April 30, 1998

SRC Research Report 155

A Comparison of Two Distributed Disk Systems

Edward K. Lee
Chandramohan A. Thekkath
Chris Whitaker
Jim Hogg

digital

Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301

http://www.research.digital.com/SRC/

Systems Research Center

The charter of SRC is to advance both the state of knowledge and the state of the art in computer systems. From our establishment in 1984, we have performed basic and applied research to support Digital's business objectives. Our current work includes exploring distributed personal computing on multiple platforms, networking, programming technology, system modelling and management techniques, and selected applications.

Our strategy is to test the technical and practical value of our ideas by building hardware and software prototypes and using them as daily tools. Interesting systems are too complex to be evaluated solely in the abstract; extended use allows us to investigate their properties in depth. This experience is useful in the short term in refining our designs, and invaluable in the long term in advancing our knowledge. Most of the major advances in information systems have come through this strategy, including personal computing, distributed systems, and the Internet.

We also perform complementary work of a more mathematical flavor. Some of it is in established fields of theoretical computer science, such as the analysis of algorithms, computational geometry, and logics of programming. Other work explores new ground motivated by problems that arise in our systems research.

We have a strong commitment to communicating our results; exposing and testing our ideas in the research and development communities leads to improved understanding. Our research report series supplements publication in professional journals and conferences. We seek users for our prototype systems among those with whom we have common interests, and we encourage collaboration with university researchers.

A Comparison of Two Distributed Disk Systems

Edward K. Lee
Chandramohan A. Thekkath
Chris Whitaker
Jim Hogg

April 30, 1998

Author Affiliations

Chris Whitaker and Jim Hogg are members of DIGITAL's OpenVMS engineering group at Livingston, Scotland. They can be reached at chris.whitaker@digital.com and jim.hogg@edo.mts.dec.com.

Abstract

As the storage needs of computer applications and users become more sophisticated and increase beyond what can easily be satisfied by a few disk array controllers, aggregating and managing the many disparate components of the storage system become severe problems. Distributed disk systems, which manage collections of disks shared by or partitioned across multiple nodes, may offer a solution to this problem by automating the management of these storage resources. Such systems can automatically tolerate and recover from component failures, gracefully scale in capacity and performance as components are added, allow the storage distributed across multiple nodes to be managed as a single system, and provide useful management abstractions such as *virtual disks* and *snapshots*.

This paper describes the different architectural and design alternatives embodied in Snappy Disk and Petal, two distributed disk systems that have similar external features but have radically different internal structures, and studies the effect of these differences on the systems' performance, availability, and manageability.

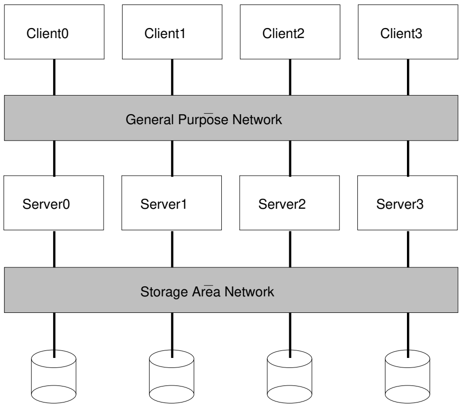


Figure 1: Shared-Disk Architecture

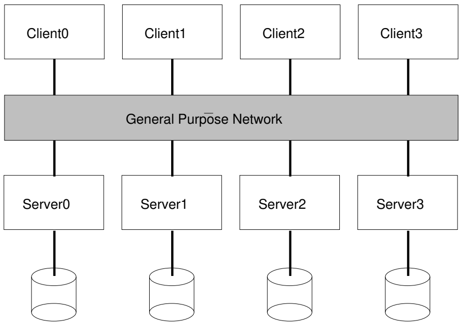


Figure 2: Partitioned-Disk Architecture

1 Introduction

As the storage needs of computer applications and users become more sophisticated and increase beyond what can easily be satisfied by a few disk array controllers, managing the many disparate components of the storage system becomes a severe problem. There has been considerable research devoted to designing large scale storage systems [1–3, 5, 8, 10, 14, 18, 19]. A *distributed disk system* is a class of storage system that can reduce the complexity of building and managing large scale storage systems. Distributed disk systems manage collections of disks shared by, or partitioned across, multiple nodes as a single logical storage system that is highly available, scales gracefully, and is easy to manage. Such systems can serve disk storage over a network to multiple file servers, database servers, and desktop machines. They can also be used for high-performance cluster-aware applications such as parallel databases and cluster file systems.

There are several ways to build distributed disk systems, but we know of no systematic study that compares the relative merits of the various architectural and design alternatives. Three important alternatives that distinguish these systems are how disks are physically connected to the nodes, how the system manages the bookkeeping information associated with the disks, and how users view the collection of disks. These alternatives can have a significant impact on the availability, performance, and manageability of the overall system.

As illustrated by Figure 1 and Figure 2, Distributed disk systems can be categorized as *shared-disk* or *partitioned-disk* architectures depending on how the disks are connected to the server nodes. In a shared-disk architecture, all server nodes can directly access all disks in the system with equal speed. In a partitioned-disk architecture each server node is directly attached to a subset of all the disks. Accesses to locally attached disks are significantly faster than accesses to remote disks.

Another distinguishing feature of distributed disk systems is the way they maintain bookkeeping information or *metadata* about the data stored on the various disks in the system. In a *shared-metadata* system, metadata can be accessed by all server nodes with equal ease. In a *partitioned-metadata* system, each node is responsible for maintaining a subset of the metadata that is stored locally.

Finally, distributed disk systems can be distinguished by how the users of the system view the collection of disks. Some systems support high-level abstractions called *virtual disks*, which use tables to flexibly map users' logical view of storage onto the physical disk resources, while other systems use less-flexible but more compact algorithmic mappings.

This paper attempts to describe and analyze the alternatives for distributed disk systems based on our personal experience in designing, implementing, and using two distributed disk systems: Snappy Disk [7] and Petal [13]. Snappy Disk and Petal provide similar functionality. Both support virtual disks and provide useful storage management features such as snapshots which allow system administrators to create nearly instantaneous copies of a virtual disk. The two systems differ significantly, however, in their architecture and design, and we believe that they occupy two near diametrical positions in the spectrum of architectural and design alternatives. Throughout this paper, we use comparisons between these two systems to drive our discussions. Section 2 provides a brief background and overview of Snappy Disk and Petal, Section 3 discusses the architectural and design issues embodied by Snappy Disk and Petal, and Section 4 analyzes the availability of shared-disk versus partitioned-disk architectures.

2 Snappy Disk and Petal

This section provides background information and a high-level overview of Snappy Disk and Petal. The next section will discuss the architectural and design differences between Snappy Disk and Petal in detail.

Snappy Disk is a shared-disk, shared-metadata distributed disk system that provides three main services. First, multiple nodes can share and simultaneously access a common pool of physical disks over a dedicated storage network. Second, clients can create virtual disks on demand. Third, nearly instantaneous copies of virtual disks, called snapshots, can be created on demand.

Snappy Disk does not directly implement mirroring, RAID [17], or other data redundancy schemes. However, Snappy Disk can easily manage disks that are already mirrored by separate software or exported by a RAID controller. Multiple

Snappy Disk nodes coordinate concurrent access to the shared disks by acquiring locks from a distributed lock manager that executes on one or more of the server nodes. Applications can either access Snappy Disk directly by running on the same nodes as the Snappy Disk subsystem, or over a general purpose network using a specialized network protocol. In either case, disks exported by Snappy Disk look just like locally attached disks to applications. Snappy Disk is currently available as a DIGITAL product that runs on OpenVMS and Windows NT.

Petal is a partitioned-disk, partitioned-metadata distributed disk system that currently runs on DIGITAL Unix and Windows NT. Similar to Snappy Disk, Petal also supports virtual disks and snapshots, although, unlike Snappy Disk, Petal snapshots are read-only. Petal is a research prototype for studying scalable distributed disk systems. A typical Petal system consists of several closely cooperating server nodes that communicate with each other over a network to implement a single logical storage system that is highly-available, scales easily in capacity and performance, and is easy to manage. Clients access Petal over a network using a custom RPC protocol. As with Snappy Disk, clients can run on the same nodes as the Petal servers, but because the disks are not shared between nodes, and most nodes cannot directly access a given disk, there is no compelling performance reason for doing so. In short, Petal is designed as a dedicated disk storage system that serves data blocks over a network to clients such as file systems and databases.

Hot-standby systems, such as Microsoft Wolfpack, are sometimes referred to as "clustered" systems, and fall between the spectrum covered by Snappy Disk and Petal. In these systems, several nodes simultaneously connect to a collection of disks via a shared storage interconnect. However, under normal operation, a designated node services all accesses to a particular disk. If the designated node fails, another node takes over its disks and duties by using the shared storage interconnect. However, a crucial distinction between these systems and shared-disk systems like Snappy Disk, is that they do not allow multiple nodes to access the same disk at the same time. This paper does not discuss hot-standby systems any further.

3 Architecture and Design

3.1 Shared- Versus Partitioned-Disks

The hardware structure of most distributed disk systems can be classified as either a shared-disk or a partitioned-disk architecture. In a shared-disk architecture, illustrated by Figure 1, the disks are attached to the server nodes via a storage area network. Any server node can access any disk directly over the storage area network. Because of its special purpose nature, the storage area network is often an interconnect, such as FibreChannel, that is optimized for transferring blocks of data, and support significantly higher data rates with lower CPU overheads than most general purpose networks. On the other hand, such interconnects often have limited scalability and are not as flexible as general purpose networks.

Figure 2 illustrates a partitioned-disk architecture in which a collection of client nodes access storage managed by a collection of server nodes. A system without distinct client nodes, in which applications run directly on the server nodes, is also feasible. In both cases, each disk is attached to a particular server node and only the node to which a particular disk is attached can directly access that disk. A node can access data on a disk attached to a different node only by transferring it across the general purpose network.

The type of hardware architecture used by a distributed disk system can have enormous implications for the system's software structure. Shared-disk systems such as Snappy Disk assume that unless a disk has failed, it is accessible from every server node. That is, the failure of one server node, does not affect the ability of another server node to access a given disk. To tolerate failures in the storage area network, the storage area network is often replicated and each server node and each disk redundantly connected to both storage area networks. The redundant data paths ensure that no single hardware failure, other than the failure of the disk itself, will make a disk inaccessible to a particular server node. Partitioned-disk systems such as Petal, on the other hand, typically identify and handle transient failures such as node crashes and communication failures that may make certain disks inaccessible for short periods of time. This can make software for partitioned-disk architectures more complex than software for shared-disk architectures.

Shared-disk systems often perform better than partitioned-disk systems constructed with similar disks and nodes because of the additional faster data paths offered by the storage area network. This is particularly true if the data are generated and consumed by applications running on the server nodes rather than being served to clients on the general purpose network. Also, when configured with redundant storage area networks, a shared-disk system can tolerate a wider combination of component failures than partitioned-disk systems. On the other hand, the additional networking hardware and the multiple data paths needed by shared-disk systems make such systems more expensive, harder to configure, and difficult to distribute over multiple geographic sites. In contrast, partitioned-disk systems are more easily distributed over multiple sites because they only require access to a general purpose network and, by necessity, are designed to handle intermittent communication failures between nodes.

3.2 Address Mapping

Storage systems that manage multiple disks typically support virtual disk abstractions to simplify management [6, 19]. A virtual disk can span multiple physical disks for performance, mirror data for reliability, and can implement a variety of administrative policies. When clients access a virtual disk, the virtual disk addresses must be mapped to the corresponding physical disk addresses. Most storage systems use simple mathematical equations to map virtual addresses to physical addresses [12]. Mappings based on mathematical equations are compact and simple to maintain but are inflexible. The mapping as well as the physical disks over which the mapping applies must be specified when the virtual disk is created. New physical disks cannot easily be added to existing virtual disks and the creation of new virtual disks usually requires adding physical disks.

An alternative to using mathematical equations is to use tables to map individual *pages* of storage. Mappings based on tables are more difficult for the system to maintain and can become quite large, but are much more flexible. The physical disks in the system are usually organized into storage pools that any virtual disk can draw on as needed. New disks can be added to the storage pool without disrupting existing virtual disks. The mapping between virtual to physical addresses can be done on demand as in modern virtual memory systems, reducing the amount of physical storage that would otherwise be over-allocated to a virtual disk while waiting for users or applications to fill up the space over time.

Another important advantage of table-based mapping is its ability to efficiently *snapshot* virtual disks using copy-on-write techniques currently used by virtual memory systems. When a virtual disk is snapshot, an identical copy of the virtual disk is created nearly instantaneously. Once a snapshot has been made, it behaves as an independent copy of the original. In particular, modifications to the original have no effect on the snapshot and modifications to the snapshot have no effect on the original. In some cases, snapshots may be read-only and cannot be modified.

Both read-only and writable snapshots are highly useful for managing large-scale storage systems. For example, a snapshot copy can be used for making consistent backups while applications continue using the original copy of the data. The snapshots can also be kept on-line so that users can easily restore accidentally deleted files and directories. Additionally, writable snapshots can be used to test new software using a copy of the actual data without investing a lot of time and resources in making a separate physical copy of the data.

The advantages of table-based mapping and snapshots are not free. Space is needed to store the mapping tables, and operations such as writing a block of data to a virtual disk may incur additional overheads, including additional disk writes, if mapping tables must be updated. For example, if we map a virtual disk using

64KB pages and each page requires 16 bytes of mapping data, then 1 TB (10^{12}) of physical storage would require 256 MB of mapping data. A key consideration in the design of distributed disk systems, is how this mapping data will be stored and accessed, particularly in the face of node and network failures.

3.3 Shared- Versus Partitioned-Metadata

Distributed disk systems that use simple mathematical equations to map virtual disks have the advantage that the mapping information is compact and rarely changes. This information, therefore, can be efficiently replicated and kept up-to-date across all the server nodes in the system. In contrast, systems that use table-based mappings must manage much larger amounts of mapping information, which may change frequently.

In a *shared-metadata* system, the information that maps virtual disks to physical disks is directly accessible to all server nodes in the storage system. The server nodes typically synchronize access to the metadata using a distributed lock manager. In a *partitioned-metadata* system, most of the mapping information is partitioned across the server nodes, and each node may only access the particular piece of metadata for which it is responsible. Even in a partitioned-metadata system, however, a small amount of the mapping information must be shared and globally available to all the server nodes in the system, if only to determine how the metadata is to be partitioned across the nodes. Snappy Disk is an example of a shared-metadata system while Petal is an example of a partitioned-metadata system.

Snappy Disk stores all its metadata including mapping information redundantly on shared disks. Server nodes coordinate accessing and caching this information using a fault-tolerant distributed lock manager. This allows the system to use a simple programming model that is similar to that used by shared-memory multi-threaded programs that synchronize using in-memory locks. The main design issue is in selecting an appropriate locking granularity. A granularity that is too coarse will limit throughput by restricting concurrency and generate false lock conflicts while a very fine locking granularity will generate large amounts of lock traffic. As an optimization, Snappy Disk updates to data structures can be piggy-backed on to lock acquisition and release messages [15]. When a lock is released, any modifications to the locked data structure are piggy-backed onto the lock using a "change record". When another node acquires this lock, the node applies any change records associated with the lock before allowing access to the data. This technique allows the use of a few locks to efficiently protect large tables, where only a few entries change between lock acquisitions. Without this technique, one would have to reload the entire table every time a lock changed possession even if only a few entries in the table had changed.

Petal distinguishes between two types of metadata. *Local metadata* makes up the bulk of the metadata in Petal and includes any information that is needed only by a single node and therefore does not need to be shared with other nodes. In particular, the loss of local metadata affects only the node on which that information is stored. Because Petal partitions its mapping information across nodes such that each node only contains the mapping information for the disks that it manages, all the mapping information is treated as local metadata.

Global metadata includes information such as which server nodes are currently a member of the Petal system and how the local metadata is partitioned across the nodes. Global metadata is frequently read by all the server nodes in the system, but is infrequently updated. Leslie Lamport's Paxos algorithm [11] is used to consistently replicate and update the global metadata across all server nodes in the system.

3.4 Data Access and Redundancy

This section examines the issues in providing data redundancy in shared-disk and partitioned-disk systems. Although the discussions focus on mirroring-based data redundancy, most of it applies to other redundancy schemes such as parity.

Consider the problem of providing single copy semantics while reading and writing mirrored data. Single copy semantics requires that the system behave as if it were operating on a single copy of the data. If read and write requests execute sequentially, achieving single copy semantics is straightforward. Each write request updates both copies of data and a read request may read from either copy of data. However, if read and write requests can execute concurrently, things are not so simple. Without explicit synchronization, certain orderings of disk reads and writes will violate single copy semantics. One could easily end up in a state where the two copies of data are different.

To provide single copy semantics in the general case, shared-disk systems acquire a global lock from a distributed lock manager to prevent a write request from executing at the same time as another write request or read request to the same data. Acquiring a global lock may require exchanging messages over a network. In partitioned-disk systems, because the two copies of data may only be accessed by the two nodes to which each of the two disks is attached, only local locks are needed. In Petal, for example, reads can be satisfied by either copy of data and acquire a reader lock for the duration of the operation. For writes, one copy is designated the primary, and writes must initiate at the node containing the primary copy. That node acquires a local writer lock and concurrently writes the data to its disk and to the node containing the secondary copy, which in turn acquires a local writer lock while writing its local disk. The node containing the primary copy then waits for both disk writes to complete before releasing its lock and acknowledging completion of the request.

In certain circumstances, it is not necessary to perform synchronization in the distributed disk system to ensure single image semantics. One such circumstance is when it is known that the distributed disk system itself is not performing any read or write requests to reorganize disk storage, and it is known that the application using the virtual disk never issues concurrent disk requests to the same data. The buffer cache manager for many file systems provide this guarantee as well as certain parallel database programs that do their own distributed locking.

For both shared-disk and partitioned-disk systems, adding non-volatile RAM (NVRAM) can improves the latency to perform client write operations and any operations that require updating persistent metadata. In both cases, for availability reasons, data must be stable on the NVRAM of at least two nodes before a write can be acknowledged as completed. In partitioned-disk systems, since only a single node can access a particular disk, and the data being written must go to the two nodes containing the primary and secondary copies, NVRAM can be transparently added to accelerate writes to a node's local disks. With shared-disk systems, things are a little more complicated. Because a single node would usually perform both disk writes, the data being written usually never reaches another node. Thus, additional communication with a second node is needed to guarantee two copies of the data in the NVRAM of two different nodes.

Also, a partitioned-disk systems may release its local locks as soon as the data has been written to NVRAM. In contrast, any global locks held by a shared-disk system to guarantee single copy semantics must be held until the data has been written to the disks or a mechanism must be provided whereby other nodes that try to access the same data are notified that the most recently written data may still be in NVRAM. Finally, in a shared-disk system, some coordination between nodes is needed to manage the allocation of NVRAM buffers. There are efficient ways to do this, however, which does not require additional messages.

3.5 Communication Overheads

This section discusses the communication overheads among disks and server nodes in shared-disk and partitioned-disk architectures. For the purposes of the comparison, we will assume a redundant configuration in which data is mirrored on two disks. As previously mentioned, shared-disk architectures tend to have lower communication overheads than partitioned-disk architectures due to the additional communication paths offered by the storage area network and the fact that the storage area network is often an interconnect that is optimized for transferring blocks of data with low CPU utilization. Because of this, we will assume in our discussions that it is preferable to use the storage area network in place of the general purpose network whenever possible.

We consider what happens in each architecture when the I/O is performed by applications running remotely on separate client nodes and when the I/O is performed locally by applications running on the server nodes. We are mainly interested in the number of messages and the number of times that data must be transferred over the general purpose and storage area networks. Table 1 summarizes the results.

A remote read in a shared-disk architecture is initiated by sending a read request message from a client node to a server node. Since all the server nodes are connected to all the disks, the request message may be sent to any of the server nodes, although to improve caching, it may be desirable to restrict requests for particular data to particular server nodes. The server node that receives the request then reads the data from an appropriate disk and returns the data to the client node. The entire transaction requires a minimum of four *messages*: a request-response pair over the general purpose network and a similar request-response pair over the storage area network. Furthermore, the above transaction includes a minimum of two data transfers: once over the storage area network and once over the general purpose network. Here, we assume that the data is piggybacked with the reply messages rather than being sent in separate messages. Depending on the physical networking technology and communication protocols, the exact number of messages needed to perform a remote read operation may vary from system to system; however, the above analysis is still useful for comparing architectures.

A remote read in a partitioned-disk architecture is processed in the same way as a remote read in a shared-disk architecture and incurs the same communication overheads. In a partitioned-disk architecture, however, server nodes access disks over individual local storage links rather than a unified storage area network. To simplify terminology, we refer to the collection of local storage links as a degenerate form of storage area network. The read request in a partitioned-disk system cannot be sent to any server node as in a shared-disk architecture, but must be sent to a server node that stores a copy of the requested data.

A remote write in a shared-disk architecture is initiated by sending a write request from a client node to any server node. The server node then writes the data to the two designated disks and acknowledges the request. This results in a total of two messages and one data transfer over the general purpose network and four messages and two data transfers over the storage area network. If NVRAM is supported, the data must also be sent to and acknowledged by a second server node, increasing the number of messages by two and the number of data transfers by one. Although these additional messages and data transfer may be performed across

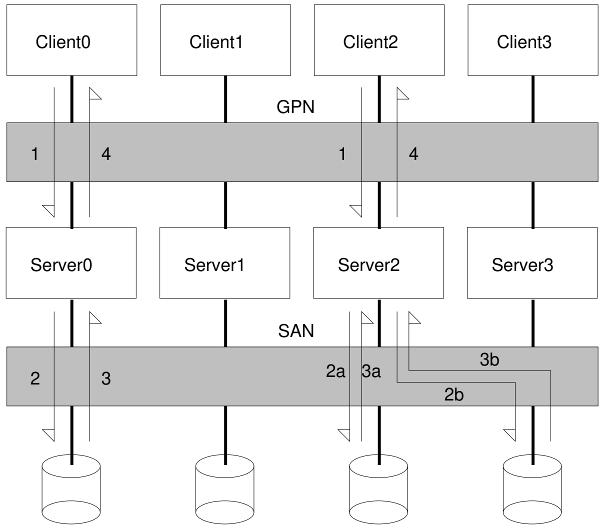


Figure 3: Shared-Disk Read and Write

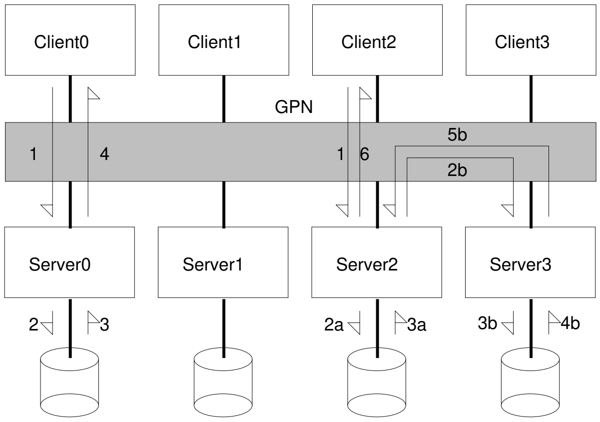


Figure 4: Partitioned-Disk Read and Write

Remote Access					**Local Access**				
	GPN		SAN			GPN		SAN	
	msgs	data	msgs	data		msgs	data	msgs	data
Read Snappy Disk	2	1	2	1	Read Snappy Disk	0	0	2	1
Read Petal	2	1	2	1	Read Petal	2	1	2	1
Write Snappy Disk	2	1	4 (6)	2 (3)	Write Snappy Disk	0	0	4 (6)	2 (3)
Write Petal	4	2	4	2	Write Petal	4	2	4	2

Table 1: Communication Overheads. *In the remote case, applications run on client nodes which request data from a server node over a network. In the local case, applications run directly on the server nodes. All overheads assume that data is mirrored.* GPN *denotes a General Purpose Network.* SAN *denotes a Storage Area Network. In the case of Petal, the collection of local storage links are treated as a degenerate storage area network. Numbers in the* msgs *columns denote the number of required messages. Numbers in the* data *columns denote the number of required data transfers. Numbers in parenthesis denote overheads with NVRAM support at the server nodes. In the remote case, Snappy Disk is more efficient for write requests than Petal. When supporting NVRAM, both Snappy Disk and Petal require the same total number of messages and data transfers. In the local case, Snappy Disk has a significant performance advantage for all cases because any server node can directly access any disk.*

either network, in most systems, it is preferable to use the storage area network.

A remote write in a partitioned-disk architecture is initiated by sending a write request from a client node to a particular server node. The server node then writes the data to its local disk and sends a write request to the server node containing the second copy. The server node acknowledges the request when both the local disk write and the write to the second server node have completed. This requires a total of four messages and two data transfers over the general purpose network, and another four messages and two data transfers over the storage area network. Because the data must be sent to two different nodes in the normal course of performing writes, supporting NVRAM in a partitioned-disk architecture does not incur any additional communication overheads.

Local read and write requests in shared-disk architectures have the advantage that all communication over the general purpose network can be eliminated with no additional communication over the storage area network. In partitioned-disk architectures, the communication overhead for local accesses is the same as that for remote accesses, unless the application happens to be collocated on the same server node as the data that it is accessing. This is unlikely in practice.

In the above discussion, we have ignored the issues of synchronization traffic and metadata updates that may be triggered by some I/O operations. A shared-disk architecture will typically use a distributed lock manager to synchronize read and write operations and updates to its metadata while a partitioned-disk architecture requires only local locks. Also, metadata updates in shared-disk architectures must be mirrored across multiple disks in order for the system to tolerate the failure of a disk containing the metadata, while in a partitioned-disk architecture, mirroring the metadata is not required since the loss of metadata stored at a particular node only affects that node, that is, the loss of metadata can be treated as a node failure. The amount of synchronization traffic over the network in a shared-disk system is highly dependent on the workload and the synchronization scheme. In practice, "well-behaved" workloads are unlikely to generate large amounts of synchronization traffic.

3.6 Summary and Conclusions

The separate storage area network and multiple disk paths give shared-disk architectures a performance advantage over partitioned-disk architectures. This advantage is most significant when applications run directly on the server nodes. The densely connected storage area network, however, can make shared-disk architectures difficult to distribute across multiple sites.

Table-based virtual to physical mappings are more flexible than algorithmic mappings for allocating physical disk storage and can be used to efficiently support

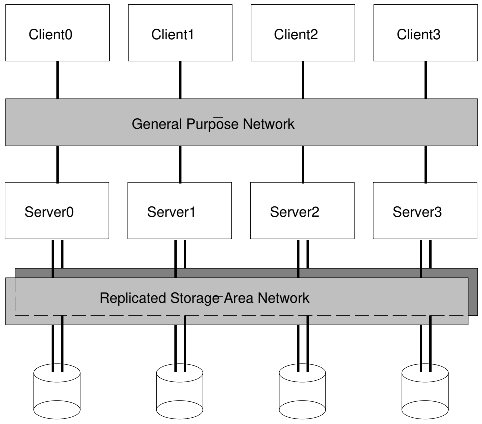


Figure 5: Redundant Shared-Disk Architecture

functionality such as snapshots. Table-based mappings, however, require maintaining significant amounts of mapping metadata.

Shared-metadata systems allow any server node to access any metadata and coordinate accesses using a fault-tolerant distributed lock manager. Partitioned-metadata systems, on the other hand, split the responsibility for managing this metadata across the server nodes in the system. Shared-metadata systems are conceptually simpler to design and program but their performance can be highly sensitive to the locking granularity and the workload. Partitioned-metadata systems find it easier to support fine-granularity locking and are much less sensitive to interactions between the locking strategy and the workload.

In conclusion, we believe that shared-disk, shared-metadata architectures such as Snappy Disk are most suitable for high-performance clustered systems where applications are expected to run on the server nodes. Partitioned-disk, partitioned-metadata systems, on the other hand, are most suitable for implementing stand-alone storage subsystems where flexibility and manageability are more important.

4 Availability Analysis

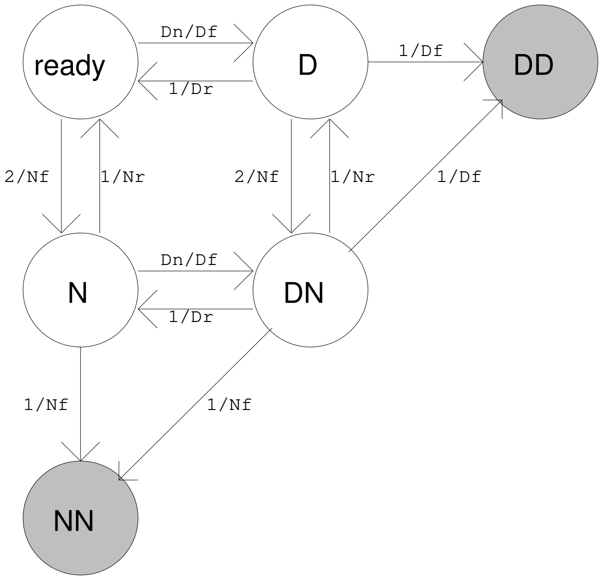
This section analyzes the availability of shared-disk and partitioned-disk architectures that use mirroring to protect data. Informally, a storage system is considered *available* if clients can access the data stored on the system; otherwise, the system is considered *unavailable*. Sometimes we will refer to the state in which a system is unavailable as a *system failure*. Many highly-available storage systems require a short period of time, around fifteen seconds to a couple of minutes in order to

D_n	$\equiv$	number of disks
D_f	$\equiv$	MTTF of a disk
D_r	$\equiv$	MTTR of a disk
S_n	$\equiv$	number of server nodes
S_f	$\equiv$	MTTF of a server node
S_r	$\equiv$	MTTR of a server node
N_f	$\equiv$	MTTF of a storage area network
N_r	$\equiv$	MTTR of a storage area network

Figure 6: Availability Parameters

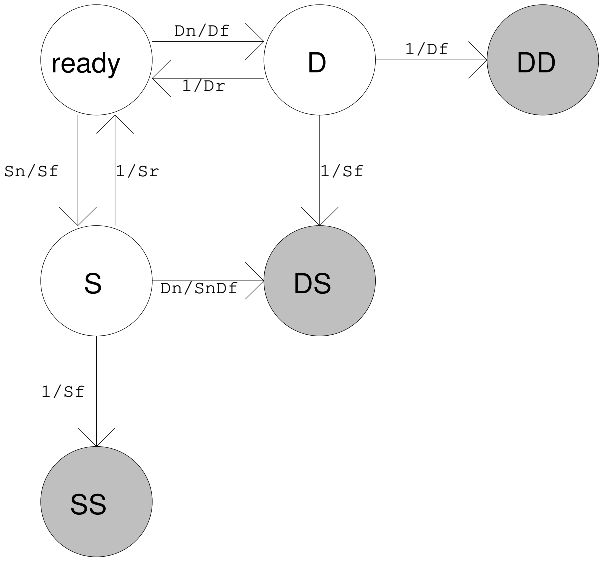
ready	$\equiv$	no failed components
D	$\equiv$	one failed disk
DD	$\equiv$	two failed disks
N	$\equiv$	one failed storage network
NN	$\equiv$	two failed storage networks
S	$\equiv$	one failed server node
SS	$\equiv$	two failed server nodes
DN	$\equiv$	one failed disk and one failed network
DS	$\equiv$	one failed disk and one failed node

Figure 7: Markov Chain States



$$MTTF_{sda} \simeq \frac{D_f{}^2 N_f{}^2}{D_n D_r N_f{}^2 + 2 N_r D_f{}^2}$$

Figure 8: Shared-Disk Availability Model. *Terminal states resulting in unavailability are shaded.*



$$MTTF_{pda} \simeq \frac{D_f{}^2 S_f{}^2}{D_n D_r S_r{}^2 + S_n S_r D_f{}^2 + D_n D_f S_f (D_r + S_r)}$$

Figure 9: Partitioned-Disk Availability Model. *Terminal states resulting in unavailability are shaded.*

detect and adapt to failed components. We will not consider these brief *fail-over intervals* as system failures.

Distributed disk systems can be made unavailable by a variety of causes including software crashes, disk failures, and network failures. Some of these causes such as disk failures are permanent and may result in data loss, but other causes such as software crashes may only make the system unavailable for short periods of time. In theory, systems without non-volatile memory can suffer data loss only as a result of disk failures. Other hardware components can be transparently replaced because they do not store persistent information; however, doing so in a real system can be difficult.

The rest of this section analyzes the availability of shared-disk versus partitioned-disk architectures using Markov chains. For the purposes of this analysis, we will assume a shared-disk architecture with replicated storage area networks as illustrated by Figure 5. The primary purpose of this analysis is to gain insight into the factors that affect the availability of each type of system rather than to accurately determine the availability of real storage systems. First consider the types of component failures that can affect the availability of distributed disk systems. As listed in Figure 7, these include server nodes, the storage area network, links to the storage area network, and disks. Both the shared-disk and partitioned-disk architectures protect against all single component failures, requiring multiple component failures to make data unavailable.

Figure 8 depicts a simplified Markov chain model that illustrates the major failure modes of a shared-disk architecture. This model ignores server node failures since in a shared-disk architecture, all server nodes must fail, a highly unlikely event, to prevent clients from accessing data. The model also ignores the failure of cables or links associated with the storage area network because this is highly dependent on the topology and, therefore, the implementation of the storage area network. The model illustrates the two failure modes that result in the vast majority of system failures. The first is the loss of two disks that contain the same copy of data, and the second is the loss of both storage area networks. The approximate mean time to system failure predicted by the model is displayed beneath the Markov state diagram.

Figure 9 presents the availability model for partitioned-disk architectures. There are three failure modes that result in the bulk of system failures. The first is the failure of two disks that contain the same copy of data. The second is the failure of two server nodes that contain the same copy of data. The third is the failure of a disk and a server node that contain the same copy of data. This model assumes that a disk on one server is mirrored by a corresponding disk on another server. That is, we assume that the data stored on a disk is not spread across a large collection of disks on another server; otherwise, the overall availability of the system could be

$$
\begin{aligned}
D_n &\equiv 100 \\
D_f &\equiv 1,000,000\,\text{hrs} \\
D_r &\equiv 1\,\text{hr} \\
S_n &\equiv 4 \\
S_f &\equiv 5\,\text{yrs} \\
S_r &\equiv 1\,\text{day} \\
N_f &\equiv 5\,\text{yrs} \\
N_r &\equiv 1\,\text{day}
\end{aligned}
$$

$$
\begin{aligned}
MTTF_{sda} &= 4544\,\text{years} \\
MTTF_{pda} &= 1065\,\text{years}
\end{aligned}
$$

Figure 10: Sample Availability Calculations. *Given a shared-disk system and a partitioned-disk system, each with a hundred disks and four server nodes, and the above values for the reliability of system components, the shared-disk system has an MTTF that is 4.3 times as large as the MTTF of the partitioned-disk system.*

significantly reduced.

The state diagrams and $MTTF$ equations in Figure 8 and Figure 9 have similar structures. This is because server nodes in partitioned-disk systems and storage area networks in shared-disk systems play a similar role in providing clients a pathway to data. If we replaced each occurrence of N in Figure 8 with an S, both the state diagrams and the equations would look very similar. A notable difference is that the failure of an appropriate disk and server node in a partitioned-disk architecture results in system failure, that is, DS is a terminal state, whereas the failure of a disk and a storage area network in a shared-disk architecture does not result in system failure, that is DN is not a terminal state. This is because the disks in the shared-disk architecture are dual-ported and can therefore be accessed via either storage area network whereas the disks in a partitioned-disk architecture are single-ported and can only be accessed via a particular server node. This accounts for the additional $D_n D_f S_f (D_r + S_r)$ term in the $MTTF_{pda}$ equation. We consider this the architectural availability advantage of shared-disk architectures when compared with partitioned-disk architectures.

Figure 10 compares the availability of the two architectures for representative values of the availability parameters.

- D_r is the mean time required to reconstruct data stored on a failed disk.
- S_f is the mean time between server node failures. These are typically hardware failures for which immediately rebooting the server node does not fix the problem.
- S_r is the mean time required to reconstruct data stored on a failed server node in partitioned-disk systems.
- N_f is the mean time between persistent failures of the storage area network. Actual values for this parameter will vary depending on the size, topology, and technology of the storage area network. We assume that its value is five years, which, in this case, is equal to S_f.
- N_r is the mean time to manually replace a failed storage area network. We assume that its value is twenty-four hours, which, in this case, is equal to S_r.

Based on these assumptions and parameter values, the models predict MTTF's of 4544 years for the shared-disk system and 1065 years for the partitioned-disk system. These numbers may sound large but consider that if system failures are exponentially distributed, an MTTF of 1065 years still means that there is a one percent chance of system failure in 10 years. The models' predictions are also based on the assumption that component failures are independent. Real systems, however, suffer from correlated component failures that significantly reduce the availability of the system. Examples of events that can cause correlated failures include, power failures, natural disasters, bad batches of disks with systematic manufacturing defects, and operators bumping into racks of equipment.

To close, we would like to reiterate that the primary purpose of this section has been to gain insight into the factors that affect the availability of shared-disk and partitioned-disk architectures rather than to derive models that accurately predict the availability of actual systems. Because of the factors already mentioned, estimating the availability of a particular distributed disk system is a complicated task that must include many intangible factors.

5 Summary and Conclusions

As storage systems become larger to meet the increasing storage needs of both existing and new applications, aggregating and managing the many disparate components of the system become severe problems. Distributed disk systems attempt to address this problem by automating the management of disks distributed across multiple server nodes. They automatically tolerate and recover from component failures, gracefully scale in capacity and performance as components are added, and allow multiple server nodes to be managed as a single system.

This paper has described the architectural and design alternatives for distributed disk systems as embodied in Snappy Disk and Petal, and studied the effect of these alternatives on the systems' availability and performance. In particular, we have examined shared-disk versus partitioned-disk architectures, shared-metadata versus partitioned-metadata approaches to managing bookkeeping information, the advantages and disadvantages of table-based virtual address mappings, and the mechanisms for synchronizing activity across multiple server nodes. The following paragraphs summarize our conclusions.

Most shared-disk systems use a separate storage area network to interconnect all disks and server nodes. The storage area network typically uses a different networking technology than the general purpose network. This technology is often optimized for transferring large blocks of data and provides higher-performance at the cost of limited scaling and decreased flexibility when compared with general purpose networks. Because of this separate high-performance storage area network, shared-disk systems can provide a significant performance advantage over partitioned-disk system. This is particularly true when applications run locally on the server nodes, where they can access disks directly over the storage area network without communicating with other nodes. Setting up and maintaining a separate storage area network, however, make shared-disk systems more expensive and difficult to partition across multiple geographic sites for disaster tolerance.

The performance advantage of shared-disk systems is reduced if the server nodes are required to support NVRAM (non-volatile RAM) buffers. To mirror data on a second server node, shared-disk systems must incur additional communication overheads, whereas partitioned-disk systems already incur this overhead in normal operation. As a result, if both systems are accessed remotely over a network, they will process the same number of messages and data transfers for each read and write request.

Shared-disk and partitioned-disk systems are likely to use very different synchronization strategies. For example, Snappy Disk shares all metadata between server nodes and uses a distributed lock manager with piggybacked metadata updates, while Petal partitions most of its metadata, uses only local locks, and relies on Leslie Lamport's Paxos algorithm for managing a small amount of metadata that must be shared across all the nodes. Partitioning the disks and metadata places restrictions on the choice of server nodes that a client must contact to access data, while sharing the disks and metadata without such restrictions may generate significant lock and metadata traffic as well as less effective data caching due to duplicate cache entries. The cost of acquiring locks over a network from a lock manager is much higher than acquiring a local lock. This makes the performance of shared-disk systems more sensitive to locality in the workload.

To tolerate all single component failures, shared-disk systems replicate the storage area network and attach each disk and server node to both storage area networks. Although more expensive, dual-porting all disk allows shared-disk architectures to tolerate a greater combination of multiple component failures than partitioned-disk architectures and gives shared-disk architectures an architectural advantage in system availability.

6 Acknowledgements

The authors would like to thank Mike Schroeder for his comments on earlier drafts of this paper.

References

[1] Thomas E. Anderson, Michael D. Dahlin, Jeanna M. Neefe, David A. Patterson, Drew S. Roselli, and Randolph Y. Wang. Serverless network file systems. *ACM Transactions on Computer Systems*, 14(1):41–79, February 1996.

[2] Luis-Felipe Cabrera and Darrel D. E. Long. Swift: Using distributed disk striping to provide high I/O data rates. *ACM Computing Systems*, 4:405–436, Fall 1991.

[3] Pei Cao, Swee Boon Lim, Shivakumar Venkataraman, and John Wilkes. The TickerTAIP parallel RAID architecture. *ACM Transactions on Computer Systems*, 12(3):236–269, August 1994.

[4] Peter M. Chen, Edward K. Lee, Ann L. Drapeau, Ken Lutz, Ethan L. Miller, Srinivasan Seshan, Ken Shirriff, David A. Patterson, and Randy H. Katz. Performance and design evaluation of the RAID-II storage server. *Journal of Distributed and Parallel Databases*, 2(3):243–260, July 1994.

[5] Wiebren de Jonge, M. Frans Kaashoek, and Wilson C. Hsieh. The logical disk: A new approach to improving file systems. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pages 15–28, December 1989.

[6] Wiebren de Jonge, M. Frans Kaashoek, and Wilson C. Hsieh. The logical disk: A new approach to improving file systems. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 15–28, 1989.

[7] A. Dewar, J. Hogg, C. Morrison, K. Playford, and C. Whitaker. *Snap Capable Disk: Version 1.0 Design Specification*. OpenVMS File System Technologies, August 1997.

[8] Garth A. Gibson, David F. Nagle, Khalil Amiri, Fay W. Chang, Eugene M. Feinberg, Howard Gobioff, Chen Lee, Berend Ozceri, Erik Riedel, David Rochberg, and Jim Zelenka. File server scaling with network-attached secure disks. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems*, pages 272–284, June 1997.

[9] Jim Gray and Andreas Reuter. *Transaction Processing: concepts and techniques*. Morgan Kaufmann, 1993.

[10] John H. Hartman and John K. Ousterhout. The Zebra striped network file system. *ACM Transactions on Computer Systems*, 13(3):274–310, August 1995.

[11] Leslie Lamport. The Part-Time Parliament. Technical Report 49, Digital Equipment Corporation, Systems Research Center, 130 Lytton Ave., Palo Alto, CA 94301-1044, September 1989.

[12] Edward K. Lee and Randy H. Katz. The performance of parity placements in disk arrays. *IEEE Transactions on Computers*, 42(6):651–664, June 1993.

[13] Edward K. Lee and Chandramohan A. Thekkath. Petal: Distributed virtual disks. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 84–92, October 1996.

[14] Timothy Mann, Andrew D. Birrell, Andy Hisgen, Chuck Jerian, and Garret Swart. A coherent distributed file cache with directory write-behind. *ACM Transactions on Computer Systems*, 12(2):123–164, May 1994.

[15] C. Mohan and I. Narang. Recovery and coherency-control protocols for fast intersystem page transfer and fine-granularity locking in a shared disks transaction environment. In *Proceedings of the 17th International Conference on Very Large Data Bases*, pages 193–207, September 1991.

[16] P. O'Niel. The Escrow transaction model. *ACM Transactions on Distributed Systems*, 11, December 1986.

[17] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the ACM SIGMOD*, pages 109–116, June 1988.

[18] M. Satyanarayanan. Scalable, secure, and highly available distributed file access. *IEEE Computer*, 23(5):9–21, May 1990.

[19] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan. The HP AutoRAID hierarchical storage system. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, pages 96–108, December 1995.