suitable for many parallel applications.

In order to better evaluate the performance levels of dNFSP, we have carried out a series of experiments with traditional file system benchmarks found in the literature. Such experiments and test-bed are described next.

## 4. Description of the Experiments

The goal of our analysis is to evaluate the level of performance presented by dNFSP in comparison to a real cluster file system, as well as to measure the impact of our extensions to the traditional NFS server in comparison to an unmodified version of that system. It is important to clarify that we do not consider NFS a proper high-performance cluster file system, but rather recognise that it is widely used for that purpose on environments where parallel I/O applications are not dominant. In this way, we try to evaluate dNFSP in both situations, by using benchmarks and applications for both general-purpose and high-performance file systems.

### 4.1. Distributed Andrew Benchmark (DAB)

In order to evaluate the performance of the proposed filesystem, as well as the overhead caused by splitting the files through IODs and replicating metafiles in metaservers, we have created a variation of the well-known Andrew Benchmark, here called Distributed Andrew Benchmark.

The original Andrew Benchmark was conceived to test the performance of the Andrew File System (AFS) [6], also a distributed file system. It tries to simulate the load that would be achieved in normal use of the file system with several users connected. The original benchmark is meant to be run on a single machine. In our modified version, we intend to evaluate the performance of several nodes accessing the distributed file server. This is done by executing several instances of the benchmark on different machines (here called *clients*).

In order to measure the times of meta-server synchronization and data I/O independently, the benchmark was modified to make all clients execute phases in a coordinated manner, i.e., all clients must complete a phase before proceeding to the next one. Also, as a guarantee that no results are masked by data caching or buffering, the remote file system is mounted and unmounted respectively at the beggining and at the end of each phase.

The modified benchmark is composed of five phases:

**mkdir** Creates the directories which will be used in the next phases.

**cp** Each client makes a copy of the original tree in its own directory in the shared file system.

**stat** The benchmark performs a *stat* in each file in the client's directory.

**read** Reads all the contents of all files in the client's directory.

**make** Compiles an average size program (in the case of this test, POV-Ray[2]).

2 http://www.povray.org

Phases 1 & 3 focus on how efficient and scalable the file system is when accessing the *metafiles*. Phases 2 & 4 try to evaluate the performance of data access. Phase 5 tries to measure the file system performance in a situation where both metafiles and data are needed.

### 4.2. The NAS/BTIO Benchmark

The NAS Parallel Benchmarks (NPB) are a set of applications based on Computational Fluid Dynamics (CFD) designed to help evaluate the performance of parallel supercomputers. There are several flavours of the NPB, allowing to evaluate different aspects. The BTIO benchmark tool is the responsible to evaluate the storage performance. It is an extension of the BT benchmark [2] which is based on a CFD code that uses an implicit algorithm to solve the 3D compressible Navier-Stokes equations. The BTIO version of the benchmark uses the same computational method, but with the addition that results must be written to disk at every fifth time step. There are different versions of BTIO, which are described below:

- BTIO-full-mpiio: uses MPI-IO file operations with *collective buffering*, which means that data blocks are potentially re-ordered previously to being written to disk, resulting in coarser write granularity
- BTIO-simple-mpiio: Also uses MPI-IO operations, but no data re-ordering is performed, resulting in a high number of seeks when storing information on the file system
- BTIO-fortran-direct: This version is similar to simple-mpiio, but uses the Fortran direct access method instead of MPI-IO
- BT-epio: In this version each node writes in a separate file. This test gives the optimal write performance that can be obtained, because the file isn't shared by all the processes, so there is no lock restrictions. In order to compare with other versions, the time to merge the files must be computed, as required by the Application I/O benchmark specification.

There is one restriction to run the test: the number of processes must be a perfect square $(1, 4, 9, 16, \ldots)$. To determine the amount of memory required for the run, a class of problem size must be chosen which represents the cubic matrix dimensions : Class A ($64^3$), Class B ($102^3$), Class C ($162^3$). The original code runs for 200 iterations and writes at every five iterations.

The tests were performed using only the epio version of the benchmark, since dNFSP was designed based on NFSv2 protocol, and MPI-2 IO requires NFSv3 to control the file access using locks. Also we have developed another version of BTIO to perform writes on every iteration instead of every five iterations, resulting in a more intensive write test.

## 5. Experimental Results

The experiments have been carried out on the LabTeC[3] cluster. This machine is composed of 20 nodes interconnected by Fast Ethernet, where each node features two Pentium III processors at 1 GHz, 1 GB RAM and one 18 GB SCSI hard disk. The operating system on all nodes is Debian GNU/Linux with kernel 2.4.26. All systems and applications have been compiled (where appropriate) with GCC v2.95.

### 5.1. The Analyzed File Systems

In order to compare the performance of dNFSP we have selected two representative file systems according to our evaluation criteria mentioned before:

**UNFS** The user level version of the widely used Network File System. We considered the values obtained with NFS [**?**] as a base of comparision to the parallel file system measures in the situation where cluster applications are not demanding high performance I/O. The results have been obtained with UNFS v2.2beta47.

**PVFS** The well-known parallel file system for the Beowulf world. The main aspect that differs PVFS from dNFSP is the fact that PVFS1 does not implement multiple meta-servers[4]. PVFS is developed jointly by the Parallel Architecture Research Laboratory (PARL) at Clemson University and The Mathematics and Computer Science Division at Argonne National Laboratory. The version used is PVFS 1.6.2.

In the PVFS and dNFSP experiments, a set of 4 nodes has been dedicated to the file server, each one holding one iod. The PVFS manager runs together with the first iod. For dNFSP, the 4 corresponding meta-servers also share the same nodes with the iods, and clients are evenly distributed among them. For each measured value, the mean of a series of five executions is presented.

---

3 Deployed within the context of a partnership between Dell Computers and the Instituto de Informática since 2002 (http://www.inf.ufrgs.br/LabTeC)

4 This feature is present in PVFS version 2, whose first stable version was only recently released, and as such we did not have experimental data at the time the paper was prepared

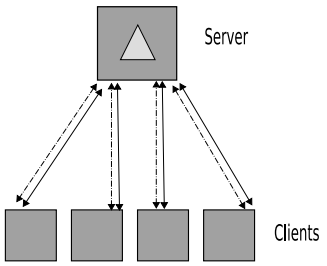


**Figure 3. UNFS Communication Model**

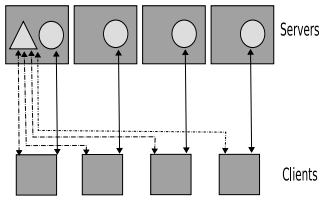


**Figure 4. PVFS Communication Model**

## 5.2. DAB Results

Figures 4 to 4.2 show the communication model for the three studied distributed file systems. The dashed lines show communication relative to file descriptor (metafiles in PVFS and dNFSP). The continuous lines show communication relative to data transfers. The triangles represent metaservers and the circles iods.

Figures **??** and **??** show the execution times for DAB using 8 and 16 clients respectively. The measured values for each system are grouped by phase for better comparison.

In phases 1 and 2, we can notice that the parallel file systems present high overhead when they execute operations like directories and file creation. In dNFSP, This overhead is mainly originated by the metafile replication mechanism, while PVFS has shown a poor performance in operation regarding metafiles.

The POVRay source code, used in phase 5 of DAB, has 1796 files and 74 directories. In the case of dNFSP, considering the test configuration with 16 clients, in

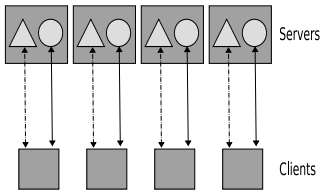


**Figure 5. dNFSP Communication Model**

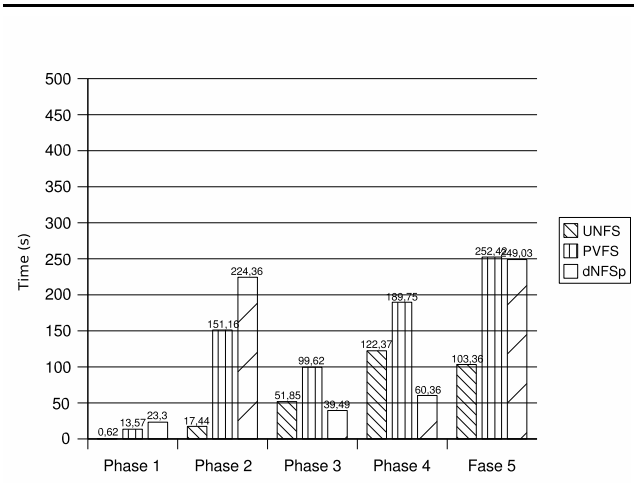


**Figure 6. Modified Andrew Benchmark Test (8 clients)**

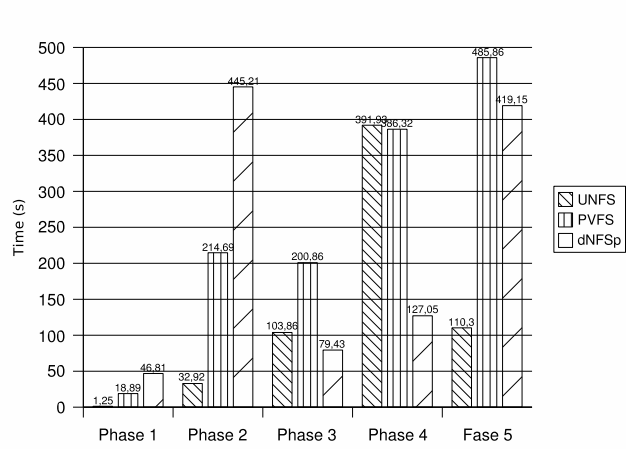


**Figure 7. Modified Andrew Benchmark Test (16 clients)**

the extreme situation when we are copying the source (phase 2), we realize a total of 89760 lookup operations in the metaservers to find the metafiles (that don't exist anywhere yet). This is because each metaserver searches for the metafile of each copied file on all other metaservers. The mechanism used to replicate the metafiles in the present version of dNFSP is relatively heavy and is one of the main aspects where we are working on. The values obtained in phase 5 reflect this behaviour.

According to Satyanarayanan [**?**], the reading operations are much more common than the writing operations, so the results of phases 3 and 4 represent considerable advantage to dNFSP. This advantage of multiple metaservers can be noticed clearly in phase 4. In this phase, the single NFS server is a serious bottleneck, which becomes more evident for a higher number of clients. PVFS, on the other hand, due to a problem with the handling of small files, presents poor performance.

In phase 5 we must consider that the compilation process consists actually in the alternation of the reading, compilation and writing. This means that the bottleneck of communication in the NFS server is not a major problem in this phase, because this alternation may result in a ad hoc synchronization over the NFS calls of the clients. Considering the current version of the parallel file systems and their inherent overhead to manage metafiles the parallel.

### 5.3. BTIO Results

This section presents results obtained using the BTIO benchmark to compare our file system with other related ones. BTIO, as described in section 3.2, is a variation of Computational Fluid Dynamics application where the intermediary results are written to disk during computation. Therefore, it requires a reactive and fast file system to achieve good performance.

As BTIO requires a perfect square number of process, the tests were executed using 1, 4, 9 and 16 clients. Figure 7 shows the average of the execution time obtained running the modified version of BTIO.

We can see that dNFSP was more effective in almost all the situations. It was between 0.4% slower and 33% faster than UNFS. Compared to PVFS the results were closer: our file system was between 1% and 6% faster than PVFS.

One reason for having a better performance compared to UNFS is the fact that dNFSP can be started using as many meta-servers and iods as needed, allowing to perform independent parallel writes. Another reason for better performance using dNFSP is the fact

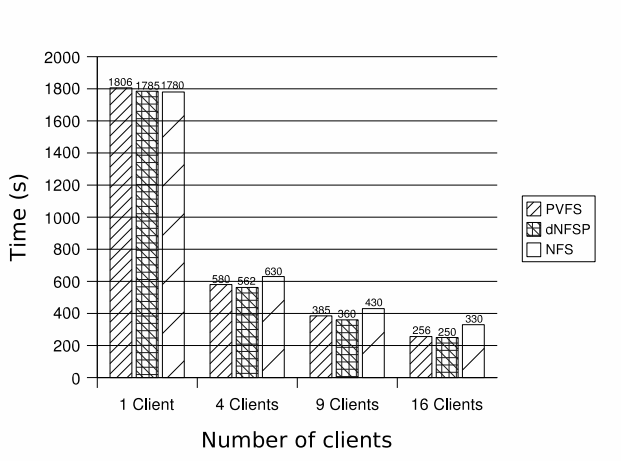


**Figure 8. Comparison between UNFS, PVFS and dNFSP using BTIO-epio benchmark**

that BTIO performs small write requests and PVFS has poor performance when used with small write block size, as stated by the authors on the PVFS website.

## 6. Conclusions and Future Work

Our experiments with dNFSP lead to the conclusion that the system is suitable for parallel applications on clusters, in the sense that an effective gain in read and write operations, comparable to those of a true parallel file system, can be observed. In our BTIO comparison, the levels of performance achieved with dNFSP are similar to those of PVFS, actually with a gain of up to 6% in execution time. Both systems perform clearly better than NFS with this benchmark, reaching 30% of advantage in some cases.

In the case of DAB, the performance of the three systems vary depending on each phase, with NFS sometimes showing better performance. This is due to the fact that DAB, being based on the Andrew Benchmark, mimics the load of a general-purpose file system, and not that of a parallel computing environment, especially by the frequent creation of new files and the handling of files of only a few kbytes. In dNFSP, the generation of lookup messages upon file creation is the main responsible for the decrease in performance, while for PVFS the problem lies on the handling of small data chunks.

Currently, in dNFSP, we are investigating a solution for the file lookup problem. We do not expect the file creation case to be very frequent, but the copying of remote meta-data is realistic, and may cause signif-

icant overhead if there are many files involved. A possible solution will be to copy several files (e.g. all the files on the same directory) on each update instead of just one, in order to try to anticipate future requests.

As a future work, we consider the possibility of introducing a level of fault tolerance on the iods, so that file striping may be performed redundantly (e.g. as in RAID). Another possibility is to port the implementation into the kernel-level NFS, since this version provides less overhead due to fewer memory copies, which also improves performance.

## References


[1] R. B. Ávila, P. O. A. Navaux, P. Lombard, A. Lebre, and Y. Denneulin. Performance evaluation of a prototype distributed NFS server. In J.-L. Gaudiot, M. L. Pilla, P. O. A. Navaux, and S. W. Song, editors, *Proc. of the 16th Symposium on Computer Architecture and High Performance Computing*, pages 100–105, Foz do Iguaçu, Oct. 2004. Washington, IEEE.

[2] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The nas parallel benchmarks. *The International Journal of Supercomputer Applications*, 5(3):63–73, Fall 1991.

[3] B. Callaghan, B. Pawlowski, and P. Staubach. *NFS Version 3 Protocol Specification: RFC 1831*. Internet Engineering Task Force, Network Working Group, June 1995.

[4] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: a parallel file system for Linux clusters. In *Proc. of the 4th Annual Linux Showcase and Conference*, pages 317–327, Atlanta, GA, 2000. Best Paper Award.

[5] Cluster File Systems, Inc. Lustre: A scalable, high-performance file system, 2002. Available at http://www.lustre.org/docs/whitepaper.pdf (July 2004).

[6] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. Scale and performance in a distributed file system. *ACM Trans. Comput. Syst.*, 6(1):51–81, 1988.

[7] P. Keleher, A. L. Cox, and W. Zwaenepoel. Lazy release consistency for software distributed shared memory. In D. Abramson and J.-L. Gaudiot, editors, *Proc. of the 19th Annual International Symposium on Computer Architecture*, pages 13–21, Gold Coast, Queensland, Australia, 1992. New York, ACM Press.

[8] P. Lombard. *NFSP : Une Solution de Stockage Distribué pour Architectures Grande Échelle*. Thèse, Institut National Polytechnique de Grenoble, Grenoble, 2003.

[9] P. Lombard and Y. Denneulin. nfsp: a distributed NFS server for clusters of workstations. In *Proc. of the 16th International Parallel & Distributed Processing Symposium, IPDPS*, page 35, Ft. Lauderdale, Florida, USA, Apr. 2002. Los Alamitos, IEEE Computer Society. Abstract only, full paper available in CD-ROM.

[10] P. Lombard, Y. Denneulin, O. Valentin, and A. Lebre. Improving the performances of a distributed NFS implementation. In R. Wyrzykowski, J. Dongarra, M. Paprzycki, and J. Wasniewski, editors, *Proc. of the 5th International Conference on Parallel Processing and Applied Mathematics*, volume 3019 of *Lecture Notes in Computer Science*, pages 405–412, Czestochowa, Poland, 2003. Berlin, Springer.

[11] K. W. Preslan, A. P. Barry, J. E. Brassow, G. M. Erickson, E. Nygaard, C. J. Sabol, S. R. Soltis, D. C. Teigland, and M. T. O'Keefe. A 64-bit, shared disk file system for Linux. In *Proc. of the 16th IEEE Symposium on Mass Storage Systems*, pages 22–41, San Diego, California, Mar. 1999. Los Alamitos, IEEE Computer Society.

[12] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proc. of the Conference on File and Storage Technologies*, pages 231–244, Monterey, CA, 2002.

[13] S. Soltis, G. Erickson, K. Preslan, M. O'Keefe, and T. Ruwart. The design and performance of a shared disk file system for IRIX. In *Proc. of the 6th Goddard Conference on Mass Storage Systems and Technologies*, pages 41–56, College Park, Maryland, Mar. 1998.

[14] T. L. Sterling, J. Salmon, D. J. Becker, and D. F. Savarese. *How to Build a Beowulf: a Guide to the Implementation and Application of PC Clusters*. MIT, Cambridge, 1999.