On the Design and Performance of Remote Disk Drivers for Clusters of PCs

Vlad Olaru Walter F. Tichy

Computer Science Department
Karlsruhe University, Germany
E-mail: {olaru,tichy}@ipd.uka.de

Abstract

This paper presents the design and performance of remote disk drivers for clusters of Commodity-Off-The-Shelf PCs that fetch disk blocks over System Area Networks. The driver offers a flexible interface, being capable to logically act either as computer- or network-attached storage. It allows for fine-grain remote cache control through exclusive caching. An event-driven asynchronous block delivery mode of operation helps making the most out of the available parallelism by overlapping request processing with block delivery at both involved nodes and thus yielding better performance than local disks. The driver has been implemented as a Linux kernel module.

Keywords: cluster computing, disk drivers, OS, SAN

1 Introduction

Clusters of Commodity-Off-The-Shelf (COTS) PCs have become a practical and cost-effective alternative to hardware supercomputers. However, their tremendous potential in terms of price-competitive and scalable processing power, huge aggregate main memory, price-competitive, highly available and scalable secondary memory, huge I/O bandwidth, cannot be always fully exploited as the PC system software suffers from various well-known problems. One widely recognized problem is the I/O bottleneck [18]. The ever increasing gap between the rates at which processor and disk speeds grow, the additional pressure put on the I/O subsystem by increased data sets, or the lack of cooperation among various optimization algorithms used at various system levels (OS, disk controller) represent some of the most challenging issues related to this problem.

Various solutions tackle the I/O problem at different levels in the system. Low-level solutions striving to improve the I/O performance at disk level either try to bridge the speed gap between the processor and the disk (Active Disks [1]) or, recently, to enhance the cooperation among the various system caches and their inner policies. Exclusive caching [28, 7] avoids double buffering occurring in independently managed caches (e.g., the page/buffer cache in the OS and the on-chip disk cache). Some systems use filesystem knowledge to improve the disk controller operation [23, 7]. At a higher level, distributed filesystems using virtual disks [26, 19] or cooperative caching enabled serverless filesystems [2, 10, 9] attempt to enlarge the storage system cluster-wide and to extend the limit of the local filesystem cache to that of a global, cluster-wide cache. At user-level, the most notable recent trend refers to direct access distributed filesystems [11] in which client machines bypass the local operating system and access the server memory through Remote DMA (memory-to-memory).

In this paper we present a low-level solution for computer-attached remote disks in COTS clusters. They can logically operate as network-attached storage as well due to a single copy protocol that allows implementing exclusive caching and thus isolating the remote system from external influence. Remote disks are mounted locally by means of a remote disk driver as any locally-attached disk would do and the filesystem considers them local storage. Therefore, these drivers must compare favorably to the performance of the local drives. That is accomplished by using a highly asynchronous mode of operation based on an event-driven model of computation that makes the most out of the performance of System Area Networks (SAN), whose latency and bandwidth figures resemble more to those of memory subsystems than to those of networks. Also, the influence of the semantics and the assumptions of the filesystem using the driver should be locally-confined so that they don't have adverse effects on the operation of the remote disk. In this context, we discuss the impact of the filesystem read-ahead policy. A Linux prototype has been implemented and its performance shows improvement over local disks.

2 Background

The interaction between a filesystem and a disk driver is intermediated by a page/buffer cache to speed up the disk access. *Buffer_head* structures [4] are used to describe the in-memory copies of the disk blocks kept in these caches.

The strategy routine [4] intermediating the access relies on buffer heads to pass disk jobs to the driver, and, if possible, to optimize their schedule (in Linux, disk access optimizations coalesce disk requests for consecutive blocks).

When a block is requested, the strategy routine passes on disk requests to the driver. The calling process goes to sleep while the driver serves the requests. When done, a disk interrupt schedules a software handler that dequeues the request, calls a callback routine associated with the buffer head representing the block and wakes up any process that might wait for that block to become available. The callback routine performs buffer head and page management tasks.

3 Motivation

The remote disk driver is a regular block device driver in the kernel. However, fetching disk blocks over a SAN implies meeting certain design decisions. First of all, having to deal with two systems, the remote disk driver design has to decide to which extent local requests will affect the remote page/buffer cache at the physical disk node, as recent research in exclusive caching [28, 7] suggested possible benefits for certain classes of applications. Second, local filesystem level policies like read-ahead may lose their efficiency if not properly exported to the remote system. And last but not least, the driver should make the most out of the available potential for parallelism in order to represent a true alternative to local drives performance-wise.

4 Single copy protocol

The separation of the various buffering systems in the kernel affects the remote disk driver design. The network interface uses specialized socket buffers to send/receive messages while the filesystem uses its own buffering capabilities (the page/buffer cache). That implies additional copying both at the local node and at the remote disk node.

Let's take the example of a file read operation. The remote disk driver prepares locally a page to store a block and sends the block request over the SAN to the remote disk node. There, if already cached, the block is copied to the socket buffer and sent back to the requester. This copy can be avoided only if the SAN card would be capable to DMA directly from the page/buffer cache. Current RDMA implementations [11] for SAN do not allow such things since they require pre-defined pinned-down buffers. Theoretically however, this should be possible, as disks for instance have no problem in sending their data through DMA to randomly chosen pinned-down addresses like those of individual pages/buffers.

For uncached blocks, we instruct the disk at the remote node to DMA directly in the response socket buffer by preparing a socket buffer large enough to hold not only the block data but also the *buffer_head* [4] structure that describes its in-memory copy (see Figure 1). Thus, we bypass both the buffer head slab cache and an additional memory allocation for the data itself. The latter is possible as we use pre-allocated response socket buffers. We fill in the buffer head allocated in the socket buffer with the appropriate values and pass it to the strategy routine [4] of the disk driver. As a result, the destination address for the disk DMA transfer registered in the disk request will be that provided by the buffer head which, in turn, points to an address in the response socket buffer itself. When the disk read operation completes, the disk software interrupt marks the "buffer" uptodate and "releases" the buffer head. In fact, nothing gets done since the buffer head occupies memory in the socket buffer past the useful region and that memory won't be transfered back over the network.

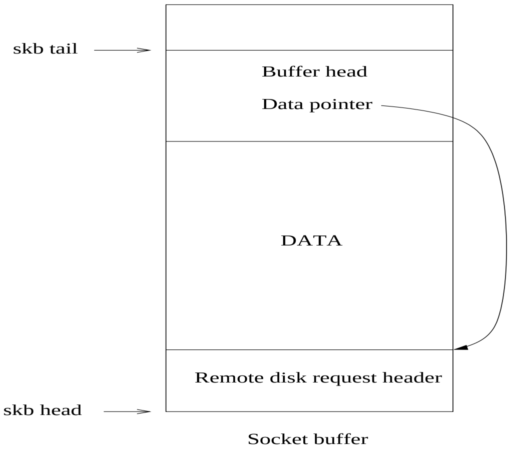

Figure 1.

As soon as the response arrives at the initiator, the remote disk driver that issued the request has to copy the block data to the page cache. Again, this copy could be avoided if the SAN card would have a DMA capability allowing data to be directly sent to the page cache. Or, as a general alternative to any DMA capability, one could use a unified network and cache buffering system such as IO-Lite [21].

The capability to control whether copies of the fetched blocks are left behind in the page/buffer cache at the remote disk node offers extended flexibility. On one hand, fine-grained exclusive caching [28] is possible by simply sending the appropriate remote disk request. On the other hand, the ability to control copies at the remote site enables dual behavior: either computer- or network-attached disk. When the node hosting the physical disk is not logically involved in computation, the single copy protocol makes the remote disk look like network-attached storage.

5 The impact of the filesystem read-ahead policy and disk fragmentation

Traditional kernels use a filesystem read-ahead policy to improve the disk usage and throughput. The default policy is to sequentially prefetch a given number of blocks. The policy adapts its behavior to the file access pattern of an application by using a read-ahead window that may shrink to zero when accesses become random. This policy is complemented by the optimizations performed by the strategy routine of the disk driver and the disk controller itself. Independent management of all these caches and their optimization algorithms may lead to performance degradation, as pointed out by recent research [23, 7].

When exporting an entire disk through a remote disk driver, the natural question is how to translate the local filesystem read-ahead policy at the remote disk node since that node is not aware of the assumptions made by the local system issuing the disk requests. A simple answer is to use a synchronous model employed by local disk drivers and to rely on the read-ahead facilities of the remote disk controller. But this may not work properly for certain classes of applications (Web servers that serve mostly small files). Moreover, disk fragmentation worsens the performance due to unnecessary disk accesses that pollute the caches and reduce meaningful disk throughput.

Fortunately, it is fairly easy to translate the read-ahead access pattern to the remote site by asynchronously sending disk requests. The low-level driver routine concluding the decisions of the strategy procedure removes the currently issued disk request from the driver queue allowing thus the next requests to be processed. The processed requests are gathered in a separate queue that will be walked through when the block replies arrive. Thus, if we consider the low latency of passing a small message over a SAN, we can affirm that the read-ahead block requests arrive and are queued at the remote disk node with minimal delay.

Reading ahead raises another question: how much to read ahead ? The filesystem makes his assumptions about the underlying storage system and sets an upper bound to the read-ahead window. With respect to the remote disk driver design, one has to answer the following question: are these assumptions still correct when the disk is not local anymore? Section 7 will show the sensitivity of our asynchronous mode of sending disk requests to this parameter.

6 Remote disk drivers and the network

The remote disk driver design has to cope also with the consequences of using a SAN to move blocks back and forth. Requests arriving at the remote disk site behave as regular messages delivered by the SAN network card at interrupt time to the remote host. In our previous work [20], we identified some design decisions to be met regarding how to handle incoming block requests. In short, we argued in favor of a mixed event-driven/blocking model in which cached disk blocks are delivered at interrupt time to enhance responsiveness (akin to systems like Active Messages [27]) while deferring the disk service to a kernel thread. This thread delivers the blocks from within a well-defined protection domain that allows avoiding unfairness caused by interrupt-driven computation as pointed out by the research experience with network subsystems [5, 6].

6.1 Asynchronous block delivery

The operation of the kernel thread delivering the disk blocks may be synchronous or asynchronous. For reasons discussed in Section 5, a synchronous model is not acceptable. Moreover, a synchronous block delivery mechanism loses the opportunity to profit from disk strategy routine and disk controller optimizations.

In asynchronous block delivery mode, the thread launches a number of disk requests in execution and blocks awaiting for the first request to complete. The benefits come here from amortizing the cost of a context-switch over several disk requests, from the optimizations performed by the strategy routine of the disk driver and from the disk access optimization algorithms implemented by the disk controller. However, there is still one problem left. Once the blocks have been loaded in memory (either in the local page/buffer cache or, as we presented in Section 4, directly in the response socket buffer), they have to be sent back to the requester as well. That may entail additional processing that cannot be carried out in parallel with the request handling.

A better solution uses an event-driven model. Disk drivers signal the completion of block reads by calling a disk software interrupt that removes the request from the driver queue, marks it free, calls potential callbacks associated with the buffers heads used for the blocks and launches new disk jobs, if available. Having the possibility of running a callback at interrupt time is the key to better performance because these callbacks can be used to sent right away the freshly loaded block back to the requester. Thus, by simply registering the appropriate callback in the buffer head passed to the strategy routine, one gets maximum amount of parallelism (actually pseudo-parallelism as usual on uniprocessor machines) between request processing and data delivery. The major disadvantage of this solution remains unfairness, as already pointed out in our previous work. The costs of sending the blocks are charged to the currently running process that happened to be interrupted by the disk completion event.

7 Performance evaluation

In order to evaluate the performance of our remote disk driver we used WebStone [24], a well known commercial benchmark for Web servers respecting a Zipf-like [29] document retrieval distribution. The WebStone software has been configured to retrieve static documents only. Therefore, throughout the rest of this section, by WebStone operations we refer to HTTP GET commands. The benchmark used HTTP 1.0 and a file set around 1 GB of data. The server(s) used local disks, NFS (over Ethernet) and Linux *ext2* on top of our remote disk driver. The driver used the single copy protocol (that is, it behaved more like a network-attached disk than a computer-attached one) and we tested with both asynchronous delivery mechanisms: the thread-based one, designated as the *asynchronous* case, and the interrupt-based one, called the *event driven* case.

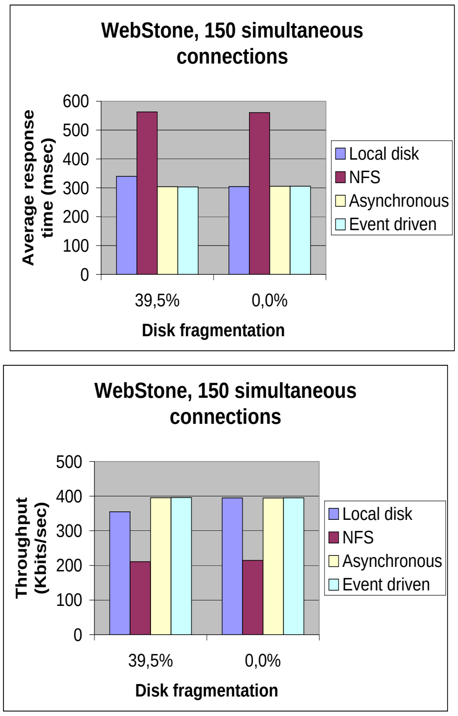

Figure 2.

We drove two kinds of experiments. The first type attempted to assess the performance of a single server using remote disk drivers. We evaluated the impact of the load, disk fragmentation and read-ahead policy of the filesystem on the server performance. The results are presented in subsections 7.2 and 7.3. The second type of experiments considered the operation of remote disk drivers in distributed servers. The aim was to get an idea about the behavior of our remote disk acting either as a computer-attached or as a network-attached disk. In the first case, the node hosting the physical disk was running a Web server instance as well. For the other case, two server machines mounted the remote disk locally. Subsection 7.4 presents and discussed the corresponding results.

7.1 Experimental Setup

We ran our experiments on a 3-node Linux cluster interconnected through a Myrinet switch and LANai 7 cards (133 MHz processor on board, 2 Gb/sec in each direction). The host interface is a 64 bit/66 MHz PCI that can sustain a throughput of 500 MB/sec. The Myrinet cards are controlled by the GM 1.6.4 driver of Myricom [25]. The PCs are 350 MHz Pentium II machines with 256 MB of RAM. All the systems run Linux 2.2.14.

The test disks are IBM DCAS-34330W Fast/Ultra-SE SCSI. Only disk partitions were remotely mounted for the experiments that we further describe. Both disks are formatted with a native Linux filesystem format (*ext2*). One of the disks has an aged filesystem on it (39.5% non-contiguous, as reported by the *fsck* command) while the other one is newly formatted (0% non-contiguous). The server machines mounted the remote disk partitions read-only.

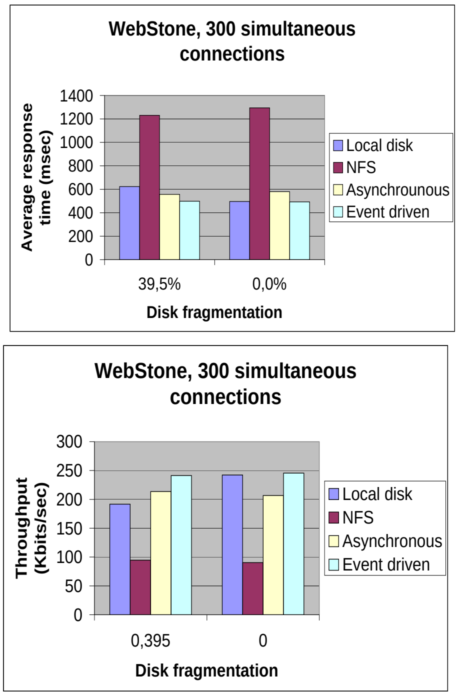

Figure 3.

As Web server we used Apache 1.3.20 [3]. A Linux router stays between the client and the server machines. Both the client and the router are Athlon AMD XP 1.5 GHz PCs with 512 MB of RAM and run Linux 2.4.18. The client, the router and the server(s) are all interconnected through regular 100Mb/s Ethernet.

7.2 The impact of the WebStone load on the remote disk driver

We varied the load WebStone puts on the server by instructing the benchmark to use a number of simultaneous connections of 150 and 300, respectively. In terms of requested data, that corresponds to roughly 900 MB and 1.6 GB, respectively. The results are presented in Figures 2 and 3. The average response time refers to the average time taken by an operation (GET command, actually).

Notice that for the light load there are no significant differences between the two remote disk methods. Naturally, the local disk performance of the aged disk is somewhat worse than that of the non-aged one. Surprisingly, NFS copes better with disk fragmentation than local disks. Its insensitivity to disk fragmentation for this load is similar to that exhibited by the remote disk driver cases. Overall, remote disk drivers outperform local disks and NFS. The difference is unnoticeable for the non-aged disk but visible for the aged one. This may be a bit surprising when it comes to local disks, but it must be recalled that the remote disk drivers use a highly asynchronous mode of operation both at the local and remote sites. That allows some degree of processing overlapping that makes up for the lack of contiguity of the aged disk whose greater mechanical latencies can be thus better hidden.

Under heavy load (Figure 3), both remote disk driver methods yield some sensitivity to the disk fragmentation. Overall, the event driven method clearly outperforms the asynchronous one. Disk fragmentation affects the comparison to local disks. For the non-aged disk, the event driven method outperforms local disks, but the gain is minimal. For the aged disk, as the load increases, the lack of contiguity entails higher performance degradation for the local disk. Both remote disk driver methods clearly yield better figures as they manage to hide more of the increased mechanical latencies in the parallel processing of the two nodes. Moreover, the higher the degree of asynchrony (and therefore pseudo-parallelism), the better the performance.

7.3 The impact of the read-ahead policy of the filesystem

As mentioned in Section 5, one interesting question is whether the remote disk driver properly exports the effects of the read-ahead policy of the local *ext2* filesystem. To answer the question, we varied the size of the maximum number of the read-ahead pages. We used the event-driven method and a load of 300 simultaneous connections. The results are shown in Figure 4. The kernel default value specifies to read ahead at most 31 pages. Notice that for both disk types, the remote disk driver yields best performance for the same value. More interesting however, for the other read-ahead values, the performance of the aged disk is better than that of the non-aged one. That clearly points out that exporting a wrong maximum value for the read-ahead window is not only suboptimal but ceases to serve the purposes of reading ahead (since the contiguous disk performs worse than the non-contiguous one, which is totally counterintuitive). Such decisions can thus affect the performance of the remote system and therefore undermine our design goal to minimize the remote impact of locally run policies.

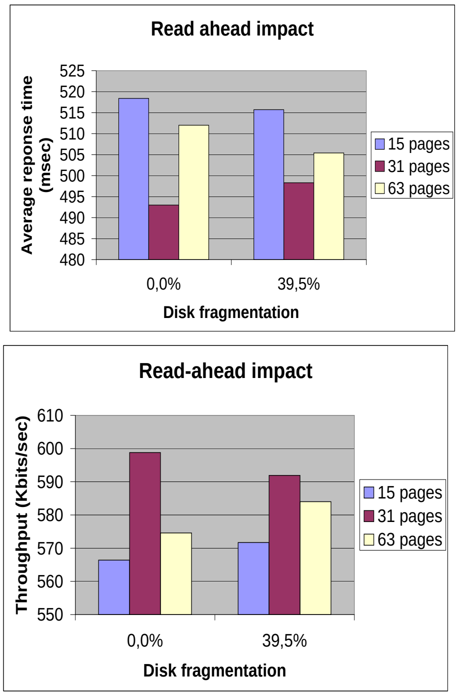

Figure 4.

7.4 Distributed server performance

The distributed server performance evaluation uses two server machines in three setups. The router acts as a front-end dispatcher using a round robin distribution policy. In the first setup, the two machines serve the requests from the local disks (same file set, replicated on both disks). The disks are mostly contiguous (0% and 1.7% non-contiguity, respectively). In a second scenario, one of the two servers uses a remote disk driver to mount the 0% non-contiguous disk locally using the event driven method. This case corresponds to the computer-attached operation of the disk. In the third setup, two machines mount locally the remote disk as a network-attached disk using the event driven method. The WebStone load used was 300 simultaneous connections. The results are reported in Figure 5.

The two servers equipped with local disks perform best. The load is almost equally split between the two disk drives and that maximizes the amount of disk parallelism. The other two cases show that the task is disk and not computational dominated. Indeed, notice that their performance doesn't differ much from that of the corresponding single server in Figure 3 (slightly worse for case two and even better for case three). For the second scenario, we assume that the additional load placed by the server software on the node hosting the disk is responsible for the light performance degradation. To conclude, disk utilization is affected by additional remote operation, but not by an increased number of clients mounting the disk remotely.

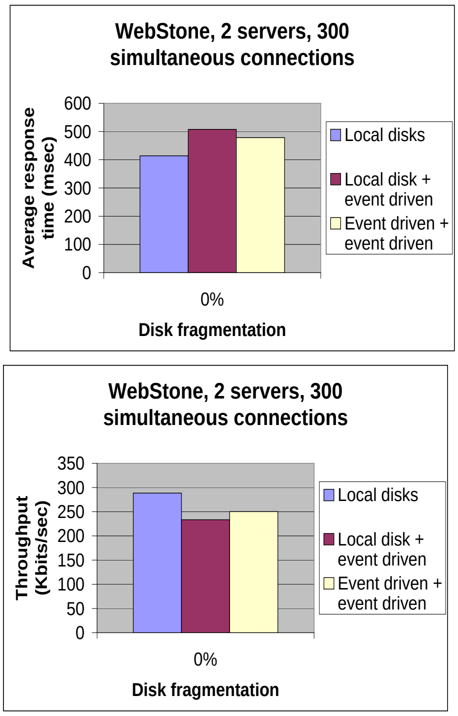

Figure 5.

8 Related work

The I/O bottleneck problem has been tackled at various levels of the system and among systems (remote I/O for parallel/distributed computing). Remote I/O systems can be broadly classified in: user-level-, filesystem- or low-level-oriented solutions.

Low-level standalone I/O systems attempt to improve performance at disk level either by bridging the speed gap between the processor and the disk (Active Disks [1]) or by integrating the various system caches into a cooperative I/O infrastructure. Exclusive caching systems [28, 7] avoid double buffering occurring in independently managed caches (e.g., filesystem page/buffer cache and disk controller cache). The mismatch between the read-ahead policies of the filesystem and the disk controller is compensated by filesystem-aware disk controller prefetching [23, 7].

Distributed low-level I/O systems include *virtual disks* [19] providing their clients a block-oriented, globally accessible and consistent view of a physically distributed disk pool, *single-image I/O systems* [15] amassing the entire (local and remote) disk capacity of a node into a RAID-like structure, *striped log-based storage* [14] or object-oriented based *network-attached secure disks* [13].

At higher levels, distributed filesystems like Frangipani [26] use Petal [19] virtual disks and supply them with meta-data consistency support. Server-less filesystems [2] distribute the meta-data management and use cooperative caching [10, 22] to scale their working set beyond the limit of the locally available memory. The usual distributed filesystem memory hierarchy (local cache, server cache, server disk) extends by adding the client caches, provided that remote client cache reads take less time than accessing the disk. The PACA [9] parallel filesystem mixes cooperative caching with global memory and RDMA.

Direct access filesystems (DAFS [11]) modify the distributed I/O memory hierarchy as well, not by adding but by removing a level, namely the local kernel cache. Addressed to a class of applications that have seldom sharing patterns, DAFS runs in user space and uses RDMA to communicate directly with the file server. Thus, the local operating system is bypassed. Unlike PACA, no global memory support is provided. Other user-level systems are mostly a work-around: block devices [17] or filesystems (PVFS [16]) in user space, remote I/O libraries [12] over MPI-IO [8]. Since traditional kernels are unaware of the distributed nature of these systems and their inner mechanisms and policies fail to match the expectations of the user space driven computation, the end result is performance penalty. Also, moving typical kernel code in user space incurs increased application software complexity and more difficult development.

9 Conclusions

In this paper we presented the design of a remote disk driver for COTS clusters. A single copy protocol allows implementing exclusive caching. As needed, the driver exhibits either computer- or network-attached storage behavior. The driver properly exports local filesystem read-ahead decisions to the remote system and uses a highly asynchronous operation mode to outperform local disks.

References

[1] A. Acharya, M. Uysal, and J. Saltz. Active Disks: Programming Model, Algorithms and Evaluation. In *Proceedings of the 8th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)* , October 1998.

[2] T. Anderson, M. Dahlin, J. M. Neefe, D. Patterson, D. Rosseli, and R. Y. Wang. Serverless Network File Systems. In *The 15th Symposium on Operating System Principles*, December 1995.

[3] Apache. *http:/www.apache.org/*.

[4] Maurice J. Bach. *The Design of the Unix Operating System*, 1986. Prentice Hall Inc.

[5] G. Banga, P. Druschel, and J. Mogul. Lazy Receiver Processing (LRP): A Network Subsystem Architecture for Server Systems . In *Proceedings of the Second Symposium on Operating Systems Design and Implementation (OSDI'96)*, October 1996.

[6] G. Banga, P. Druschel, and J. Mogul. Resource containers: A new facility for resource management in server systems . In *Proceedings of the Third Symposium on Operating System Design and Implementation* , February 1999.

[7] Enrique V. Carera and Ricardo Bianchini. Improving Disk Throughput in Data-Intensive Servers. In *Proc. of the 10th IEEE International Symposium on High Performance Computer Architecture*, February 2004.

[8] P. Corbett, D. Feitelson, Y. Hsu, J.-P. Prost, M. Snir, S. Fineberg, B. Nitzberg, B. Traversat, and P. Wong. MPI-IO: A parallel file I/O interface for MPI. Technical Report NAS-95-002, NASA Ames Research Center, Moffet Field, CA, January 1995.

[9] T. Cortes, S. Girona, and L. Labarta. PACA: A Distributed File System Cache for Parallel Machines. Performance under Unix-like workload. Technical Report UPC-DAC-RR-95/20 or UPC-CEPBA-RR-95/13, Departament d'Arquitectura de Computadors, Universitat Politecnica de Catalunya, 1995.

[10] M. Dahlin, R. Yang, T. Anderson, and D. Patterson. Cooperative Caching: Using Remote Client Memory to Improve File System Performance. In *The First Symp. on Operating Systems Design and Implementation*, November 1994.

[11] Matt DeBergalis, Peter Corbett, Steve Kleiman, Arthut Lent, Dave Noveck, Tom Talpey, and Mark Wittle. The Direct Access File System. In *Proceedings of the 2nd Usenix Conference on File and Storage Technologies(FAST '03)*, pages 175–188, San Francisco, USA, March–April 2003.

[12] I. Foster, Jr. D. Kohr, R. Krishnaiyer, and J. Mogill. Remote I/O: Fast Access to Distant Storage. In *Proceedings of the Fifth Annual Workshop on I/O in Parallel and Distributed Systems*, November 1997.

[13] G. A. Gibson, D. F. Nagle, K. Amiri, F. W. Chang, H Gobioff, E. Riedel, D. Rochberg, and J. Zelenka. Filesystems for Network-Attached Secure Disks. Technical Report CMU-CS-97-118, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3890, July 1997.

[14] J. H. Hartman, I. Murdock, and T. Spalink. The Swarm Scalable Storage System. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS '99)*, June 1999.

[15] R. S. C. Ho, K. Hwang, and H. Jin. Single I/O space for Scalable Cluster Computing. In *Proceedings of the 1st IEEE International Workshop on Cluster Computing*, December 1999.

[16] W. B. Ligon III and R. B. Ross. An Overview of the Parallel Virtual File System. In *Proceedings of the Extreme Linux Workshop*, June 1999.

[17] Kangho Kim, Jin-Soo Kim, and Sungin Jung. A Network Block Device Over Virtual Interface Architecture on LINUX. In *Proceedings of the IEEE International Parallel and Distributed Symposium*, April 2002.

[18] David Kotz and Ravi Jain. I/O in parallel and distributed systems. In Allen Kent and James G. Williams, editors, *Encyclopedia of Computer Science and Technology*, volume 40, pages 141–154. Marcel Dekker, Inc., 1999. Supplement 25.

[19] E. K. Lee and C. A. Thekkath. Petal: Distributed virtual disks. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1996.

[20] V. Olaru and W. F. Tichy. CARDs: Cluster Aware Remote Disks. In *Proceedings of the Third IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, May 2003.

[21] V. S. Pai, P. Druschel, and W. Zwaenepoel. IO-Lite: An Unified I/O Buffering and Caching System . In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, February 1999.

[22] Prasenjit Sarkar and John H. Hartman. Efficient Cooperative Caching using Hints. In *Proceedings of the Second Symposium on Operating Systems Design and Implementation*, October 1996.

[23] M. Sivathanu, V. Prabhakaran, F. Popovici, T. Denehy, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Semantically-Smart Disk Systems. In *Proc. of the 2nd Usenix Conference on File and Storage Technologies (FAST '03)*, March 2003.

[24] The Standard Performance Evaluation Corporation (SPEC). *http://www.spec.org/web99/*.

[25] Myricom Inc. GM: the low-level message-passing system for Myrinet networks. *http://www.myri.com/scs/index.html*.

[26] C. A. Thekkath, T. Mann, and E. K. Lee. Frangipani: A scalable distributed file system. In *Proceedings of the 16th ACM Symposium on Operating System Principles*, 1997.

[27] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active Messages: a Mechanism for Integrated Communication and Computation. In *Proceedings of the 19th International Symposium on Computer Architecture*, May 1992.

[28] Theodore M. Wong and John Wilkes. My Cache or Yours? Making Storage More Exclusive. In *Proc. of the USENIX Annual Technical Conference*, pages 161–175, 2002.

[29] G. Zipf. *Human Behavior and the Principle of Least Effort*, 1949. Addison Wesley.