Autonomous Disks: The Building Block for a Scalable Distributed File System

Cuneyt Akinlar[1] Walid G. Aref[2] Ibrahim Kamel[1] Sarit Mukherjee[1]

[1] Panasonic Information and Networking Technologies Laboratories
Panasonic Technologies, Inc.
2 Research Way
Princeton, NJ 08540
E.mail: {akinlar, ibrahim, sarit}@research.panasonic.com

[2] Department of Computer Sciences
Purdue University
1398 Computer Science Building
West Lafayette, IN 47907-1398
E:mail: aref@cs.Purdue.edu

Abstract

Repository for multimedia data differs from that of the traditional text-based data both in storage space and streaming bandwidth requirements. Thus, the file systems used in multimedia environment need to support large volumes and high bandwidth. In this paper, we propose a novel concept called autonomous disks to build scalable distributed file systems. Autonomous disks are attached to the network and are able to do lightweight processing. We propose a file system architecture based on the autonomous disks. We discuss different ways to realize autonomous disk, and describe a prototype implementation on Linux platform using PC-based hardware. We present experimental results on the performance of the proposed file system implemented using the autonomous disks. We show that it scales linearly with number of disks and number of clients, and performs much superior to NFS running on the same hardware platform.

1 Introduction

Advances in networking and storage technology, along with the digitization of multimedia streams have created the need for large and fast servers. The servers are typically used as repositories connected to the network. Multiple client hosts are able to use them online over the network. Usually a file system runs on the server. The clients mount the file system on their hosts, and use the server functionalities seamlessly.

Typically, multimedia servers can be of three types: centralized, distributed, and serverless. In a system with a centralized server (e.g., Network File System (NFS) [13], Symphony [15], Continuous Media File System (CMFS) [3]) a single dedicated node controls the admission process as well as all other file operations and security issues. All the read/write operations are approved and handled by the centralized server. Thus the implementation is simple and the data integrity maintenance overhead is low. Because of the centralized control, the security can be enforced easily. The main disadvantage of the centralized approach is the scalability problem. As the number of clients grow, the server becomes a performance bottleneck. The server is also a threat as a single point of failure. Although a mirror of the server can be maintained but the failure of the server will affect all the clients at the same time without any graceful degradation.

In a distributed server environment (e.g., Global File System (GFS) [16], Frangipani [18], CentraVision File System (CVFS) [2], File systems for network-attached secure disks (NASD) [8,9]) a set of designated nodes share the load and functions of the server. Thus the load is distributed over the servers enabling scalability of the system. More servers can be added as the number of clients increases. There is no single point of failure and in case of failure of a server the system gracefully degrades as the load of the faulty server can be handled by the other servers. In any case, the failure of a server affects only a subset of the clients. In distributed servers the level of security is lower compared to that of the centralized server as more nodes (all the servers) are being trusted. In addition, we need to implement algorithms for resource sharing as well as for maintenance of consistent information across all the servers.

A serverless system (e.g., Xfs [4]) is employed in a trusted environment where security is not a major concern. All the clients as well as the storage devices are connected directly to the network. Any device can be accessed by any of the clients directly. In this environment there are no servers. Based on the system status, the client host performs its own admission control and resource allocation. Thus the functionality of the server is distributed in a self-service fashion. These systems have no performance bottlenecks and are highly scalable. Server fault-tolerance issues are non-existent in serverless systems.

The failure of a client does not affect any other clients. The clients share the load involved in maintaining data consistency. With the advances in the concept of active disks [1,14], in future, the serverless file system approach may become very efficient and can ensure security.

In this paper we propose a distributed file system based on a novel architecture, that we call Autonomous Disk (AD). An AD is a disk that can be attached to the network directly. It has a small processing engine that allows it to perform some basic operations. AD provides flexibity in designing a file server. It can be used to build a distributed file system by delegating tasks among multiple ADs. A serverless file system can be implemented by performing file system operations in the AD. It is also possible to build security module into AD to prevent unauthorized usage of the system. We would like to stress that an AD is a concept that can be realized by different hardware/software means.

We propose a distributed file system (DFS) architecture that uses AD as a building block. We show that the concept of an AD makes the design of the file system very simple, and expandable. We describe a prototype implementation of AD on the Linux platform using PC-based hardware. We build the DFS using the prototype ADs. Our design allows the DFS to have the same “look and feel” of the native Linux second extended file system (ext2fs). Using experimental results we show that DFS performs significantly better than NFS on the same hardware configuration. It also demonstrates that on the same hardware platform, DFS built around AD scales up linearly while NFS does not.

This remainder of the paper is organized as follows. In the next section, we elaborate on the concept of autonomous disks and describe the architecture of the file system. Section 3 describes the emulation platform for the implementation of an AD. It also describes the results obtained from the DFS experiments and a comparison with NFS. The paper is concluded in Section 4.

2 The Architecture of DFS

The schematic architecture of the Distributed File System (DFS) is shown in Figure **1**. As shown in Figure **1**(a), the file system has a distributed architecture with a number of storage devices, called autonomous disks, connected to a high-speed network. The following section elaborates more on the autonomous disks. The network typically is Ethernet based (i.e., Fast Ethernet or Gigabit Ethernet), or Fibre Channel for high network bandwidth. The user hosts (i.e., DFS clients) are connected to the same high-speed network. One of the DFS clients is equipped with capabilities to maintain and distribute DFS-specific data structures (i.e., meta-data), system configurations and access control. We call it a configuration manager. Figure **1**(b) shows the kernel of the DFS. It is distributed across the autonomous disks, the user hosts and the configuration manager. They work together in a coordinated fashion to make the operations seamless to the users.

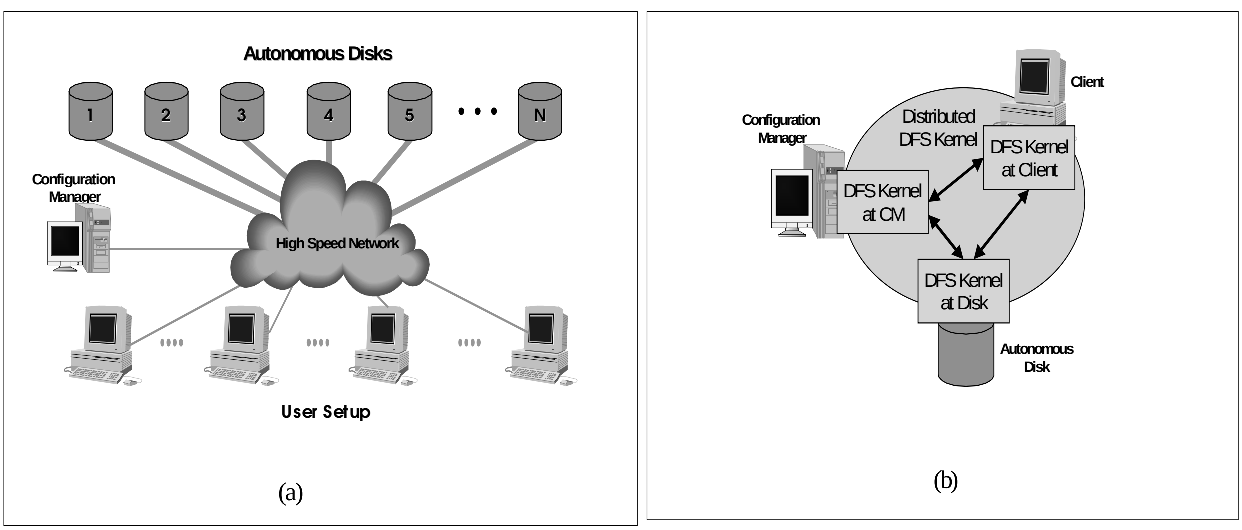


Figure 1: The architecture of DFS, (a) autonomous disks attached directly to the network, (b) the kernel of DFS.

2.1 Architecture of an Autonomous Disk

The AD (refer to Figure 2) is a disk with a small programmable processor and memory. It has a network interface to connect it directly to a high-speed network. The processor on the AD is used to perform some lightweight file system related functions. These functions are performed as a part of the DFS kernel running at the disk. The functions include:

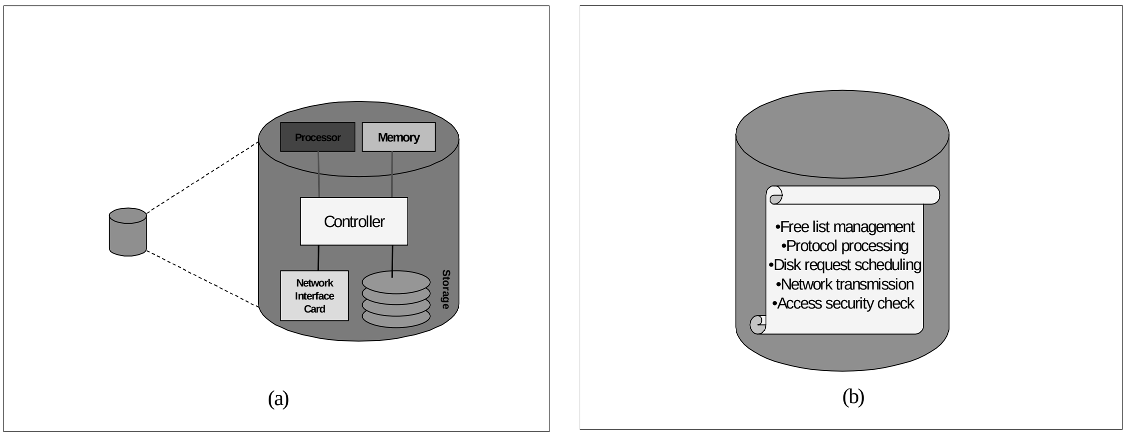


Figure 2: The autonomous disk, (a) its structure, (b) its functions.

- Free list management: Unlike legacy file systems where the host maintains the free list of the disks, and on-demand allocate and de-allocate blocks from the list, the AD's processor performs the same set of functions.
- Network protocol processing and packet transmission: The AD is capable of performing a subset of the protocol processing, ideally the Internet Protocol. The data-link layer for the AD could be Fibre Channel or other high speed MAC (e.g. Fast or Gigabit Ethernet) protocol.
- Disk request scheduling: The disk resident processor is able to conduct intelligent request scheduling. It is desirable that the scheduling algorithm be programmable.
- Access security control: Since the AD is "directly" connected to the network (instead of the traditional "behind-the-server" approach), it is now prone to security attacks. Thus, it has to perform its own access security checks.

Note that AD is a concept that can be realized through different hardware platforms, e.g., a regular workstation with disks or active network-attached disks.

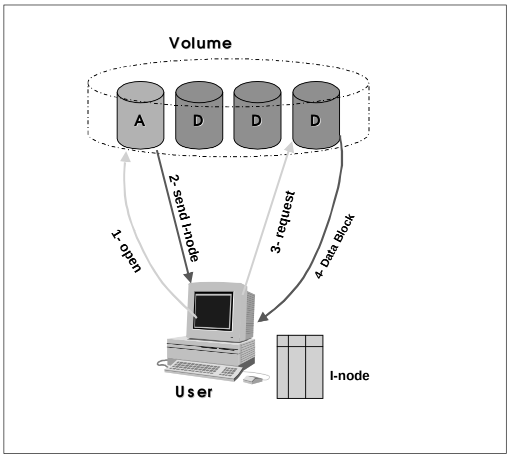
2.2 Volumes in DFS

DFS is organized in data volumes. Each volume consists of one or more ADs. A file is striped on the data disks (denoted by **D** in Figure 3) of the volume. The file system related meta-data for the volume is stored in another disk called the attribute disk (denoted by **A** in Figure 3). This allows DFS to treat the control and data separately and also makes upgrade of a volume (for bigger storage and higher bandwidth) possible without changing the attributes.

2.3 Read and Write Operations

A user is allowed to perform read/write operations into the DFS only after it has logged in and has been authorized by the configuration manager to use the volume. A high-level view of the read procedure in DFS is shown in Figure 3(a). In response to the application's read request, the DFS kernel running at the user site first contacts the attribute disk of the volume for the meta-data. Among other information, the attribute disk sends the file index table to the user. With the help of the file index table the DFS kernel at the user site can translate any read request into the address of the appropriate data disk and the block to read from that disk. This request is then sent to that disk, and the disk returns the requested data block.

The write operation is somewhat different from the read operation in the sense that the file index table does not exist in the attribute disk of the volume. The user creates the file index table as the file is being written and then flushes the table into the attribute disk. For each write operation the DFS kernel at the user sends a block of data to the appropriate data disk. The data disk selects a free block from its own free list, writes the data into it, and sends the block address to the user. The user builds the file index table from this information. Figure 3(b) shows the messages that go back and forth between the user and the disks during a write operation.



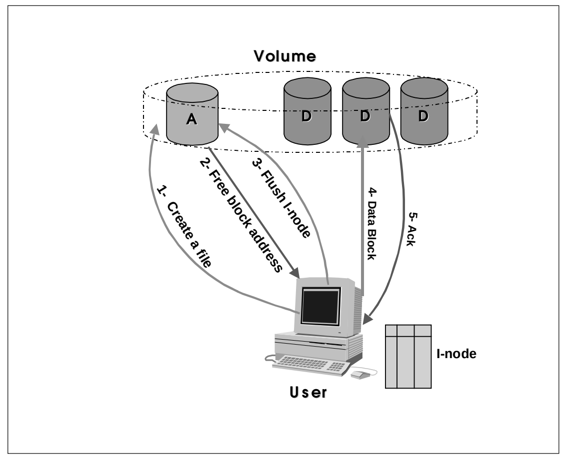


Figure 3: The steps needed to perform (a) read operation, (b) write operation.

3 Numerical Results and Performance Comparison

We implement an AD using a PC-based workstation running Linux operating system. The processing for the ADs is performed by the host CPU, and the workstation's NIC works as the network attachment. The functionalities of an AD are built into the Linux kernel with all the functions described in Section 2.1, except the security. Note that a single workstation can implement a number of ADs as long as the aggregate disk bandwidth matches that of the NIC.

Figure 4 shows our experimental setup. One of the hosts is a workstation with a Pentium-II 400 processor and a Packet Engine GNC-II Gigabit Ethernet NIC. We attach four disks to it since it has enough network bandwidth to support the aggregate bandwidth of four disks. This host implements four ADs. The other two hosts have Fast Ethernet NICs and implement one AD each in order to match the disk bandwidth. This demonstrates the flexibility of our implementation. All the client hosts are Fast Ethernet based. A switch interconnects all the components of the system. Each client runs the following simple code for bandwidth measurement:

```
function MeasureBW (int no_of_blocks)
        start_timer();
        for (i = 0; i < no_of_blocks; i++)
                read (DFS_BLOCK_SIZE);
        stop_timer();
        BW = (no_of_blocks*DFS_BLOCK_SIZE)/ElapsedTime;
end;
```

Notice that the experimental setup allows creation of DFS using n ADs, $n = 1, \ldots, 6$. Figure 5 shows the results obtained from the experimentation of DFS varying the number of ADs and the number of clients. Each line in the figure shows the aggregate bandwidth obtained from DFS by all the clients using a specific number of ADs. The aggregate bandwidth increases with number of clients and line becomes horizontal when the aggregate disk bandwidth from DFS saturates to its maximum. As the number of disks is increased, the bandwidth scales up and saturates at a higher level. From the figure we observe that DFS scales almost linearly with the number of ADs and the number of users present in the system. Also notice that DFS can seamlessly integrate multiple ADs implemented on different workstations.

For the sake of comparison, we use the same hardware to run NFS server that comes with Linux distribution. Since NFS runs on a single host, we use the workstation with Gigabit Ethernet NIC and multiple disks as the NFS server. The disk bandwidth is scaled up by striping a number of disks with Linux's second extended file system (ext2fs) disk striping. The striped disk set is formatted with ext2fs. The clients run the same bandwidth measurement programs, but the requests now go to the NFS server. Figure 6 shows the results obtained from the experimentation. We observe that the processing at the server becomes the bottleneck as the number of clients increases, and the aggregate bandwidth from the server goes down. Aggregate disk bandwidth increases with the size of the stripe set, but it does not scale.

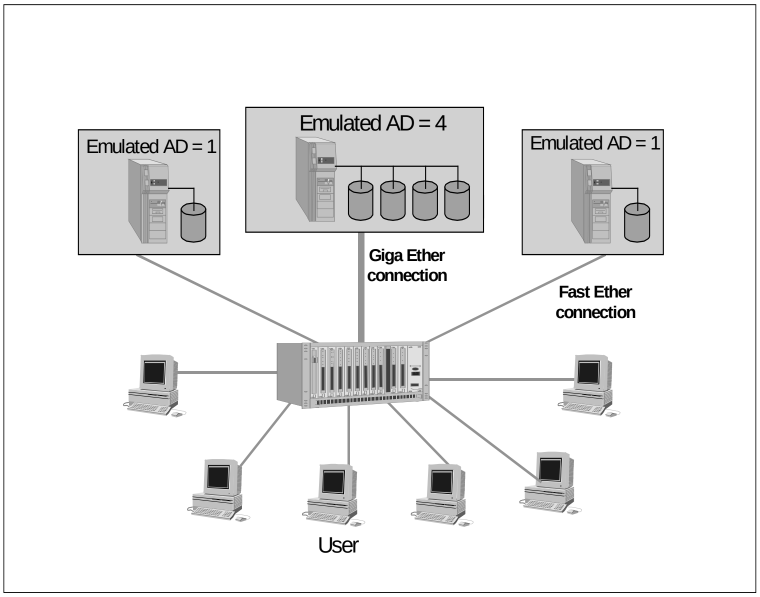


Figure 4: The experimental setup.

4 Conclusions

In this paper we proposed a novel concept, called autonomous disk, that can perform some computations and is network attached. We described the implementation of such a disk using regular PC-based hardware and software. The autonomous disks bring flexibility in the design of a file system. It can provide higher bandwidth and better scalability.

5 References

1. A. Acharya, M. Uysal, and J. Saltz, "Active Disks," Int. Conf. on the Architectural Support for Programming Languages and Operating Systems, 1998.
2. H. Aine, "The CentraVision File System," MountainGate White Paper, Aug. 1997 Available at http://www.mountaingate.com.
3. D. P. Anderson, Y. Osawa, and R. Govindan, "A File System for Continuous Media," *ACM Trans. on Computer Systems*, pp. 311-337, Nov. 1992.
4. T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang, "Serverless Network File Systems," *ACM Trans. on Computer Systems*, Feb. 1996.
5. W. J. Bolosky, R. P. Fitzgerald, J. R. Douceur, "Distributed Schedule Management in the Tiger Video Server," *ACM Symposium on Operating System Principles*, 1997.
6. P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID: High-Performance, Reliable Secondary Storage," *ACM Computing Surveys*, pp. 145-185, June 1994.

7. DataDirect Networks, The Next Generation of Network Computing, 1998. Available at http://www.datadirectnet.com.
8. G. A. Gibson, et al., "Filesystems for Network-Attached Secure Disks," Tech. Report CMU-CS-97-118, Carnegie Mellon University, July 1997. Available at http://www.cs.cmu.edu/Web/Groups/NASD.
9. G. A. Gibson, et al., "A Case for Network Attached Secure Disks," Tech. Report CMU-CS-96-142, Carnegie Mellon University, June 1996. Available at http://www.cs.cmu.edu/Web/Groups/NASD.
10. H. Kaneko and J. A. Stankovic, "A Multimedia Server on the Spring Real-Time System," Tech. Report 96-11, UMass Computer Science Department, 1996.
11. A. Molano, K. Juvva, and R. Rajkumar, "Real-Time File Systems: Guaranteeing Timing Constraints for Disk Accesses in RT-Mach," *Real-Time System Symposium*, 1997.
12. A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case," *IEEE Trans. on Networking*, pp. 137-150, April 1994.
13. B. Pawlowski,, C. Juszczak, P. Staubach, C. Smith, D. Lebel, D. Hitz, "NFS Version 3 Design and Implementation", Winter USENIX Conf. Proc., USENIX Association, Berkeley, CA, January 1994.
14. E. Rieldel and G. A. Gibson, "Active Disks – Remote Execution for Network-Attached Storage," Tech. Report CMU-CS-97-198, Carnegie Mellon University, 1997.
15. P. J. Shenoy, P. Goyal, S. S. Rao, and H. M. Vin, "Symphony: An Integrated Multimedia File System," *ACM SIGMETRICS Conference on Modeling and Evaluation of Computer Systems*, 1998.
16. S. R. Soltis, G. M. Erickson, K. W. Preslan, M. T. O'Keefe, and T. M. Ruwart, "The Global File System: A File System for Shared Disk Storage," Submitted to the *IEEE Trans. on Parallel and Distributed Systems*.
17. J. A. Stankovic and K. Ramamritham, "The Spring Kernel: A New Paradigm for Real-Time Systems," *IEEE Software*, pp. 62-72, May 1991.
18. C. A. Thekkath, T. Mann, and E. K. Lee, "Frangipani: A Scalable Distributed File System," *ACM Symposium on Operating System Principles*, 1997.
19. F. A. Tobagi, J. Pang, R. Baird, and M. Gang, "Streaming RAID: A Disk Storage System for Video and Audio Files," *Proc. of ACM Multimedia*, pp. 393-400, 1993.
20. M. Vernick, C. Venkatramani, and T. Chiueh, "Adventures in Building the Stony Brook Video Server," *Proc. of ACM Multimedia*, 1996.

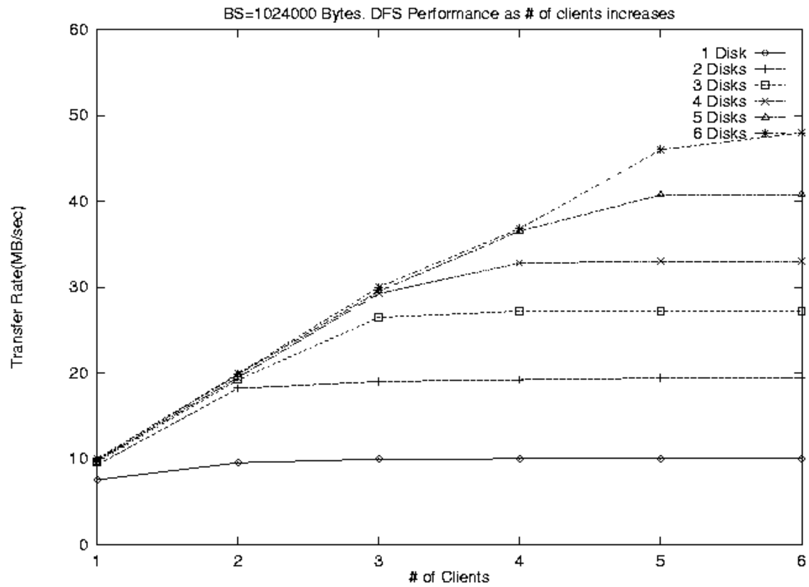


Figure 5: The effect of increasing the number of clients and the number of autonomous disks on the overall transfer rate of DFS.

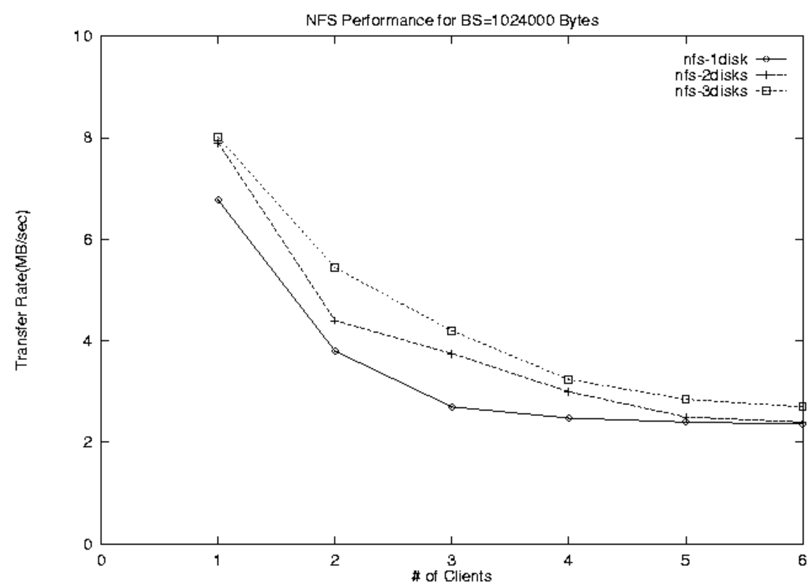


Figure 6: The effect of increasing the number of clients and the number of disks on the overall transfer rate of NFS.