Performance Evaluation of a Prototype Distributed NFS Server

Rafael B. Ávila[*], Philippe O. A. Navaux
Instituto de Informática/UFRGS
Caixa Postal 15064
91501-970 Porto Alegre – Brazil
Email: {avila,navaux}@inf.ufrgs.br

Pierre Lombard, Adrien Lebre, Yves Denneulin
Laboratoire ID/IMAG
51, avenue Jean Kuntzmann
38330 Montbonnot-Saint Martin – France
Email: *First.Last*@imag.fr

Abstract

A high-performance file system is normally a key point for large cluster installations, where hundreds or even thousands of nodes frequently need to manage large volumes of data. While most solutions usually make use of dedicated hardware and/or specific distribution and replication protocols, the NFSP (NFS Parallel) project aims at improving performance within a standard NFS client/server system. In this paper we investigate the possibilities of a replication model for the NFS server which is based on Lasy Release Consistency (LRC). A prototype has been built upon the user-level NFSv2 server and a performance evaluation is carried out.

Keywords: *Parallel file systems, NFS, Lazy Release Consistency, parallel I/O.*

1. Introduction

With the development of new technologies and consequently lower prices of cluster components, clusters may be seen growing each time larger and larger in size and capacity, for solving each time more and more complex problems. This continuous growth in cluster sizes implied a new issue: how to manage permanent storage.

Given the potential bottleneck presented by traditional, centralized systems like NFS [5], and the fact that hardware-based solutions do not fit well in the Beowulf philosophy, the research in the field of cluster file systems has followed the direction of distributing the file service across the cluster.

In an attempt to achieve a balance between performance and management simplicity, the NFSP project [11] proposes a distributed version of the traditional NFS server which better handles the load generated by multiple concurrent accesses from the compute nodes, but still remains compatible with the NFS client present on every Linux system. In this paper, we investigate the possibilities of a NFSP branch in which the NFS server is replicated across several nodes, and as a consequence consistency needs to be maintained among the various clients. Our main goal with this work is to propose an alternative for medium-to-large cluster systems which do not benefit from expensive data storage systems and whose administrators wish to keep management tasks within the established common-knowledge.

Next section introduces the NFSP model in details in order to provide a background for the work being developed; in Section 3, we present the proposed replication model, and in Section 4 the results of a performance evaluation. Section 5 brings an overview of the related research activities and makes some observations in relation to our work. Finally, Section 6 presents some final considerations and future activities.

2. NFSP

NFSP — NFS Parallel — is an extension of the traditional NFS implementation, developed at the ID/IMAG Laboratory[1] of Grenoble, France. It distributes the functionality of the NFS daemon over several processes on the cluster [11]. The idea behind NFSP is to provide an improvement in performance and scalability and at the same time keep the system simple and fully compatible with standard NFS clients.

Inspired in PVFS [6], NFSP makes use of I/O daemons running on several machines in the cluster in order to distribute regular files across a set of disks, in a mechanism referred to as *striping*. A file is “striped” by having its data separated into several blocks, which are then stored over several distinct machines. When the file is accessed, there can be potentially several data blocks being fetched at the same time, consequently increasing performance. In the

* Work partially supported by CAPES and HP Brazil

1 http://www-id.imag.fr

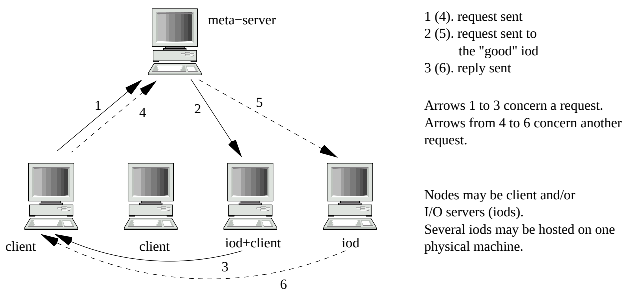
meta−server
1 (4). request sent
2 (5). request sent to the "good" iod
3 (6). reply sent
Arrows 1 to 3 concern a request. Arrows from 4 to 6 concern another request.
Nodes may be client and/or I/O servers (iods). Several iods may be hosted on one physical machine.
1
4
2
5
client
client
iod+client
iod
3
6

Figure 1: Request forwarding in NFSP

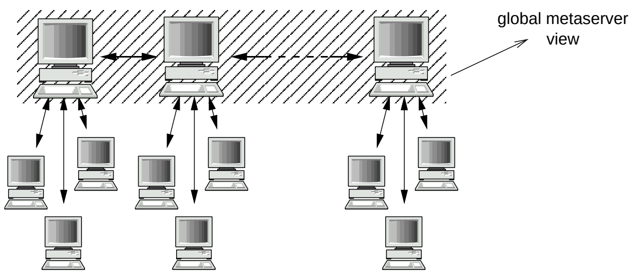

Figure 2: Distributed metaserver design

case of NFSP, the unit for striping is the NFS block (usually 8192 bytes).

The role of NFS server is played by *nfspd*, defined in NFSP as the *metaserver*. This daemon appears to clients as the regular NFS server; when a request for a given data block is received, the metaserver forwards the request to the corresponding I/O daemon, which in turn performs the operation and sends the desired information directly to the client[2]. Figure 1 illustrates this mechanism. The information regarding where and how each file is stored is kept on the metaserver's local file system, being part of the metadata (i.e. date/time, ownership, permissions, size, and the like).

This design of NFSP allows concurrent read operations to be effectively performed in parallel, given that distinct requests may correspond to distinct I/O daemons. However, write operations still need to be centralized at the metaserver, since the data come from the clients. This is one specific issue we are tackling with the model proposed in this paper, presented next.

3. The Server Replication Model

The goal of this work is to reach an improved level of performance in NFSP by replicating the metaserver. There are mainly two aims in this approach: offering several entry points for the clients (and hence let them use more bandwidth when doing write operations) and better balance the load onto several metaservers instead of only one.

3.1. Replicating nfspd

In the original NFS design, there is one single server (nfsd) to which all the client machines connect. An immediate approach to try to improve scalability in this scenario is to replicate the NFS server over several machines.

2 Techniques of IP spoofing may be necessary to achieve this goal of transparency from the clients' point of view, since some implementations require the answer to be originated from the server, not the I/O daemon.

The main idea is that a kind of grouping be established among the compute nodes, so that each metaserver instance serves only a given number of clients. For example, if the metaserver is replicated over 10 machines and there are 50 compute nodes, then each metaserver instance would be (considering a simple equal division of load) bound to 5 clients.

For each group of compute nodes accessing the same metaserver, the situation is similar to the original NFS model, i.e., all accesses to the file system are directed to one single server. Besides reducing the scalability problem, this approach allows a good deal of performance improvement if the connections to the metaservers are faster than that of the compute nodes, as often featured by Fast Ethernet switches that provide one extra Gigabit Ethernet port.

The metaservers run independently from but in cooperation with each other, by means of network communication, and together they form the notion of a single, global metaserver (see Figure 2). With that picture in mind, it is simple to observe that the goal here is to distribute the load of metadata operations, so that a single server does not become saturated in the face of multiple concurrent requests. In addition, this design may allow for future enhancements in terms of redundancy/fault tolerance.

3.2. Consistency Model

The direct implication of replicating the metaserver is to keep consistency between the multiple instances. In order to accomplish that task without incurring in too much overhead (e.g. increasing the network load with control messages), we rely upon a relaxed consistency model, supported by some characteristics that can be observed on a typical cluster computing environment.

The usual procedure for running a parallel program on a cluster is to allocate a given number of nodes, or partition, and then launch the application on it. Naturally, the size of such partitions and the allocation time vary depending on the application, but also according to the local policy of the site hosting the cluster, which depends on the users' status, priorities, previous executions, and the like. In addition, each user is normally using a private account, in which his files are stored, and thus the set of files being manipulated by an application is usually not the same as that of another application. This means that *we do not need to have all the files available everywhere all the time*.

As a coherence protocol for the distributed metaserver design, we make use of an adaptation of the Lazy Release Consistency protocol [8], or LRC, as used in the TreadMarks distributed shared memory system [2]. The basic principle is that the enforcement of coherence is postponed until the moment a client taking part in the protocol effectively needs it.

Similarly to TreadMarks, where consistency is checked only when a new client signals entry upon a shared segment (given that other clients may have modified it), our proposed system only checks for it when a client effectively accesses a file.

Coherence-checking messages are issued when some kind of error occurs, which is possibly an indication that the metadata regarding that specific file has changed. For example, if a file opening operation results in a `ENOENT` (non-existent file) error code, it is possible that the file has been created by a client bound to another metaserver, which means that the corresponding metadata only exists on that metaserver's local file system. In this case, the metadata needs to be copied to the current metaserver's file system, and then the operation (file opening) can proceed.

In some cases an operation can be executed even if the metadata is outdated. For example, if a client needs to read the first 4 kB of a file whose metadata indicate 10 kB of length, the operation can be successfully executed even if the file has already been enlarged (by some other client) to 100 kB. Notice that, even if the metadata is old, the file contents (i.e. the "real" data) are stored on the I/O daemons, which are shared among all the clients, and consequently have been updated by the operation that originally enlarged the file. In other words, even if the metadata is outdated, the client will not read stale data.

Naturally, some operations like file deletion need to be explicitly notified to all the metaservers in order to prevent access to data which is no longer valid. We understand, however, that this kind of operation occurs less frequently and mainly on application startup/shutdown, and thus should not cause significant impact on performance.

One important issue is to decide which metaserver to contact when metadata is missing/outdated. Several approaches can be thought of, like establishing a neighbourhood relationship between the metaservers and perform the search from the nearest to the farthest. Another possibility is to organize the metaservers hierarchically in a tree. This is one of our current study subjects.

4. Performance Evaluation

In order to validate the distribution model, we have implemented a prototype of the replicated metaserver based on the user-level implementation of NFSP. The main intention was to allow an evaluation of the "cache-miss" overhead imposed by the eventual need of fetching metadata (actually a metafile), from another metaserver, which is the basis of our model. Additionally, we wanted to observe the level of performance gain obtained by distributing the metaserver load among several replicated instances.

The original user-level NFSP implementation has been extended in order both to assign an address to each metaserver and to introduce a first level of communication between them. To keep the changes simple, we rely upon `rcp` to transfer metafiles between metaservers. This ensures not only that the contents of the metafile be transferred, but also the implicit metadata (owner, group, permissions, etc.) associated with it. On the other hand, we are aware that this mechanism is likely to impose a higher overhead than that of a dedicated protocol. Finally, only the case of missing metadata is being evaluated.

4.1. Cache-miss Overhead

The first evaluation on the implemented prototype refers to measuring the overhead of a cache-miss on a given metaserver and consequent metafile searching and fetching.

The metafile fetching mechanism must be based upon a decision algorithm which should indicate where to search. In our evaluation, in order to obtain the worst-case results, we forced each metaserver to perform a sequential search, i.e. starting from metaserver #0, then metaserver #1, and so on, up to metaserver #$(n-1)$ if necessary.

The benchmark we have run consists on creating 100 files on one of the client nodes, and then forcing the other nodes on reading all the files. As we want to measure the worst-case situation, the created files have 0-byte length (i.e., we want to measure metadata overhead only).

The execution has been carried out on the *i-cluster*[3] available at the ID Laboratory. We have configured the system with 1 iod (which is enough for this test), 16 metaservers and 16 clients (one for each metaserver).

The files for the benchmark have been created on the client mounting metaserver #15. Then, starting at client #14 and going down to client #0, we measured the time each node takes to read the 100 files (here *read* means `cat file > /dev/null`). Only one node executes at a time.

3 A 225-node cluster deployed within the context of a joint ID/INRIA/HP project; the nodes are Pentium III 733 MHz with 15 GB IDE disks connected by Fast Ethernet; all the nodes run Mandrake GNU/Linux with kernels 2.2 and 2.4

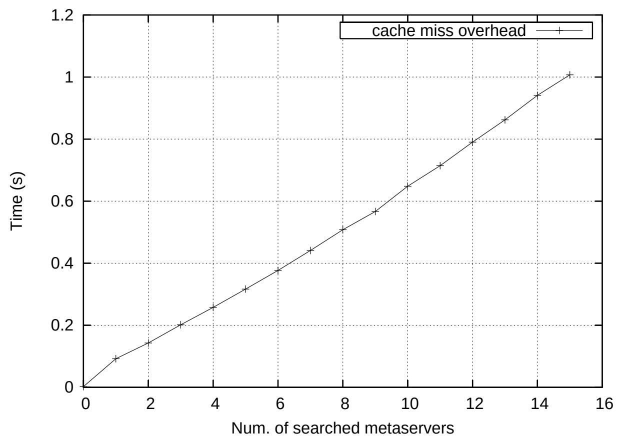

Figure 3: Cache-miss overhead according to the number of metaservers searched

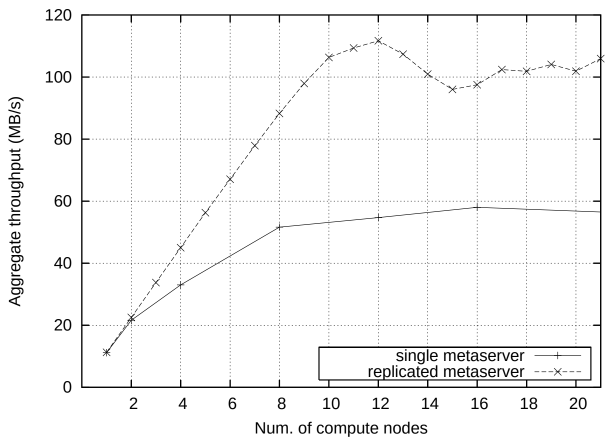

Figure 4: Aggregate read bandwidth obtained with the prototype, in comparison to the original model

Since the search always begins by metaserver #0, the first metaserver (#14) will try 15 times before finding the files on metaserver #15. The second metaserver to run will try one time less, since now the metafiles are available at #14. And so on, up to client/metaserver #0.

The obtained results are shown in Figure 3. Each result has been divided by 100 in order to represent the per-file overhead. We can observe that the curve grows in a practically linear slope, which corresponds to about 66 ms for each additional cache miss. This results in a worst-case delay of roughly 1 second in the case of 16 metaservers, which we consider a reasonable overhead to bear with. For example, considering the case where an application spans three metaservers, the maximum overhead to pay in this case would be of about 200 ms. In addition, the simplicity of using a simple `rcp` to fetch the remote metafile incurs in extra overhead, which we can expect to be reduced if a dedicated protocol is used.

4.2. Read Performance

We have also run performance evaluation experiments with the implemented prototype, to observe the level of gain that can be obtained in relation to the previous NFSP implementations. In the first case, we are evaluating distributed read performance. The experiment consists on reading a large file concurrently on all the clients. For that purpose we have created a 1 Gigabyte file on one of the nodes, and then concurrently launched the command `dd if=file of=/dev/null bs=1M count=1024` on the nodes (we did not force a metafile update on all the nodes prior to running the experiment, since the time is takes to read the data is much higher than that of the measured overhead).

The system, in this case, has been configured with 12 iods, 7 metaservers and 21 clients (which makes 3 clients per metaserver), in a total of 40 nodes.

The results shown in Figure 4 compare the aggregate read bandwidth obtained with the replicated metaserver model with that of the original model. As expected, having multiple metaserver instances distributes the load imposed by the concurrent accesses, and thus allows for a higher level of performance. In a previous work [11], we had observed that the performance was limited by the CPU on the metaserver saturating. In the replicated model we can observe that the performance reaches its maximum when the number of clients equals the number of iods, which was also expected.

As an alternative form of evaluation, we can consider a *per-client efficiency* measure in relation to the maximum network bandwidth achievable by each client. Considering 11 MB/s for each one (given the Fast Ethernet connection), we obtain practically 100% efficiency up to 10 clients, when then the curve flattens and tends to stabilize at around 110 MB/s, equivalent to 50% efficiency with 21 clients. As a simple comparison, a regular, centralized NFS server would reach only 4% with the same 21 clients, since the bandwidth would be always limited by the 11 MB/s network connection.

4.3. Write Performance

A similar experiment was also executed, but now with the clients writing a 1 Gigabyte file (each client writes a distinct file).

A comparison with the original NFSP implementation is not meaningful in this case, since writing is, as we stated before, centralized as that of the traditional NFS model.

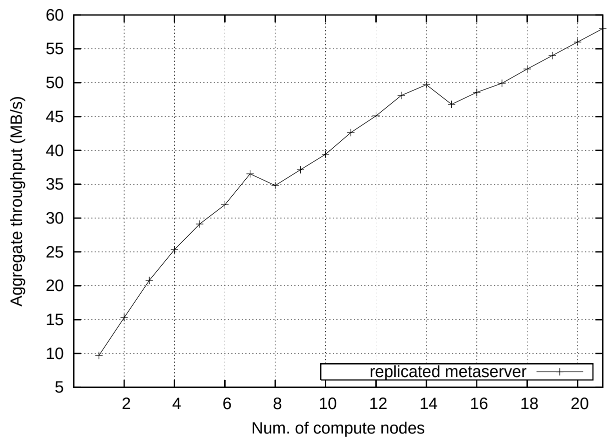

Figure 5: Aggregate write bandwidth obtained with the prototype

Measured results are presented in Figure 5. The execution has been carried out so that the maximum number of metaservers is always used (i.e. in the case of 7 clients, for example, each client connects to one distinct metaserver). The curve presents three distinct linear segments, separated by a "step" in performance every time the number of clients reaches a multiple of the number of metaservers. This is due to a single metaserver having to suddenly serve a higher number of clients than the others. For example, performance increases constantly up to 7 clients, when each metaserver is connected to at most one client. When we add the eighth client, metaserver #0 starts serving two clients, who share that server's bandwidth and consequently performances drops.

Again, we can observe the advantage of having multiple entry points for the clients, and specially in the case of writing. With one single metaserver, write bandwidth is primarily limited by the single network connection, and would remain constant at the lowest value shown (around 10 to 11 MB/s) for any number of clients. If we consider the per-client efficiency, the replicated model presents a mean value of about 33% with the configuration used, in contrast to 9% achievable with a centralized server.

5. Related Work

Several research projects exist which aim at a better performance for cluster file systems, using different approaches. Petal/Frangipani [9, 10] is a parallel file system built upon the concept of a *distributed virtual disk*, where a set of daemons running on a number of machines cooperate to form the view of a single storage device. The *Shared Logical Disk* [13] follows the same idea. Another approach is that of *Storage Area Networks*, in which there is a dedicated hardware support for parallel access to permanent storage. These are mostly commercial systems; examples are the IBM General Parallel File System [7], SGI XFS [15], and the OpenGFS (Global File System) [1].

It is undeniable that these systems are of recognized importance, however we do not consider that a comparison would be appropriate, since they follow different approaches and present distinct requirements. In the sequence, we present systems more directly related to the model we propose.

NFSP has its basis on the Network File System [5], or NFS, which is the *de facto* standard for distributed file sharing in the Unix world, and consequently has been naturally absorbed by the Beowulf cluster model since its first steps [14]. It is actually a protocol for transparent remote access from a client to a server's file system, and thus has not been devised for parallel computing. NFS raises a potential scalability constraint when clusters start to grow larger, due to its centralized server design. For this reason, many developments are currently being carried out in both the directions of enhancing NFS and providing new solutions [4, 12].

The *Berkeley xFS* [3] was a prototype "serverless" file system developed at the University of California at Berkeley from 1993 to 1995. It builds upon several research efforts developed at the time, like RAID, LFS (Log-structured File System), Zebra and Multiprocessor Cache Consistency, and thus presents a totally distributed design, where data and metadata are spread among the available machines (which may be all or part of the available computing resources) and can dynamically migrate. Though an interesting design and promising results, the project was interrupted with the end of project NOW, and hence, to our knowledge, no further development or porting to Linux has been carried out to the present days.

A possible successor of xFS in the Beowulf world is PVFS, *Parallel Virtual File System* [6], a joint project conducted by the Parallel Architecture Research Laboratory, at Clemson University, and the Argonne National Laboratory, both in the USA. The goal of PVFS is to provide a high-performance file system for the Beowulf class of parallel machines, being able to profit from commodity hardware. PVFS is able to deliver very good performance and provides different interfaces for applications: VFS, MPI-IO and a native one.

NFSP compares with such projects in the sense that it aims at performance and scalability for cluster computing. A different approach taken by NFSP, however, is that of keeping the changes to a traditional Beowulf system as minimal as possible. For example, the client side on a NFSP installation remains untouched, which eases the task of managing the cluster. Even though we are aware that a NFS-compatible file system may not be the best solution for certain applications, we also believe that the NFS approach might still be enough for many situations, and in such cases a simple solution close to the traditional Beowulf environment might be desirable.

6. Final Considerations and Future Work

Our intention with the replicated metaserver model for NFSP is to provide an additional level of performance while still keeping the simple design that guides the project. With the implementation of a prototype based on the user-level version of NFSP, we could observe an effective gain in performance in relation to a single, centralized NFS server, and with other "flavors" of NFSP. With minimum effort, we are able to improve by about 5 times the write performance and to double the read performance (in this case reaching almost 100% network efficiency), in comparison with the previous implementation.

Our evaluation of the adopted replication model is positive. The consistency mechanism based on LRC is efficient for applications running on a typical cluster environment, and the overhead imposed by eventual cache misses was considered low (around 0.066 seconds), even if a simple and heavier `rcp` mechanism was used.

Current and future activities in NFSP follow different approaches. As mentioned before, we are investigating different methods such as closest-neighbour and hierarchy for finding the correct metaserver in the case of metafile cache misses. Another activity concerns the integration of NFSP into the kernel-level NFS server, which, according to results already obtained, offers a significant gain in performance in relation to the user-level version. Concerning the metaserver replication, our next steps will be the design and implementation of a dedicated protocol for communication between the multiple servers, and further performance evaluations with real applications.

Acknowledgements

The authors would like to thank Caciano Machado and Rodrigo Kassick for their help in setting up the execution environment and running the experiments.

References

[1] The openGFS project, Apr. 2003. http://opengfs.sourceforge.net.

[2] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. Treadmarks: Shared memory computing on networks of workstations. *IEEE Computer*, 29(2):18–28, Feb. 1996.

[3] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang. Serverless network file systems. In *Proc. of the 15th Symposium on Operating Systems Principles*, pages 109–126, Copper Mountain Resort, Colorado, Dec. 1995. ACM.

[4] A. Calderón, F. García, J. Carretero, J. M. Pérez, and J. Fernández. An Implementation of MPI-IO on Expand: A Parallel File System Based on NFS Servers. In *9th PVM/MPI European User's Group*, September 2002.

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

[6] 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.

[7] P. F. Corbett et al. Parallel file systems for the IBM SP computers. *IBM Systems Journal*, 34(2):222–248, Jan. 1995.

[8] 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.

[9] E. K. Lee and C. A. Thekkath. Petal: Distributed virtual disks. In *Proc. of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 84–92, Cambridge, MA, 1996.

[10] E. K. Lee, C. A. Thekkath, C. Whitaker, and J. Hogg. A comparison of two distributed disk systems. Technical Report 155, Systems Research Center, Digital Equipment Corporation, Apr. 1998.

[11] 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.

[12] D. Muntz. Building a Single Distributed File System from Many NFS Servers. Technical Report HPL-2001-176, 2001.

[13] R. A. Shillner and E. W. Felten. Simplifying distributed file systems using a shared logical disk. Technical Report TR-524-96, Dept. of Computer Science, Princeton University, Princeton, NJ, 1996.

[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.

[15] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck. Scalability in the XFS file system. In *Proceedings of the USENIX 1996 Technical Conference*, pages 1–14, San Diego, CA, USA, Jan. 1996.