FLEXIBLE ALLOCATION AND SPACE MANAGEMENT IN STORAGE SYSTEMS

A Dissertation

by

SUK WOO KANG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2007

Major Subject: Computer Engineering

FLEXIBLE ALLOCATION AND SPACE MANAGEMENT IN STORAGE SYSTEMS

A Dissertation

by

SUK WOO KANG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	A. L. Narasimha Reddy
Committee Members,	Riccardo Bettati
	Deepa Kundur
	Warren Heffington
Head of Department,	Costas N. Georghiades

May 2007

Major Subject: Computer Engineering

ABSTRACT

Flexible Allocation and Space Management in Storage Systems. (May 2007)
Sukwoo Kang, B.S., Seoul National University; M.S., Seoul National University
Chair of Advisory Committee: Dr. A. L. Narasimha Reddy

In this dissertation, we examine some of the challenges faced by the emerging networked storage systems. We focus on two main issues. Current file systems allocate storage statically at the time of their creation. This results in many suboptimal scenarios, for example: (a) space on the disk is not allocated well across multiple file systems, (b) data is not organized well for typical access patterns. We propose Virtual Allocation for flexible storage allocation. Virtual allocation separates storage allocation from the file system. It employs an allocate-on-write strategy, which lets applications fit into the actual usage of storage space without regard to the configured file system size. This improves flexibility by allowing storage space to be shared across different file systems. We present the design of virtual allocation and an evaluation of it through benchmarks based on a prototype system on Linux.

Next, based on virtual allocation, we consider the problem of balancing locality and load in networked storage systems with multiple storage devices (or bricks). Data distribution affects locality and load balance across the devices in a networked storage system. We propose user-optimal data migration scheme which tries to balance locality and load balance in such networked storage systems. The presented approach automatically and transparently manages migration of data blocks among disks as data access patterns and loads change over time. We built a prototype system on Linux and present the design of user-optimal migration and an evaluation of it through realistic experiments.

To Yeonhee Doh and Donghyun Kang

ACKNOWLEDGMENTS

I would like to thank God, who makes all these things possible for me. I could not have come this far without his love and will.

I would like to thank my advisor, Dr. Reddy, for giving me a chance to work with him. During my study, he has been a great mentor who guided my research and motivated me to challenge myself. At the same time, he has been a great life teacher, who gives very considerate advice to help me whenever I need advice in my life. I cannot describe how much I appreciate his guidance, and I just would like to thank him again.

I would like to thank Dr. Bettati, Dr. Kundur and Dr. Heffington for their interest in this research and for their helpful suggestions. I would like to thank Dr. Eun Jung Kim and Dr. Leo Luan for giving invaluable advice every time I sought their opinion.

The other students in my research group have helped me throughout the course of my study. I would like to thank them all, especially Sumitha Bhandarkar, Yong Liu, Seongsoo Kim and Xiaonan Ma.

I would like to thank Donna and Warren. During my study, they have taken care of me, my wife and my child with all their heart. They also have taught us the Bible for three years to let us know God. I really appreciate their efforts and time. They are like my parents here in the U.S.

Without my wife, Yeonhee Doh, I doubt if it would ever be possible to pursue a Ph.D. I would like to express my deepest gratitude and love to her for her constant love and encouragement. My son, Donghyun Kang, gave me the hope and strength to get through all my courses. Finally, I would like to thank my parents and my in-laws for their full support and belief in me.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

A typical storage system architecture consists of various components in several system layers. As shown in Fig. 1, it includes a system call interface, a buffer cache, a file system, an operating system interface to block devices, a disk device driver, a disk controller and a set of disks. An application accesses the storage system through the system call interface. The buffer cache stores and manages recently accessed data blocks from disk in memory, so that additional accesses to them don't need to go to the disk whose access latencies are much larger than memory access latencies. The file system manages the contents on the disk. It exports a file and a directory structure to the application and handles the interface between the application and the disk. All operating systems provide interface to block devices. This block interface supports read and write operations and can carry out various operations such as remapping of block addresses. It interacts with the storage system device driver, which interfaces with disk controller. Disk controller manages the electronics of the disks by running firmware. It may also implement address remapping to provide logical storage characteristics as in Redundant Array of Inexpensive Disks (RAID [1]).

Traditionally disk devices are directly attached to the hosts or servers, i.e., through the Small Computer Simple Interface (SCSI) standard or the Integrated Digital Electronics (IDE) standard. Recently, they are being attached to networks, either Storage Area Networks (SANs) or IP networks. Recent industry wide efforts have established Internet SCSI (iSCSI [2]) as a standard to enable storage devices to

---

The journal model is *IEEE Transactions on Automatic Control*.

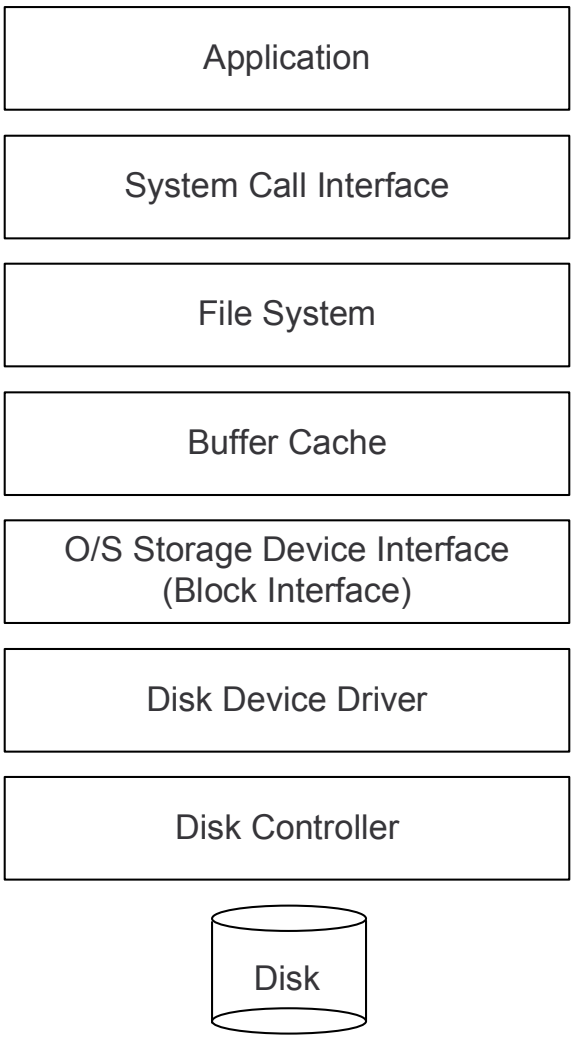


Fig. 1. Storage System Architecture

be directly attached to IP networks. The iSCSI protocol encapsulates SCSI command blocks in IP packets such that SCSI devices can be connected to servers and hosts through IP interfaces rather than SCSI buses. It is expected that such IP connectivity of storage systems will result in (a) improved scalability of I/O interconnects by leveraging the significant amount of research and development invested in scaling network throughputs, (b) enable consolidation of infrastructure around one network (instead of requiring separate Local Area Networks (LANs) and Storage Area Networks (SANs) with Fibre Channel), (c) enable wider (geographically) distribution and access of storage and (d) enable new storage paradigms (such as "storage as a service" [3]).

Early projects on connecting devices to networks have taken an application- specific or operating-systems specific approaches. Network attached disks have earlier been proposed to reduce the activity of a file manager in a file system in order to improve server throughput [4]. The Network Attached Secure Disks (NASD) project at Carnegie Mellon University (CMU) has built a prototype system which migrates much of file system functionality to the disk [4]. Commercially available SNAP server is an example network-attached storage system that understands specific file systems such as Network File System (NFS) [5]. Network attached disks have also been proposed to improve the I/O connectivity [6, 7, 8]. Recent iSCSI efforts focus on standardizing IP connectivity of SCSI disks based on the simple linear block addressable interface (of current disks).

Fig. 2 shows the different ways storage can be currently accessed. Disk A is attached directly to the client's system, disk B is attached to the server's system or to another host, disk C is attached to the IP network (through iSCSI protocol) and disk D is attached to the SAN network. The networked storage, SAN-based storage, and storage attached directly to systems (either locally or at the server) will

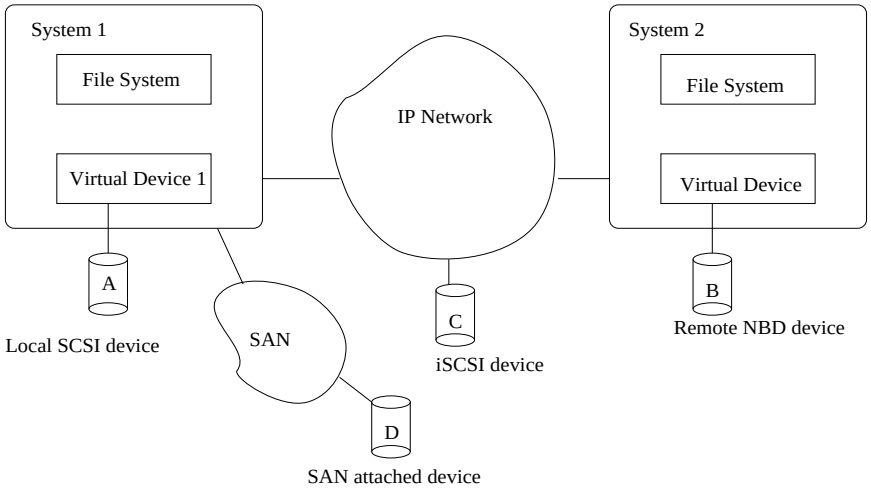


Fig. 2. Transparency over Multiplicity of Storage Device Attachments

likely coexist in the future. The increased diversity poses challenges to file systems and applications above the devices. Ideally, this diversity is hidden and the device attachment is made transparent to the layers above the device.

The IP connectivity of I/O devices also enables pooling of storage systems and data centers within a metropolitan area to improve sharing, utilization and administration of storage systems. In scientific communities, it is becoming increasingly important to share large data collections in collaborative environments. The Data grid [9] is an example of such a network of distributed storage resources.

If file systems and applications are designed to span multiple devices across a network, locality and the resulting performance issues will be of immediate concern. While flexible deployment may dictate that we do not worry about the actual location and attachment of storage, performance issues may force us to revisit data allocation and organization issues. The architectural and performance issues need to be considered in the light of networking storage devices and the consequent implications on the performance of storage systems, networks and their interaction. A database

application's performance may be significantly affected by the increased I/O delays if data is accessed over WANs [10]. In such cases also, it will be necessary to employ *data caching* or *data migration* to improve the performance. Previous work on iSCSI accesses over WANs has shown a similar need to cache data close to the application [11].

Current existing solutions such as Storage Resource Broker (SRB) [12] work as middleware between file systems and applications. Such approaches are oblivious to the storage location when storage is pooled at layers below the file system, for example using Logical Volume Managers (LVMs).

In this study, we focus on two main issues which are posed by the diversity of storage: (a) flexible storage allocation across multiple file systems/platforms and (b) enhancing data distribution considering locality and load balance in networked environments. The following sections provide a brief overview of the problems and why they motivate us. More detailed descriptions and our solutions to the problems are provided in the chapters that follow.

A. Storage Allocation

Storage Area Networks (SANs) and storage virtualization [13] allow storage devices to be shared by multiple heterogeneous operating systems. However, native file systems, such as Windows NTFS or Linux ext2, expect to have exclusive access to their volumes. In other words, each operating system reserves storage devices for its own use, and the space in a storage device owned by one operating system cannot be used by another. This problem seriously hampers the flexibility of storage management [14]. This lack of flexibility manifests in the following restrictions: (a) file systems cannot use storage space in another device owned by another file system, (b) a com-

puter can only create files on devices it owns. In Chapter II, we inspect this issue in detail and provide a general solution for improving the flexibility of storage allocation across multiple file systems. This is followed by detailed evaluation of the solution.

B. Data Distribution in Networked Storage Systems

In the next part of study, we focus on data distribution in networked storage systems. Network connectivity of storage systems has resulted in wider distribution and access of storage over network. Data distribution affects locality and load balance across the devices in a networked storage system.

In a large-scale storage system, it is especially important to (a) allocate data efficiently because the cost of data reallocation is high due to the large-scale and large network latencies and (b) redistribute data adaptively in an automated manner according to configuration changes or workload changes (after initial allocation) to improve performance. Most of current storage systems do static allocation of storage at the time of file system creation and do not provide the flexible data distribution. In Chapter III, we study this problem in more detail and propose a solution for improving data distribution in such networked storage systems. We study the behavior of the proposed solution via an implemented prototype system on Linux.

CHAPTER II

STORAGE ALLOCATION[1]

Traditional file systems are closely tied to their underlying storage. The file systems currently employ a static allocation approach where the entire storage space is claimed at the time of the file system creation. This is somewhat similar to running with only physical memory in a memory system. Memory systems employ virtual memory for many reasons: to allow flexible allocation of resources, and to share memory safely, etc. Traditional static allocation of storage at the time of file system creation lacks such flexibility. To address this problem and to enable storage as a discoverable resource, we have developed a *virtual allocation* technique at the block-level.

When a file system is created with X GBs, instead of allocating the entire X GBs of space to the file system and making it unavailable for others, virtual allocation only allocates storage space based on the current needs of the file system. Such an approach may allocate Y GBs of storage space, where Y could be smaller than X. The remaining storage space $(X - Y)$ GBs will be unused and available to be used flexibly. As the file system grows beyond Y GBs, the storage space can be allocated on demand or when certain usage thresholds are exceeded.

Such an approach separates the storage space allocation from the file system size and allows us to create virtual storage systems where unused storage space can be provided to any application or file system as the needs arise. This approach allows us to share the storage space across multiple (and possibly different) operating systems. The file systems can function transparently on top of such virtual storage systems

[1]©2006 ACM. Reprinted, with permission, from "An Approach to Virtual Allocation in Storage Systems" by Suk Woo Kang and A. L. Narasimha Reddy, published in ACM Transactions on Storage, Vol. 2, No. 4, pp. 371-399, November 2006.

as if they have access to a large storage device even though only a small fraction of that device is actually allocated and used at that time. It also makes it possible for a storage device to be expanded easily to other available storage resources because storage allocation is not tied to the file systems.

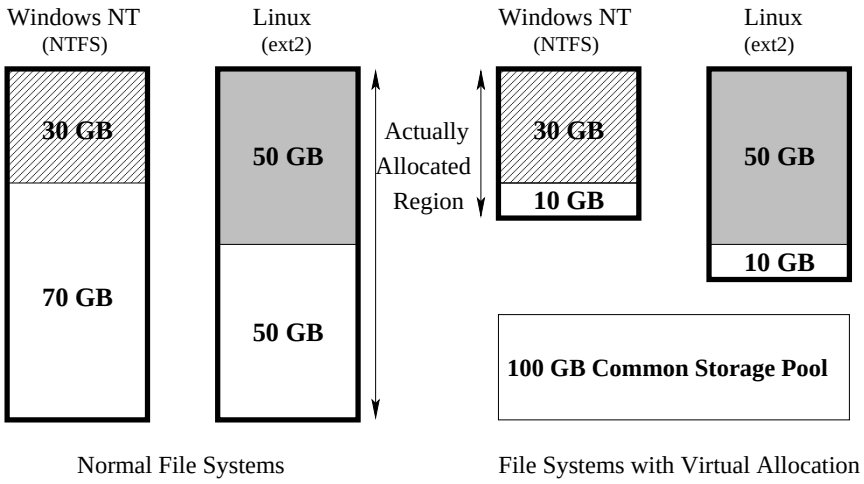


Fig. 3. Example Illustrating Virtual Allocation

Fig. 3 shows two example file systems with and without virtual allocation. In Fig. 3, highlighted regions illustrate the current usage of disks, and bold rectangles indicate actual allocations. The figure shows that unused storage space can be pooled across different operating systems and different file systems with virtual allocation.

Part of the motivation for our work here came from the observation that some of the students' PCs were running out of disk space while their neighbors' newer machines in the same lab had plenty of unused space. outlined as the following: The following issues further motivated our work: (a) Web hosting is a popular service and it is anticipated that similarly storage hosting could be sold as a service. These services provide storage for many users and allow server resources (disks, processors) to be shared across multiple users. While processor sharing is automatic (when one

user does not use the processor, the others get more time), storage sharing is not currently feasible. If a user asks for 100GB of space, but only uses 10GB, there is no way some other users can use that 90GB of space on disk. This is the primary problem our study is addressing. (b) Power and energy savings is another motivation. By only allocating space for actual data, the amount of space (and hence the number of disks) can be reduced, resulting in power and energy savings. (c) Utility computing initiatives can benefit from our approach by allocating storage as needed by users, rather than the maximum amount they will need. Different motivations for decoupling storage allocation from file systems have been put forward [15, 16].

Virtual allocation is developed to improve the flexibility of using the available storage. It is to be emphasized that virtual allocation is being pursued more to improve flexibility of storage management than to improve disk space utilization (which may be a side benefit). Virtual allocation has the following combination of characteristics:

- It uses the generic block interface widely used in today's systems. This allows it to support a wide range of heterogeneous platforms, and allows the simplest reuse of existing file system and operating system technology.

- It provides a mechanism to create virtual storage systems where unused storage space can be provided to any application or file system as a discoverable resource.

- It can be built on existing systems with little changes to operating systems.

In the following sections, we present the design rationale of virtual allocation and explain our experimental methodologies and results.

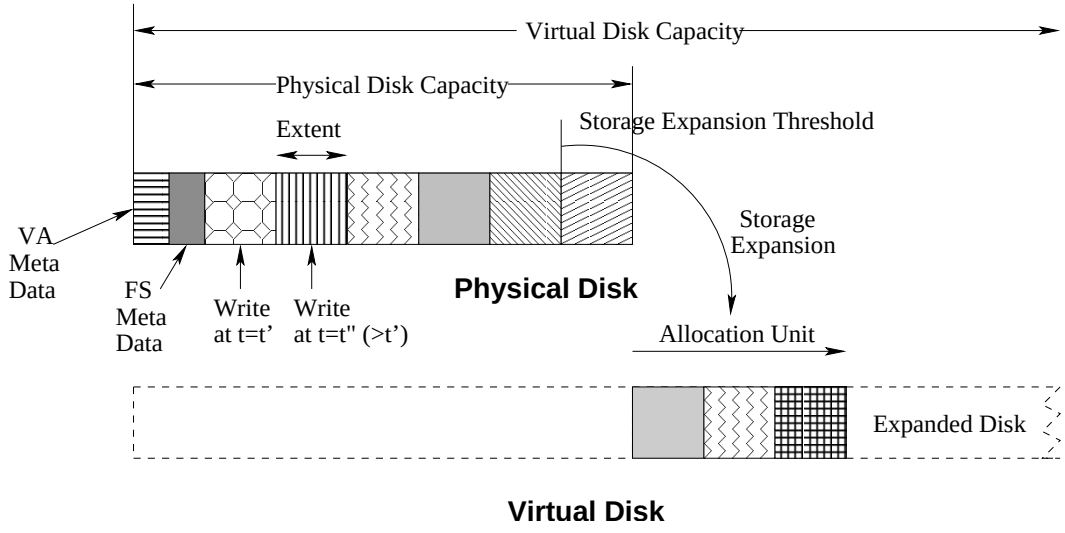


Fig. 4. Allocation Strategy of Virtual Allocation

A. Virtual Allocation : The Proposed Solution

In this section, we present the design rationale for virtual allocation and its component architectures. Virtual allocation employs an *allocate-on-write* policy, i.e., storage space is allocated when the data is written. Fig. 4 illustrates the storage allocation strategy of virtual allocation. The figure shows an example in which new data is written at time $t = t'$ and at time $t = t''$ where $t'' > t'$. In this example, virtual allocation writes all data to disk sequentially in a log-like structure based on the time at which data is written to the device. This approach is similar to log-structured file systems [17, 18] where every write is written at the end of a log. However, in virtual allocation, only storage allocation is done at the end of a log.[2] Once data is written to disk, data can be accessed from the same location, i.e., data is *updated in-place*. Virtual allocation maintains a block map for this purpose. Virtual allocation's block

[2]It is possible to employ other policies for space allocation other than sequential allocation.

map is similar to a logical volume manager's (LVM's) block map, i.e., it converts file system block addresses to actual physical device block addresses. However, virtual allocation's block map is dynamically constructed as data is written and not at the time of file system (or logical volume) creation. Virtual allocation can be seen as a generalization of LVM's functionality.

This block map data structure is maintained in memory and regularly written to disk for hardening against system failures. The on-disk block map, called *VA (Virtual Allocation) metadata*, is stored at the front of the allocation log, as shown in Fig. 4. Virtual allocation uses an *extent*, which is a group of (file system) blocks, to reduce the VA metadata information that must be maintained. The block map data structure contains information such as the logical device where the data belongs (*LDEV-ID*), the logical block address of the data (*LBA*), the physical device where the allocated extent resides (*PDEV-ID*), and the physical block address of the extent (*PBA*).

Table I. Example of a Block Map in Virtual Allocation

LDEV-ID	**LBA**	**PDEV-ID**	**PBA**
LDev0	100	PDev0	1000
LDev1	500	PDev0	1100
LDev0	1000	PDev1	1200

Table I shows an example of block map data structure. For example, the first row of Table I means that a logical block *100* of the logical device *LDev0* resides at the physical block address *1000* of the physical device *PDev0*. Each entry of the block map corresponds to one extent and whenever a new extent is written, VA metadata is hardened for data consistency in case of a system failure. There are other hardening

policies which will be explained later. If we use a 256KB extent, the size of VA metadata is less than 64KB per 1GB disk space.

File systems tend to create metadata at the time of file system creation. Due to our sequential allocation policy, file system metadata tends to be clustered in front of the allocation log. This *metadata cluster* is denoted by a single *FS (file system) metadata* region in Fig. 4. IBM Storage Tank takes an approach of separating metadata and normal data at the file system-level [14]. Our allocation policy tends to result in a similar separation of the file system metadata from the file system data.

Virtual allocation makes it easy for file systems to span multiple disks. Fig. 4 shows that when the capacity of the file system's disk is exhausted or reaches a certain limit (*storage expansion threshold*), the disk can be expanded to other available storage resources. This process of *storage expansion* allows file systems or applications to potentially span multiple devices. As the file system usage increases, more disk space is allocated transparently. The storage expansion is done in terms of *allocation units*. For example, if the allocation unit is 1GB, whenever the storage expansion is performed, virtual allocation adds 1GB of disk space to an existing disk by finding other available storage resources. With virtual allocation, it is possible to configure the storage systems such that the total physical capacity is smaller than the sum of sizes of the file systems. In such a case, care needs to be taken to not run out of physical space as file systems expand or mechanisms need to be provided in file systems to deal with the consequences of running short of physical space. We define a *virtual disk* as a storage device seen by the file systems or applications. File systems or applications can function transparently on top of such a virtual disk as if they have access to the complete storage space even though only a small fraction of that device is actually allocated and used at that time.

Virtual allocation is implemented as a loadable kernel module that can be in-

serted below any file system. Fig. 5 shows that virtual allocation is a layer between the file system and the disk device driver. Virtual allocation can be integrated into a LVM layer when it is present. When virtual allocation is stacked on top of the disk device driver, all I/O requests are intercepted by the VA module before being passed to the underlying disk. For a write request to an unallocated extent, virtual allocation dynamically allocates space for the extent and directs the request to that location. It adds this allocation information to the block map so that data can be accessed later from the same location. Further writes to the blocks in the same extent are written to allocated blocks in that extent. For a read request, the block map is consulted for mapping information. If it exists, it is directed to that location. Otherwise, it will be an illegal access because an accessed extent is not yet written. Such a read miss will not occur unless file system metadata is corrupted. We plan to address this exception in the future for applications that access the raw device.

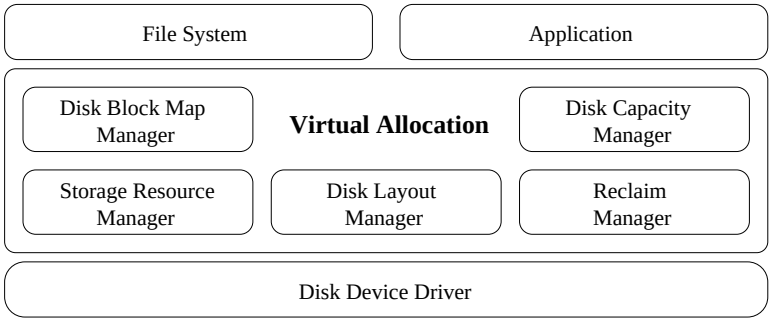


Fig. 5. Virtual Allocation Architecture

Fig. 5 shows the components of the virtual allocation architecture. The disk layout manager in Fig. 5 directs incoming read and write requests to appropriate locations, and the disk block map manager maintains this information as a block map. The disk capacity manager continuously monitors disk usage, and generates

a service message to the storage resource manager once the disk usage reaches the storage expansion threshold. If the storage resource manager is invoked, it tries to find available storage resources in the local computer or on the network. Once storage is discovered and obtained, the existing device will be expanded to incorporate the new storage. The reclaim manager is in charge of reclaiming allocated storage space of deleted files. Reclamation of blocks of deleted files allows the continued flexibility of virtual allocation and prevents abuse of virtual allocation as explained in later sections.

1. Design Issues

a. VA Metadata Hardening (File System Integrity)

As mentioned earlier, our in-memory block map must be hardened to disk regularly to protect against system failures. VA should be designed so that it does not compromise the safety or integrity mechanisms employed by the higher layers (file systems) or require changes to the higher layers. Since VA metadata hardening requires an additional disk access, we have to carefully decide *when* or *how often* VA metadata will be committed to disk to minimize the overhead.

In this study, we considered two kinds of VA metadata hardening policies: an *extent-based policy*, and a *file system-based policy*. The extent-based hardening employs the strict policy of writing VA metadata to disk whenever a new extent is allocated. The file system-based hardening policy exploits the update behavior of the file system and defers VA metadata commitment to disk whenever possible. File system-based hardening is less general[3], but it is expected to improve performance. We consider two file systems on top of VA to illustrate the metadata hardening poli-

[3]In Section 6, we discussed more general approach of file system-based hardening.

cies, e.g., the Linux *ext2*, and Linux *ext3* file systems.

In the extent-based policy, whenever an extent is allocated on the storage system (through file system writes), the VA metadata is hardened before the file system write is allowed to proceed. Such a policy ensures that VA metadata is always correct and does not compromise the integrity of the file system. This can be explained through an example illustration with Linux ext2.

Table II. An Example of the Extent-based Policy in VA with the ext2 File System

Operation	**In-memory**	**On-disk**
1. I alloc	$I \rightarrow B$	
2. B writes to disk		
2-1. V updates	$V \rightarrow B$	
2-2. V writes to disk		V written
2-3.		B written

Table II depicts a time line of operations when the new data block B is allocated to a file I (§1) and written to disk (§2). VA observes this request and allocates the new extent for B and updates its in-memory block map V (§2-1). VA commits this metadata change to disk (§2-2) before B reaches the disk (§2-3). The file system is consistent despite a crash at any point up to §2-1 because VA does not modify any normal operation of the file system (except the V updates in-memory which will be lost in case of the system crash, but it does not affect the file system integrity because data B is not written to disk yet). If the system crashes between §2-2 and §2-3, this also does not affect the file system integrity because B is not yet written to disk. Although VA allocates the new extent to B, this mapped extent will be reused when the restored system allocates data block B again. It is noted that VA metadata needs

to be hardened only when the extent is allocated, i.e., further writes to an allocated extent do not involve any VA metadata updates.

If we commit VA metadata V after B is written, the file system integrity could be broken. Assume that file system metadata I is written to disk before B and B is written to disk. If the system crashes at this point (before V reaches the disk), although the file system can track B through its inode I, our system cannot track data B due to mapping information loss, which results in broken file system integrity. If we keep update ordering of VA metadata and FS (meta)data as explained in Table II, the extent-based hardening policy can be used with *any* file system regardless of its update behavior without compromising the file system integrity.

Table III. An Example of the File System-based Policy in VA with the ext3 File System

Operation	**In-memory**	**On-disk**
1. I alloc	$I \rightarrow B_1$	
2. I alloc	$I \rightarrow B_2$	
3. I writes to journal		
3-1. B_1 writes to disk		
3-2. V updates	$V \rightarrow B_1$	
3-3. B_2 writes to disk		
3-4. V updates	$V \rightarrow B_2$	
3-5.		B_1 written
3-6.		B_2 written
3-7. V writes to disk		V written
3-8.		I written
4. I writes to disk		I written

We explain the file system-based policy with the ext3 file system. The ext3 file system is a journaling file system in which metadata consistency is ensured by write-ahead logging of metadata updates. Ext3 supports three journaling modes of operation. In one of the modes, in *ordered mode*, ext3 only logs changes to metadata to the journal, but flushes data updates to disk before making changes to associated metadata. Due to this property, underneath the ext3 ordered mode, we can optimize the VA metadata hardening policy by aggregating multiple VA metadata updates as depicted in Table III.

New data blocks B_1 and B_2 are allocated to a file I (§1, §2) and I is going to be written to a journal (§3). The ordering property of ext3 ordered mode ensures that B_1 and B_2 are written to disk first (§3-1, §3-3). At this point, VA can safely defer V commitment to disk before I is written to the journal because system crashes at any point make this operation invisible to the file system until I is written to the journal. After both data blocks are written to disk (§3-5, §3-6), VA commits V to disk (§3-7) before I is written to the journal (§3-8). Such delayed hardening may enable aggregation of VA metadata updates and reduce the number of associated disk accesses. To ensure that we are providing the same safety semantics as the ext3 ordered mode, we commit any VA metadata changes to disk before any FS metadata reaches the journal. Such an approach requires the identification of FS metadata at the block-level. We employed the file system layout discovery approach described in [19].[4]

[4]More general approach was discussed in Section 6.

b. Extent Size

An extent size is a configurable parameter in our system. Larger extents retain more spatial locality, and reduce the block map size that must be maintained, which results in reduction of the overhead of VA metadata hardening. However, larger extents may cause data fragmentation on the disk. Since even small requests are allocated an extent, storage space will be fragmented if following write requests are not sequential. The trade-offs with the size of an extent are similar to the trade-offs with the page size in virtual memory or the block size in cache memory systems.

c. Reclaiming Allocated Storage Space

VA needs a mechanism to reclaim allocated space of deleted files to continue to provide the benefits of virtual allocation. Without this mechanism, the storage space once allocated to one application (e.g., file system) cannot be used by other applications even if the allocated space is no longer used by that application.

Without reclamation, an application can write large files and delete them to turn virtual allocation into static allocation. Second, as the files are deleted and created through normal activity, if a large amount of block space is allocated while utilizing only a small portion of that block space, the effectiveness of virtual allocation will decrease over time. We conduct experiments to study this issue.

VA reclaims allocated space of deleted files employing *dead* block finding approaches described in [19, 20].[3] When the reclaim operation is invoked, VA gathers the information about dead blocks and looks up our block map to free dead blocks in the allocated extents. When all the blocks in an extent are dead, that extent can be reclaimed for reuse. The frequency of the reclaim operation must be decided carefully considering the file system activity. It is to be noted that file systems can continue

working without reclamation of free space at the VA layer, (but without the benefits of virtual allocation) and hence is not as critical an operation as garbage collection in log-structured file systems.

d. Chunk Size in RAID

When our system is used with RAID, VA is layered on top of RAID. For each read or write request, VA first remaps the block address, then RAID software or RAID hardware layer handles this request according to its RAID policy, e.g., RAID-5.

RAID systems do considerable work in choosing the chunk size for optimizing the performance [1]. In this study, when VA is used with RAID, the extent size of VA is chosen to be the same as the chunk size of the RAID layer in order to simplify the allocation. The write overhead of RAID systems and the metadata hardening overhead of virtual allocation impact the choice of ideal chunk/extent size. In this study, we consider RAID-5 systems and study the impact of the choice of chunk/extent size on the performance of a system employing VA along with RAID.

2. Spatial Locality Issues

a. Metadata and Data Separation

Our storage allocation policy changes the on-disk layout of a file system located on top of our system. One of the major differences between the system with virtual allocation and the system without virtual allocation is that in the former, metadata is separated from data. File systems put significant effort into keeping the metadata in close proximity to the data. Earlier studies [19, 21, 22, 23] have shown the importance of exploiting spatial locality on disks. Hence, it is necessary to closely evaluate our approach in light of these studies.

In this study, we compared three kinds of systems:

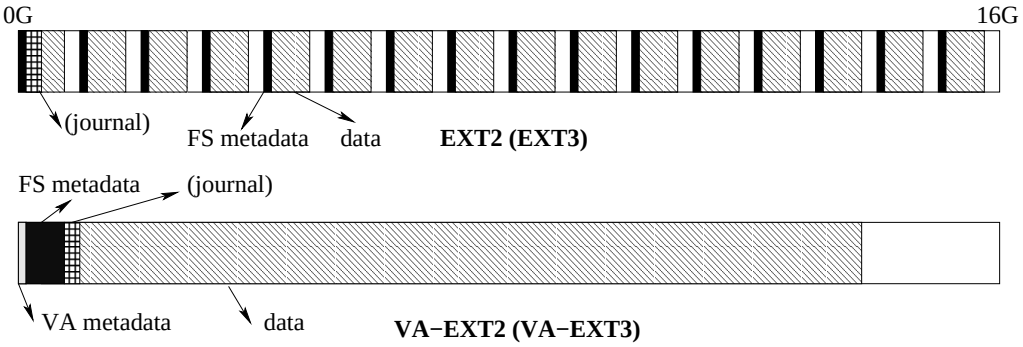
- Linux ext2 with virtual allocation (**VA-EXT2**) and ext2 with a normal storage system (**EXT2**)
- Linux ext3 with virtual allocation (**VA-EXT3**) and ext3 with a normal storage system (**EXT3**)
- RAID-5 with virtual allocation (**VA-RAID-5**) and a normal RAID-5 storage system (**RAID-5**)

In the case of the ext3 file system (VA-EXT3 and EXT3), we used, by default, ordered mode unless specifically noted. In RAID-5 configurations (VA-RAID-5 and RAID-5), we used the ext2 file system.

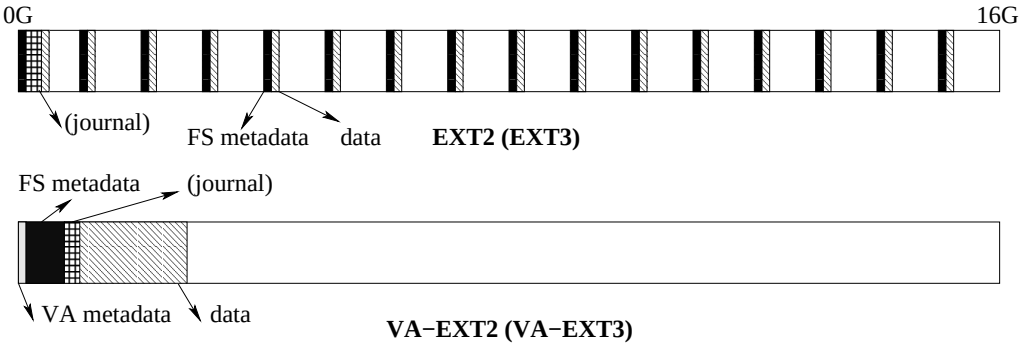
Fig. 6 shows differences of the on-disk layout between EXT2 (EXT3) and VA-EXT2 (VA-EXT3). The on-disk layout of EXT3 is the same as EXT2 except for a journal; the journal (or log) is commonly stored as a file within the file system, although it can also be stored on a separate device or partition. Fig. 6 depicts the journal stored as a file, which is the default in ext3. Fig. 6(a) shows the on-disk layout of a 16GB partition at 85% file system utilization and Fig. 6(b) shows the on-disk layout of the same partition at 10% file system utilization. In our system, all file system metadata is clustered in front of the allocation log, and the data is written with temporal locality after this metadata cluster. We expect that this metadata and data separation will affect the performance of different workloads differently.

b. Data Clustering

Due to our storage allocation policy, all written data are clustered in VA, as shown in Fig. 6. Data clustering will improve performance of VA-EXT2 compared to EXT2 by



(a) 85% File System Utilization



(b) 10% File System Utilization

Fig. 6. Illustration of the On-disk Layout Differences Between EXT2 (EXT3) and VA-EXT2 (VA-EXT3)

reducing seek distances. The effect of the clustering will appear differently according to the partition size and the file system utilization. At low file system utilization, VA-EXT2 can reduce seek distance significantly compared to EXT2, as shown in Fig. 6(b). As the file system utilization gets higher, the benefit of clustering in VA is likely to decrease.

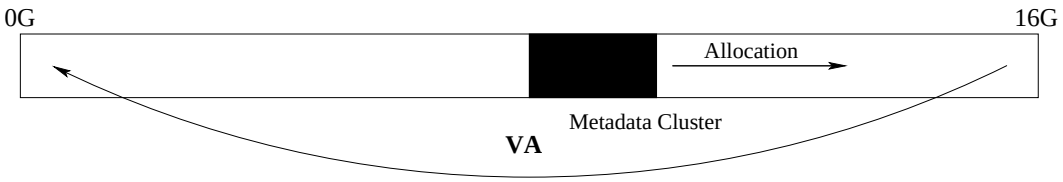


Fig. 7. Illustration of the Data Placement Policy

c. Space Allocation Policy

Fig. 6 shows a linear allocation of space on the device. Modern disk drives have higher transfer rates in outer cylinders due to the *zoned constant angular velocity* (ZCAV) techniques [24]. When the linear allocation starts in outer cylinders, the metadata cluster can exploit the higher data rates to improve file system performance.

Alternatively, the linear allocation of space can start in the middle of the disk as shown in Fig. 7 and wraparound. This may reduce average seek distances to metadata by placing metadata in the middle of the disk.

d. File System Aging (Fragmentation)

Earlier file systems have used clustering to improve performance [25, 26]. These studies indicated that clustering can improve performance on empty or new file systems. As the file system ages, free space on the disk becomes fragmented. This fragmentation degrades the spatial locality and hence the performance of the file system. VA

may similarly experience fragmentation as files expand through append operations. In order to understand the long term effectiveness of our system, we have to consider the effect of fragmentation due to file system aging [27].

e. Multiple File Systems

Currently, if multiple file systems are used, the number of partitions must equal the number of file systems. Each file system resides on its own partition and uses its own dedicated disk space. In contrast to this fixed partition approach, the data of different file systems are written to disk sequentially with temporal locality if multiple file systems are used with virtual allocation, i.e., all data are intermixed on the disk regardless of partitions.

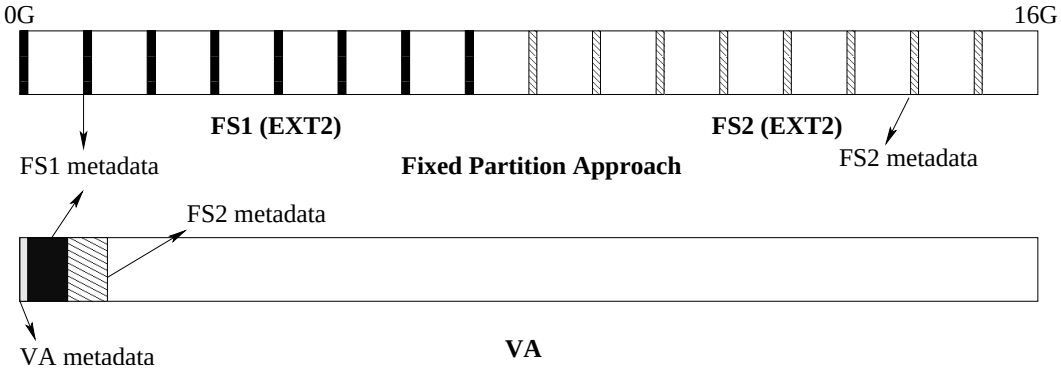


Fig. 8. The On-disk Layout Difference Between VA and the Fixed Partition Approach

Fig. 8 shows one example of the on-disk layout comparing the fixed partition approach with our approach. In this example, two partitions are used. First, one file system (FS1) is created on the first partition and then the other file system (FS2) is created on the second partition. As shown in Fig. 8, in the fixed partition approach, each EXT2 FS owns its partition and distributes its metadata. On the contrary, in our approach, FS1 metadata is written in front of the allocation log and FS2 metadata

is written right after the FS1 metadata cluster.

3. Performance Issues

There are four factors that affect the performance of VA. The first factor is VA metadata hardening. The extra synchronous writes required for VA metadata hardening may impact the overall performance. The second factor is the seek distance. Since we change the spatial locality of the data on the disk, seek distances are different from those of a normal storage system. The third factor is related to space allocation policy, i.e., in VA, data observe different data rates than in a normal storage system. The last factor is the in-line overhead. VA has to consult its block map for every I/O request, which is not the case in normal storage systems. These four factors impact the performance of VA.

B. Implementation

We developed a prototype of virtual allocation as a kernel module for Linux 2.4.22 based on the layered driver architecture. For the dynamic redirection (or remapping) of disk I/O requests, the virtual allocation module must have the capability of observing all disk I/O requests. For this reason, we place VA between the operating system block device support routines and the disk device driver. The in-memory block map was implemented as a hash table. Each hash entry consists of 24bytes; 4bytes for LDEV-ID, 4bytes for PDEV-ID, 4bytes for LBA, 4bytes for PBA, and 8bytes for doubly-linked lists.

C. Evaluation

In this section, we compare the following systems: **EXT2** and **VA-EXT2**, **EXT3** and **VA-EXT3**, **RAID-5** and **VA-RAID-5**. We used four workloads in our experiments. The first workload was sequential reads and writes of large files of Bonnie++ benchmark [28]. We used test files of various sizes depending on the experiment and used a 4KB chunk size. Bonnie++ sequentially writes a test file to disk and then sequentially reads it by the unit of the chunk size.

The second workload we chose was Postmark [29]. We configured Postmark to create files between 8KB and 64KB in a number of directories and perform 100,000 transactions. This file size range matches file size distributions reported in the file system studies [30]. The number of directories and files are varied depending on the experiment. In each case, the number of directories chosen is sufficient to span the entire partition. Postmark focuses on stressing the file system by performing a series of file system operations such as file creations, deletions, reads, and appends.

The third is the TPC-C benchmark developed for testing the performance of database systems running OLTP workloads by the Transaction Processing performance Council (TPC) [31]. The scale of the TPC-C benchmark is expressed in terms of the number of warehouses represented. The database used in this study contains 16 warehouses. We used the Oracle 10g database with the Hammerora open source TPC-C script [32].

As the fourth workload, we employ NFS trace with a replay tool. We used one hour of Harvard EECS trace [33] which is a research workload from a university Computer Science Department and employed TBBT [34] as a replay tool.

1. Experimental Setup

All experiments were performed on a commodity PC system equipped with a 3GHz Pentium 4 processor, 900MB of main memory, and two kinds of 10,000 RPM Seagate SCSI disks (ST3146807LW: 147GB, ST336607LW: 37GB) controlled by the Adaptec SCSI Card 29160. All single disk experiments were performed on a disk of 147GB size except the experiments of the space allocation policy and the multiple file systems, where a disk of 37GB size was also used. In RAID experiments, three disks (one 147GB disk and two 37GB disks) were used to form RAID-5 array, and a Linux Software-RAID driver was used.

The operating system was Red Hat Linux 9 with a 2.4.22 kernel, and the file system was the ext2 or the ext3 file system depending on the experiment. All experiments were run on an empty file system except the file system aging experiments. Total accessed data of each test was much larger than the system RAM (900MB). We ran all tests at least ten times, and computed 95% confidence intervals for the mean throughput.[5]

2. Evaluation Issues

As mentioned earlier, the ZCAV effect can skew benchmark results enormously. Since the effect of the allocation policy that we measure may be subtle, we had to reduce ZCAV effects as much as possible [35]. Where necessary, to reduce ZCAV effects, we used a partition of 7GB size (using outer cylinders) on the 147GB disk. There was less than 0.2% performance variation due to ZCAV effects on up to 7GB of the disk. The 16GB partition of the disk showed 2.4% performance variation, and the 32GB partition showed 5.4% variation. All these results were measured by ZCAV

[5]Where necessary, we showed the confidence interval as y-error bars in the results.

benchmark [36].

To see the impact of disk arm movements in a heavily utilized disk, we also used a partition of 32GB size on the 37GB disk and compared results with the other configuration (a 7GB partition on the 147GB disk) in the experiments of multiple file systems.

We measured the in-line overhead due to the dynamic remapping of block addresses by employing a Postmark benchmark.[6] Postmark was configured to create 50,000 files (between 8KB and 64KB) and perform 100,000 transactions in 200 directories. The performance overhead ranged from 0.6% to 3.3% compared to the normal storage system. All following VA experimental results reported here include this overhead.

3. Impact of Various Factors

a. VA Metadata Hardening

We measured the impact of VA metadata hardening for a large-file workload (Bonnie++) and a small-file workload (Postmark). We did experiments on ext2 and ext3 with two VA metadata hardening policies, i.e., an extent-based hardening (with ext2 and ext3) and a file system-based hardening (with ext3). We used a test file of 2GB size in Bonnie++, and Postmark was configured to create 50,000 files (between 8KB and 64KB) and perform 100,000 transactions in 200 directories. The 512KB extent size was used in both experiments.

Fig. 9 shows the results of Bonnie++ benchmark. The figure depicts the write and the read throughputs of Bonnie++ under different configurations. Each group of

[6] Another overhead is additional memory to manage the block map. With a 512KB extent size, for 1TB storage, the memory overhead is 48MB.

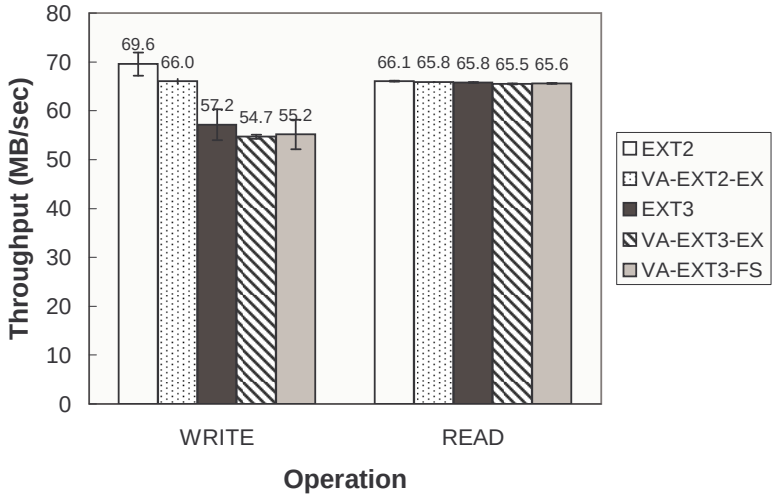


Fig. 9. Impact of VA Metadata Hardening on Performance of VA for the Large-file Workload (Bonnie++)

bars compares the performance among different configurations. Each group consists of five bars: for reference, the first bar and the third bar show the performance of EXT2 and EXT3 respectively. In system configurations, **-EX* represents VA with the extent-based hardening and **-FS* denotes VA with the file system-based hardening.

The Bonnie++ results show that VA-EXT2 with the extent-based hardening incurred an overhead of 5.1% in write operations and incurred no measurable overhead in read operations compared to EXT2. Similarly, VA-EXT3 with the extent-based hardening incurred an overhead of 4.3% in write operations and incurred no measurable overhead in read operations compared to EXT3. VA-EXT3 with the file system-based hardening caused 3.5% overhead compared to EXT3. The impact of the extent-based hardening was smaller in VA-EXT3 compared to VA-EXT2, which can be explained as frequent journal accesses amortized the seek operation cost of VA metadata hardening because of the journal's proximity to VA metadata on the disk.

Fig. 10 shows Postmark results. In Fig. 10, we observed that VA-EXT2 with the extent-based hardening showed a 2.2% performance increase in the transaction rate

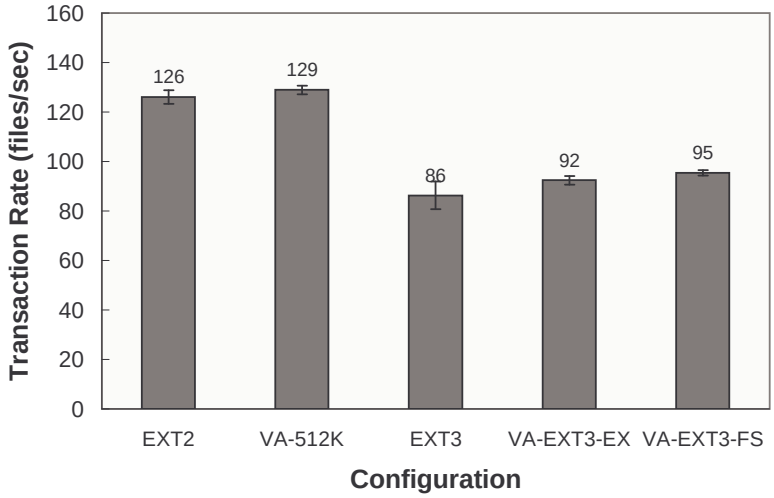


Fig. 10. Impact of VA Metadata Hardening on Performance of VA for the Small-file Workload (Postmark)

and VA-EXT3 with the extent-based hardening showed a 7.1% performance increase. VA-EXT3 with the file system-based hardening further increased performance up to 10.5%. The hardening overhead seems to be more than compensated by other factors such as improved seek times due to sequential allocation. We will present more data later to clarify this further.

These results indicate that the impact of VA metadata hardening is larger in the large-file workload and when the VA is used with the ext2 file system (VA-EXT2). We used, by default, VA-EXT2 with the extent-based hardening in all the following experiments to provide the base performance. When a comparison is needed, we also present the results of VA-EXT3 with both hardening policies.

b. Extent Size

We measured the impact of the extent size using Bonnie++ and Postmark. Four different extent sizes were used in this experiment, i.e., 128KB, 256KB, 512KB, and 1MB. Bonnie++ and Postmark were configured to have the same configuration as

the previous experiment. We compared each performance result of VA-EXT2 to that of EXT2.

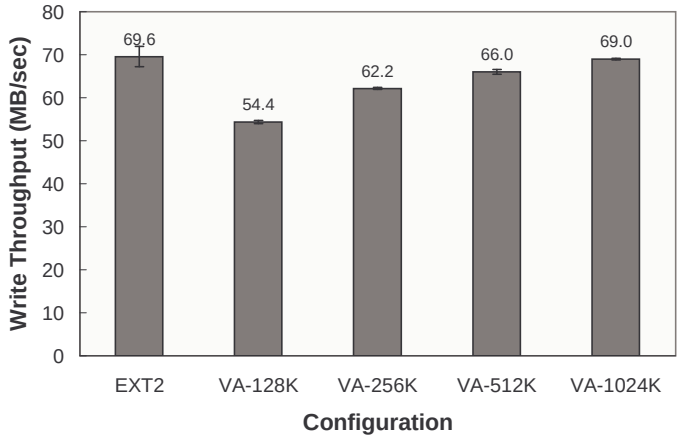


Fig. 11. Impact of Extent Sizes on Performance of VA for the Large-file Workload (Bonnie++)

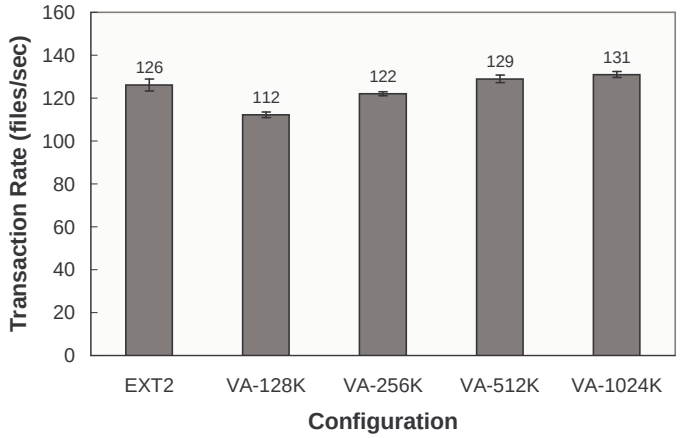


Fig. 12. Impact of Extent Sizes on Performance of VA for the Small-file Workload (Postmark)

Fig. 11 and Fig. 12 show the impact of the extent size in both workloads. Each bar in Fig. 11 shows the write throughput of Bonnie++[7] and each bar in

[7]The read throughput of each configuration was nearly the same, which was not shown due to space constraints.

Fig. 12 denotes the transaction rate of Postmark. In both Bonnie++ and Postmark results, larger extent sizes show better performance. Larger extents increase spatial locality and reduce VA metadata overhead (both number of hardening operations and the size of the hash table). Bonnie++ incurred overheads even at extent sizes of 1MB whereas in Postmark, extent sizes of 512KB or larger incurred no overhead or performed better than EXT2. The nature of I/Os, large (Bonnie++) versus small (Postmark) contributed to these differences.[8]

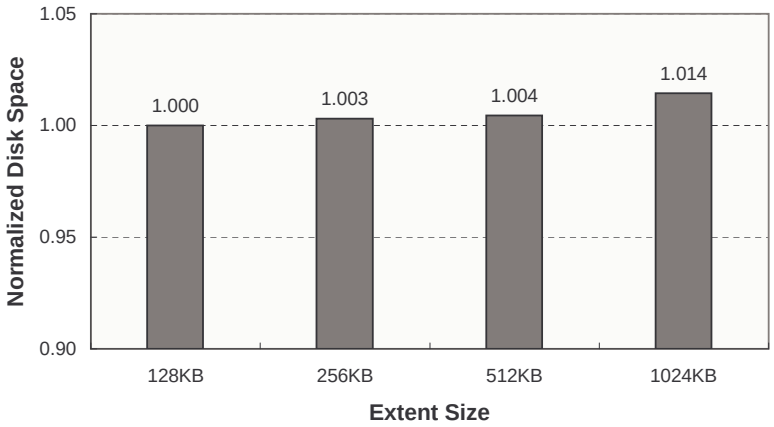


Fig. 13. Normalized Disk Space to the Occupied Disk Space of the 128KB Extent for Each Postmark Experiment

Large extent size can increase overall performance of VA-EXT2, but it may increase occupied disk space due to fragmentation. We measured the occupied disk space for each previous Postmark experiment and compared it for different extent sizes. Fig. 13 shows these results. In the case of the 512KB extent size, only 0.4% disk space overhead was observed compared to that of the 128KB extent. These results indicate that we can use extents up to 1MB without much disk space overhead. All the following VA experiments in this study used a 512KB extent unless specifically

[8]We also ran both benchmarks with VA-EXT3 and got results similar to VA-EXT2. We didn't report this result due to the similarity.

noted.

c. Reclaiming Allocated Storage Space

We studied the impact of deleted files in two scenarios, periodic purges and the usual file system activity of file creation and deletion. For periodic purges, we used Bonnie++ to create a large test file (of 2GB size) and delete it, which is followed by the reclaim operation. For the usual file system activity, we used Postmark with the same configuration as the previous experiment except for one parameter; we changed the create/delete ratio from 5 (default) to 3. In this configuration, the file creation happens with 30% probability and the file deletion happens with a probability of 70%. So, during transactions, the amount of data that the file system contains is reduced. After transactions of Postmark, we reclaimed allocated space of deleted files and created an additional 10,000 files three times to see the impact on the file system block space. In each experiment, we ran the benchmark for EXT2 and VA-EXT2 and recorded block allocation statistics, e.g., the number of allocated blocks in EXT2, the number of allocated extents in VA-EXT2.

Table IV. Reclaim Operation of Periodic Purges

Operation	**EXT2 Live Blocks**	**VA Live Extents (Number of Blocks)**
1. Initial	0	0
2. FS create	27,509	217 (27,776)
3. File write to disk	552,311	4316 (552,448)
4. File delete	27,509	4316 (552,448)
5. Reclaim	27,509	217 (27,776)

Table IV shows the result of the Bonnie++ experiment. In this experiment, a *live block* denotes a data block which contains valid data and a *live extent* denotes an extent which contains at least one live block. The table represents the number of live blocks of EXT2 and the number of live extents of VA-EXT2 during the experiment. In the case of live extents in VA-EXT2, the corresponding number of blocks are also shown in parenthesis for comparison with that of EXT2. The second row of Table IV shows the statistics after the file system is created (§2). The third row shows that as a result of test file creation (§3), EXT2 allocated 552,311 blocks whereas VA-EXT2 allocated 4,316 extents (552,448 blocks). The number of allocated blocks in VA-EXT2 was a little bit higher than that of EXT2 because the extent size (512KB) was larger than the file system block size (4KB). The fourth row shows the statistics after the test file is deleted before the reclaim operation (§4). The last row shows the statistics after the reclaim operation (§5); the number of live extents of VA-EXT2 shown corresponds to the value after the reclaim operation. In this experiment, 100% of the allocated extents could be reclaimed.

Table V shows the results of the Postmark experiment. In this experiment, a *touched block* denotes a data block which is allocated by EXT2 at least once. The table depicts the number of live blocks and touched blocks of EXT2 and the number of live extents of VA-EXT2 during the experiment. Similar to the results of the previous experiment, the second and the third rows of Table V show the number of live blocks, touched blocks and live extents after the file system is created (§2), and after the file set for transactions is written to disk (§3). At the fourth row of Table V, which shows the statistics after transactions (§4), we can observe that the number of touched blocks increased. The increased number of touched blocks corresponds to newly allocated (not reused) blocks by EXT2 due to the file creation during the transactions. VA-EXT2 also showed an increase of the allocated extents by a similar

Table V. Reclaim Operation of Usual File System Activity

Operation	**EXT2 Live Blocks**	**EXT2 Touched Blocks**	**VA-EXT2 Live Extents (Number of Blocks)**
1. Initial	0	0	0
2. FS create	27,509	27,530	217 (27,776)
3. File set write to disk	516,020	516,020	4,059 (519,552)
4. Do transactions	163,191	537,982	4,237 (542,336)
5. Reclaim	163,191	537,982	3,623 (463,744)
6. 10k files write to disk	260,494	537,982	3,713 (475,264)
7. 10k files write to disk	359,303	537,982	3,917 (501,376)
8. 10k files write to disk	457,126	539,926	4,176 (534,528)

amount.

The fifth row of Table V shows the resulting number of live extents in VA-EXT2 after the reclaim operation (§5). In contrast to the result of the previous experiment, VA-EXT2 could reclaim only 15% of total allocated extents. Transactions in Postmark caused *extent fragmentations*, so that some extents could not be freed, remaining partially live. The following three rows of Table V show the statistics of EXT2 and VA-EXT2 when we created an additional 10,000 files three times (§6, §7, §8). When we created 10,000 files (§6), EXT2 allocated 97,303 blocks while not increasing the number of touched blocks, which means that EXT2 efficiently reused previously allocated blocks. VA-EXT2 allocated 90 new extents (11,520 blocks) and reused partially live extents for other blocks. These results indicate that the file system efficiently reuses blocks so that partially live extents will eventually be filled because reused blocks reside at the extents already allocated. The following two rows of Table V show the results as more files are created.

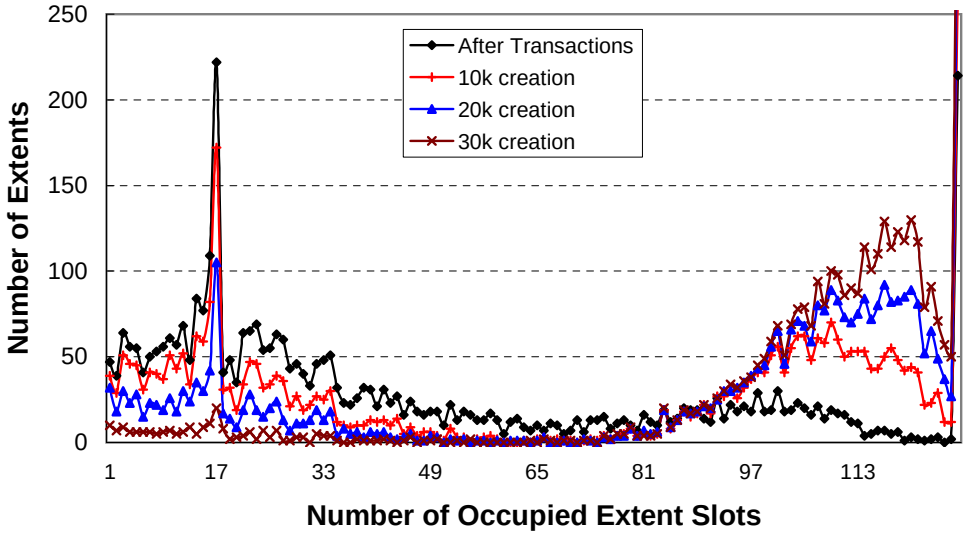


Fig. 14. The Filled Status of Extents

Fig. 14 presents the extent statistics after doing transactions (§4) to after cre-

ating total 30,000 files (§8). The x-axis represents the filled status of the extent, and the y-axis depicts the corresponding number of the extents. We can observe that as more files are created after the reclaim operation, partially live extents get more and more filled. Reclaiming allocated space of deleted files in the small-file workload may require sub-extent valid bits at the VA layer. Large files, when deleted, can be very efficiently reclaimed so that reclaimed space can be reused for allocation at the VA layer.

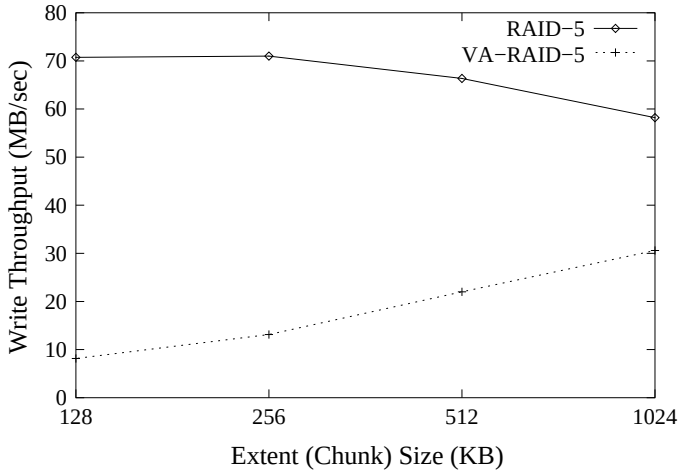


Fig. 15. Performance of VA-RAID-5 of the Large-file Workload (Bonnie++)

d. RAID

We measured the performance of VA with RAID using the same benchmarks and configurations as the extent size experiment. Fig. 15 shows the write throughput of RAID-5 and VA-RAID-5 systems.[9] As the extent size increases, the write throughput of VA-RAID-5 increases, but even with the 1MB extent size, the write throughput corresponds to only 52.5% of RAID-5. This overhead is due to the expensive small-

[9]RAID-5 and VA-RAID-5 systems showed nearly the same read throughput.

write cost of VA metadata hardening.

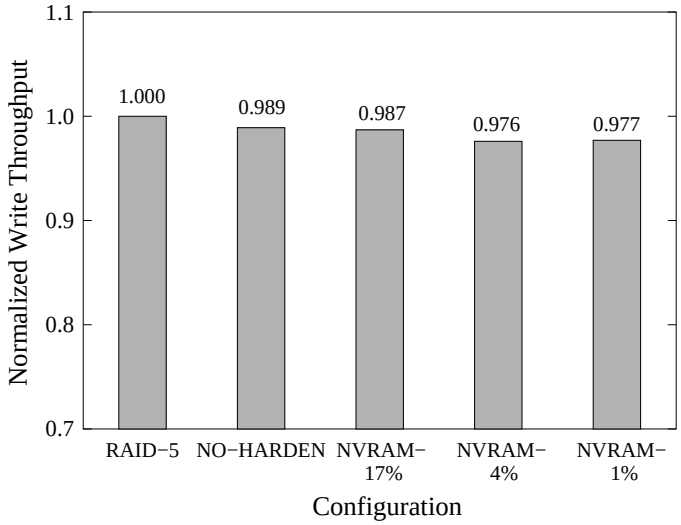


Fig. 16. Impact of NVRAM

To reduce the overhead, we considered an alternative configuration using non-volatile memory (NVRAM) for this workload; we gather VA metadata changes in NVRAM until it reaches a predefined size, at which point, we commit all VA metadata to disk. This enables aggregation of VA metadata writes to disk and correspondingly reduces the small-write costs of RAID-5. Fig. 16 shows the result when we used NVRAM with a 512KB extent size. The height of bar denotes the write throughput normalized to that of RAID-5. In system configurations, *NO-HARDEN* denotes the configuration where VA metadata hardening is turned off and *NVRAM-** denotes the configuration with NVRAM. The percentage in this configuration is the ratio of the dedicated NVRAM size to total amount of VA metadata. With NVRAM-1%, the write performance only suffers 1.2% penalty compared to the case when hardening is turned off. This result indicates that a very small amount of NVRAM can eliminate almost all VA metadata hardening costs in a VA-RAID-5 system. In all other experiments except this one (e.g., VA-RAID-5 for the large-file workload),

we do not assume that we have NVRAM.

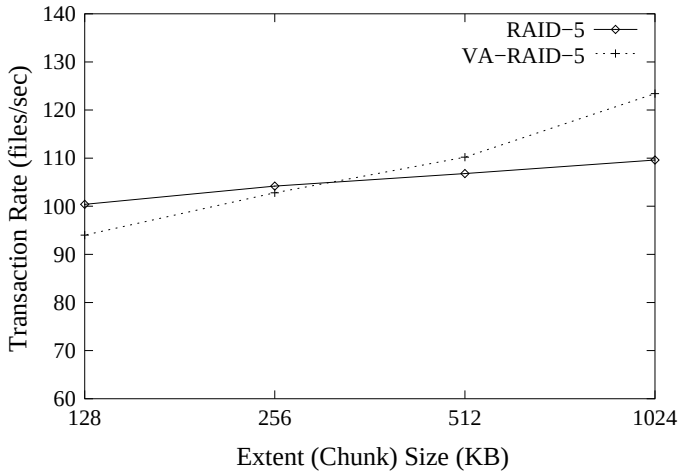


Fig. 17. Performance of VA-RAID-5 of the Small-file Workload (Postmark)

Fig. 17 shows the Postmark transaction rate of RAID-5 and VA-RAID-5 systems for various extent (chunk) sizes. VA-RAID-5 with extent sizes of 256KB or higher performed nearly as well or better than RAID-5. VA-RAID-5 shows better performance with larger extent sizes in both workloads. It is observed that the small files in Postmark impact the performance of the normal RAID-5 system as well as the VA-RAID-5 system.

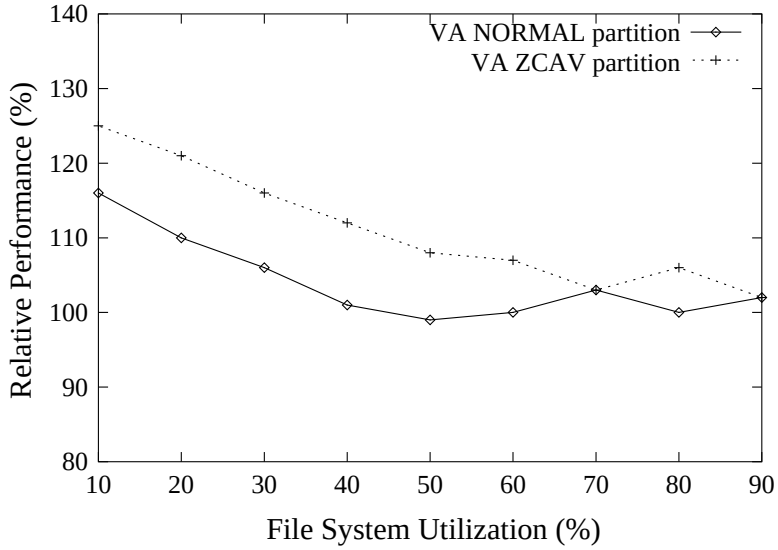
e. Space Allocation Policy

We performed two experiments to measure the impact of various data placement policies on the performance of VA. We used Postmark in these experiments. Postmark was configured to create 3,500,000 files and perform 100,000 transactions in 200 directories. First, we measured the transaction rate of two configurations: VA on a 16GB partition of the 147GB disk (*VA NORMAL partition*) and VA on a 16GB partition of the 37GB disk (*VA ZCAV partition*). The data rate difference of the location of

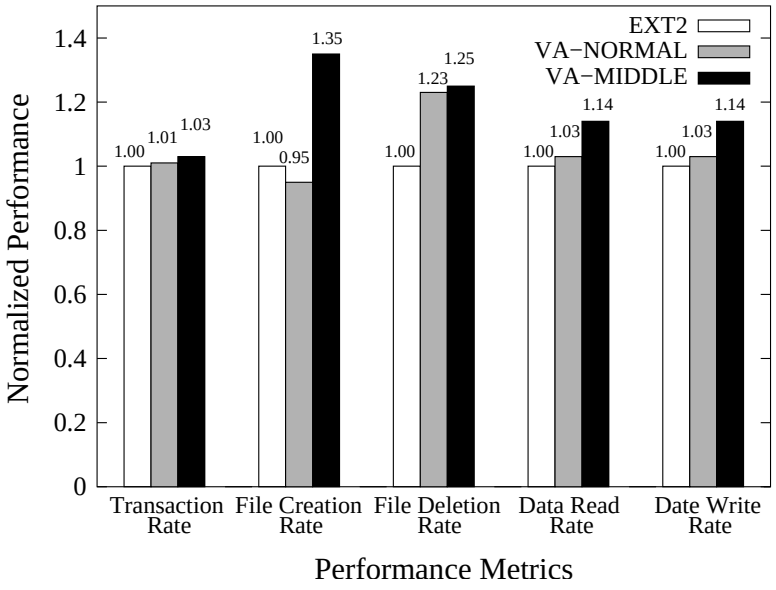
0GB and 16GB of each partition was 2.4% and 14.0% on the two disks (due to ZCAV effects).

The graph in Fig. 18(a) shows the performance of each configuration as a percentage of the transaction rate of the corresponding EXT2. VA on a ZCAV partition performed better than EXT2 by 25% at 10% file system utilization, whereas VA on a NORMAL partition showed 16% performance improvement. As the file system utilization increases, the performance improvement decreases for both cases as the seek time benefits from clustering decrease. However, VA on a ZCAV partition always showed better performance improvement than VA on a NORMAL partition due to higher data rates available for metadata on ZCAV partition.

The second experiment was done in the NORMAL partition which has almost the same data rate across the partition, allowing us to focus on the impact of seek times. We did experiments with two data placement policies. One policy (VA-NORMAL) starts allocation from the beginning of the partition and the other one (VA-MIDDLE) starts allocation from the middle of the partition as shown in Fig. 7. The latter policy will place the metadata cluster (which consists of VA and FS metadata) in the middle of the partition because FS metadata is created at the time of file system creation. Fig. 18(b) shows various performance metrics of the two policies. As shown in Fig. 18(b), VA-MIDDLE improved the transaction rate by 2%, the file creation rate by 40%, the file deletion rate by 2%, the data read rate by 11% and the data write rate by 11% compared to VA-NORMAL. These performance improvements of VA-MIDDLE are attributed to the fact that average seek distance to metadata is reduced compared to VA-NORMAL.



(a) Space Allocation Policy 1



(b) Space Allocation Policy 2

Fig. 18. Impact of Space Allocation Policies

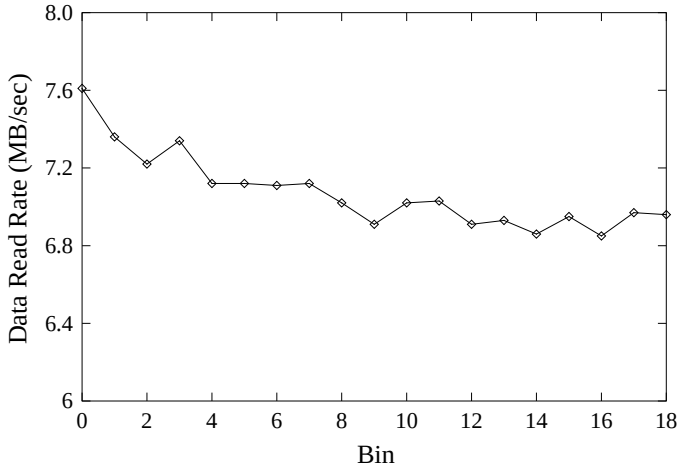


Fig. 19. Impact of Metadata and Data Separation

f. Metadata and Data Separation

In order to study the impact of separating metadata and data into two separate clusters, we modified Postmark to report on the data read rate of files based on their proximity to metadata. We used a 32GB partition and populated files up to 75% file system utilization. Files in the system are divided into 19 bins, where *bin 0* is the closest to metadata and *bin 18* is the farthest away.[10] The data read rate for files in each bin is reported in Fig. 19. *Bin 18* shows about 9% degradation of the data read rate compared to that of *bin 0*. If we exclude the bias from ZCAV effects, the degradation from metadata and data separation will be 4.6%. This is attributed to the fact that the distance from metadata to *bin 18* is farther than to *bin 0*.

[10]The (VA and FS) metadata cluster is located before bin 0.

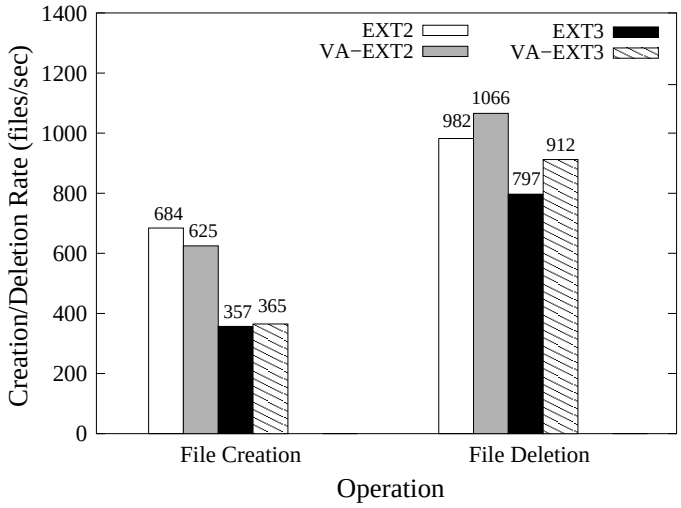


Fig. 20. Impact of Metadata Clustering

g. Metadata Clustering

We configured Postmark to measure the speeds of file system create and delete operations. Postmark was set to have a file system utilization of 30% in a 7GB partition. Fig. 20 shows the results of this experiment. VA-EXT2 with the extent-based hardening incurred an overhead of 8.6% in the file creation phase and showed 8.6% improvement in the file deletion phase. VA-EXT3 with the extent-based hardening showed 2.2% performance increase in the file creation phase and 14.4% improvement in the file deletion phase. VA-EXT3 with the file system-based hardening showed 16% performance increase in the file creation phase and 19.4% improvement in the file deletion phase. In VA-EXT2, VA metadata hardening and seek distances affect the performance of file create operations. In the file deletion phase, the seek operations to access metadata determine the performance. VA-EXT2 showed faster deletion performance than EXT2 due to metadata clustering. In VA-EXT3, the journal access improved the performance by reducing the seek times to metadata.

h. File System Utilization

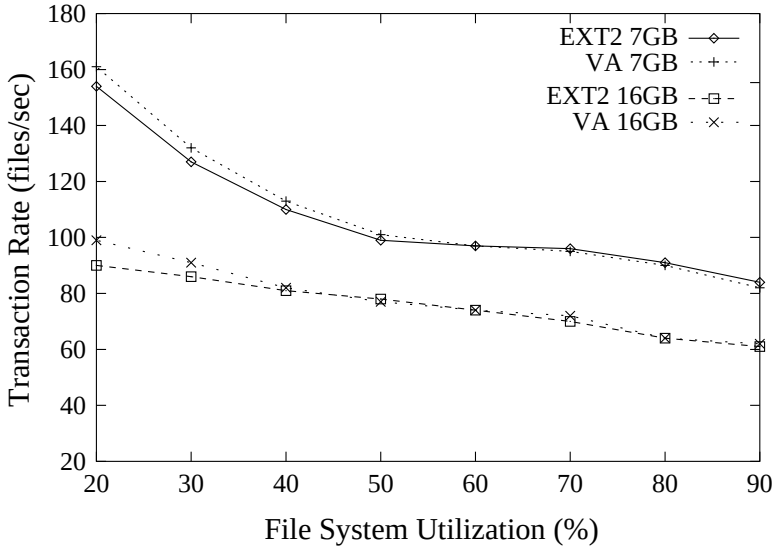
We measured the performance of VA according to various partition sizes and various file system utilizations. We used Postmark to analyze the performance of normal file system activities. Two different partition sizes were used: 7GB and 16GB. For each partition size, we configured Postmark to have a file system utilization from 20% to 90% in units of 10%. For each partition size and each file system utilization, we measured the transaction rate of VA-EXT2 and compared it to that of EXT2 with the same configuration.

Fig. 21(a) shows the results of Postmark. A large partition with smaller utilization works best in VA-EXT2 because data is relatively more clustered than EXT2. VA-EXT2 showed about 7.6% performance improvement (excluding the ZCAV bias) with a 16GB partition at 20% utilization compared to EXT2. As the file system gets full, the on-disk layout of the two systems becomes similar. Therefore, the two systems show similar performance at larger utilizations. The performance results under the partition size of 7GB show similar trends.

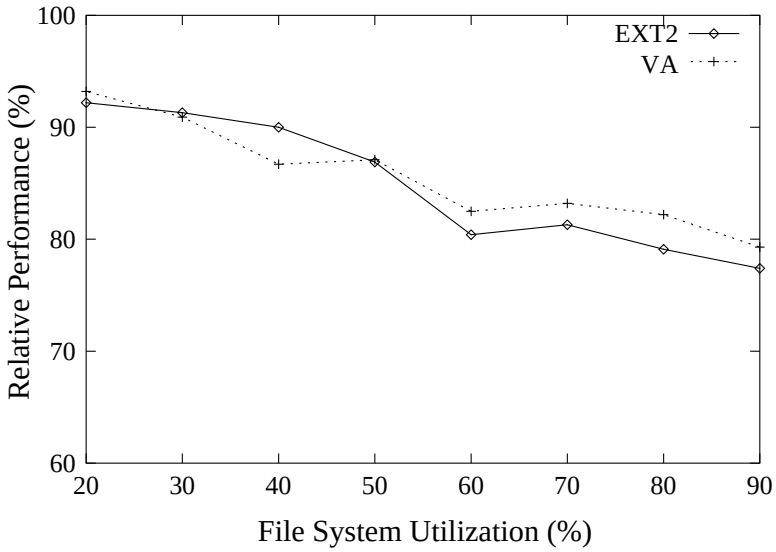
i. File System Aging

In this experiment, we measured the impact of fragmentation of the file system due to aging on VA-EXT2 and compared it to EXT2 in order to examine the long term effectiveness of our system. We modified Postmark to simulate file system activities over time similar to [27].

First, it creates a number of directories and populates them with files up to 10% file system utilization. It measures the transaction rate the same way as is done in Postmark. Then it populates more files to make a file system usage of 20% and measures the transaction rate again. It repeats these procedures up to 90% utilization.



(a) File System Utilization



(b) File System Aging

Fig. 21. Impacts of File System Utilization and Aging

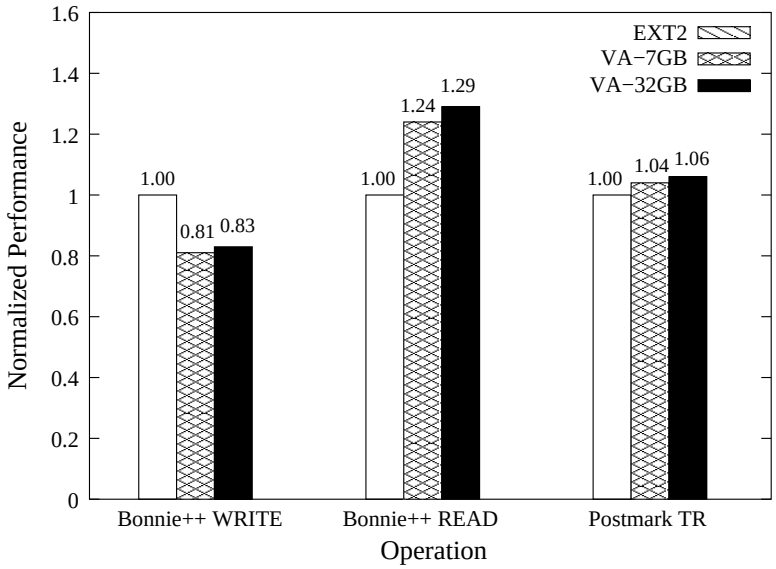
The graph in Fig. 21(b) shows the performance of the file system at each utilization as a percentage of the transaction rate of the corresponding empty file system. As shown in Fig. 21(b), as the file system ages, VA-EXT2 incurs a performance impact similar to that of EXT2 at different file system utilizations.

j. Multiple File Systems

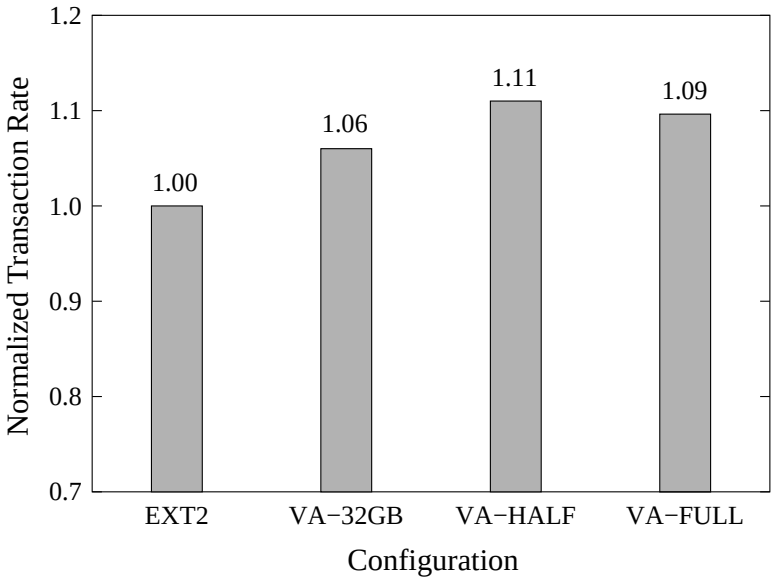
Next, we considered the use of multiple file systems with virtual allocation. Specifically, We compared the fixed partition approach with our approach by performing two experiments. In the first experiment, we used two configurations to see the impacts of disk arm movements by employing two benchmarks (Bonnie++ and Postmark). First, we created two 3.5GB partitions on the 147GB disk and created an ext2 file system[11] on each partition (*VA-7GB*). Both benchmarks were configured to have a file system utilization of 30%. In the second configuration, we used two 16GB partitions on the 37GB disk with ext2 file systems (*VA-32GB*), where both benchmarks were set to have 80% utilization. We ran the benchmark on two ext2 file systems concurrently and compared the results between VA and the fixed partition approach.

Fig. 22(a) shows the average throughput of two file systems from Bonnie++ benchmark and the average transaction rate of two file systems for Postmark. In the large-file write workload, we observed that VA-7GB incurred an overhead of 19% and VA-32GB caused an overhead of 17% compared to the fixed partition approach. The VA metadata hardening is the main reason for this overhead as observed earlier. The read operation of VA-7GB showed large performance improvement, i.e., by 24%, and VA-32GB increased performance up to 29%. These performance gains are attributed

[11]We performed the same experiments using an ext3 file system with various journaling modes and got similar results to those of EXT2, which were not shown due to similarity.



(a) Multiple File Systems 1



(b) Multiple File Systems 2

Fig. 22. Performance of Multiple File Systems with VA

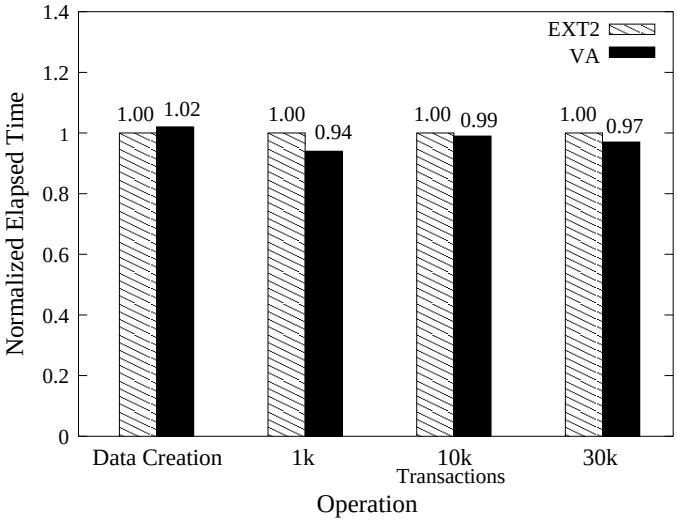


Fig. 23. Performance of VA for Database Workload

to the fact that if multiple file systems access a single storage device concurrently, the fixed partition approach will incur higher disk seek costs between the two partitions, whereas VA can reduce these seek costs significantly due to its allocate-on-write policy.

In the small-file workload, VA-7GB and VA-32GB improved performance by 3.6% and by 6.1%. These performance improvements are due to the reduction of seek distances similar to the case of the large-file workload. These results indicate that VA could provide performance improvement over fixed allocation for these workloads while increasing the flexibility of allocation across multiple file systems. In both workloads, VA in heavily utilized disk (VA-32GB) showed better performance improvements than in the other VA configuration (VA-7GB). The higher media rates possible due to ZCAV effects and the placement of metadata in those regions contributed to this difference.

In the second experiment, we modified the configuration of VA-32GB into two other configurations to study the impacts on performance when two file systems are not created at the same time. In the first configuration, the second file system is

created after 40% of the first file system is written. In the second configuration, the second file system is created after 80% of the first file system is written. We call these *VA-HALF* and *VA-FULL*. These configurations are designed to force metadata separation of the two file systems. We measured the transaction rate of two file systems the same way as is done in VA-32GB.

Fig. 22(b) shows these results. VA-HALF showed better performance than EXT2 by 10.7% and better performance than VA-32GB (where two file systems were created at the same time) by 4.6%. In VA-HALF, the impact of VA metadata hardening is reduced because the first file system already allocated 40% extents. These results indicate that virtual allocation can get similar or better benefit even if multiple file systems are not created at the same time. The performance result of VA-FULL shows a similar trend.

k. Results for the Other Workloads

Fig. 23 shows the results of TPC-C workloads. The bar height represents the elapsed time normalized to that of EXT2 for various database operations. VA-EXT2 showed performance in different database operations comparable to EXT2 because the on-disk layout of the workload was similar. Transactions were done on one large file (a table-space), so most of the data was clustered in both the systems.

Fig. 24 shows the results of NFS trace replay. The bar height represents the request latency of each NFS operation. VA-EXT2 showed similar performance to that of EXT2.

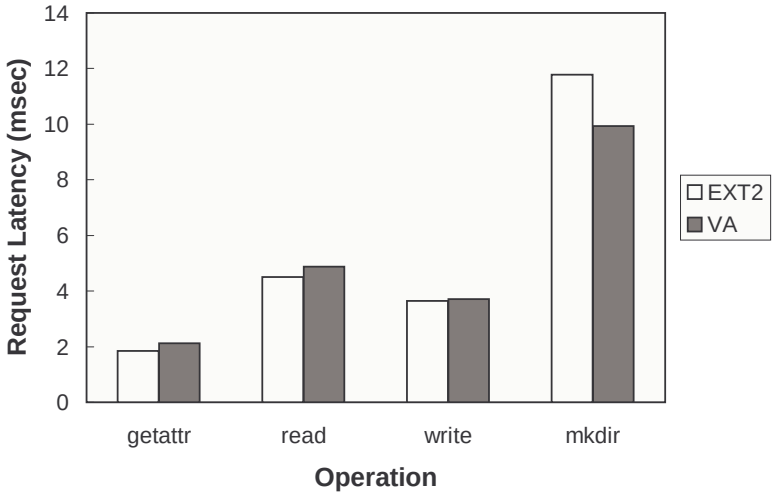


Fig. 24. Performance of VA for NFS Trace Replay

D. Related Work

The related work on VA can be grouped into four categories. The first group proposes file system-level changes, requiring that file systems be modified or replaced to improve storage allocation flexibilities. The second group proposes a block-level approach, the category under which VA belongs. The third group proposes changes to the storage interface, i.e., the interface between the file system and the storage system. Finally, the fourth group discusses the work in other system environments (e.g., virtual machines).

1. File System-level

IBM's Storage Tank [14] separates metadata operations from data operations to allow a common pool of storage space to be shared across heterogeneous environments. Log-structured file systems (for example, LFS [17] and WAFL [18]) allow storage allocation to be detached from file systems because all new data is written at the

end of a log. File systems such as ReiserFS [37] and JFS (Journaled File System) [38] support expansion of file systems. None of these systems allocates storage space based on the actual usage.

2. Block-level

Veritas Volume Manager [39] and other existing Storage Resource Management (SRM) products such as IBM Tivoli Storage Manager [40] provide considerable flexibility in storage management, but allocate storage space based on the file system size. Loge [41] separates storage allocation from the file system to optimize write operations; it writes blocks near the current disk-head position. The Loge system does not provide storage allocation flexibility with traditional file systems. In contrast, the focus of this study is on the integration of the flexible storage allocation scheme into a traditional file system environment. HP's AutoRAID employed dynamic allocation in order to reduce the small write penalty of RAID systems [42]. Petal [43] proposes a disk-like interface which provides an easy-to-manage distributed storage system to many clients, (e.g., file systems and databases). Petal's virtual disks cleanly separate a client's view of storage from the physical resources, which allows sharing of the physical resources more flexibly among many clients, but they still allocate storage space according to the configured file system size. Recently a storage resource allocation technique called *thin provisioning* has been introduced [44], which provides storage allocation flexibility as our system does. However, the details about their architecture, implementation and performance are unknown.

3. More Expressive Interfaces (Between the File System and the Storage System)

Object-based storage proposed by CMU [15] allows storage allocation to be done at the storage systems as opposed to a file system manager. Logical disk [45] defined a new interface to disk storage that separates file management and disk management. Boxwood [16] exposed allocate/deallocate interface to the higher layer through the chunk manager to allow flexible storage allocations. These approaches require changes of the existing storage interface (in Boxwood), requiring the new storage devices (in object-based storage) or modified file systems (in Logical disk).

4. Other System Environments

VMWare's virtual machine [46] can operate in a mode where it uses a host file system file as a virtual disk; in this mode, the host file is expanded dynamically when needed (if the host OS supports sparse files, which is the usual case). A number of recent studies have considered building federated file and storage systems [47, 48, 49] across multiple workstations. These systems focus on the issues of trust and availability among others, while allowing storage resources to be shared across multiple systems.

Virtual allocation is *on-demand* allocation; it is orthogonal to storage virtualization products sold by vendors. Storage virtualization hides the connectivity, physical characteristics, and organization of devices from file systems. However, existing products allocate storage based on file systems' size, and don't allow sharing of unused space. This is akin to programs allocating the maximum amount of memory they may need and not relinquishing this or sharing it even when they actually need much smaller amount of memory. It is because the products employ static allocation where every mapping between logical and physical block is predetermined for given size

when creating a logical volume. In contrast, in VA, every mapping is determined dynamically as data is written (on-demand) so that storage can be allocated based on actual use. Using VA, several different file systems may share single storage resource or a single file system may span multiple storage resources, resulting in better flexibility.

E. Discussion

In virtual allocation, we identified the need for reclaiming the deleted file space at the storage system. We also found that identification of file system metadata could lead to efficiencies in hardening virtual allocation metadata through such policies as file system based-hardening. We employed *gray-box* approach proposed by Wisconsin [19, 20] for the file system-based hardening and the reclaim operation. As an alternative, we could use a stackable file system [50]. We can identify data types (e.g., metadata or data) and detect file system delete operations by adding functionalities to the stackable file system layer. The stackable file system approach is less constrained by various platforms or file systems than the (gray-box) approach employed here. We also implemented this approach and confirmed it is feasible.

F. Summary

In this chapter, we have proposed virtual allocation employing an allocate-on-write policy for improving the flexibility of managing storage across multiple file systems/platforms. By separating the storage allocation from the file system creation, common storage space can be pooled across different file systems and flexibly managed to meet the needs of different file systems. Virtual allocation also makes it possible for storage devices to expand easily so that existing devices incorporate other avail-

able resources. We have shown that this scheme can be implemented with existing file systems without significant changes to the operating systems. Our experimental results from a Linux PC-based prototype system demonstrated the effectiveness of our approach.

CHAPTER III

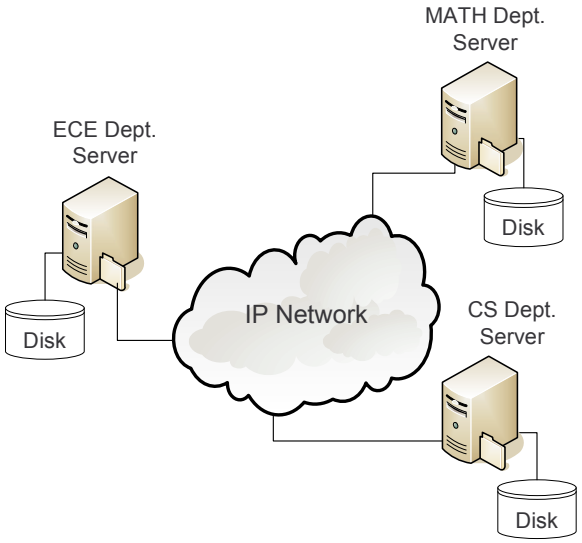
DATA DISTRIBUTION IN NETWORKED STORAGE SYSTEMS

Storage "bricks" are being used to build cost-effective storage systems that can scale over a broad range of system sizes. A storage brick contains a microprocessor, a small number of disk drives and network communications hardware. Such an approach has been advocated by many researchers [4, 51, 52]. Some of the production systems have taken a similar approach [53].

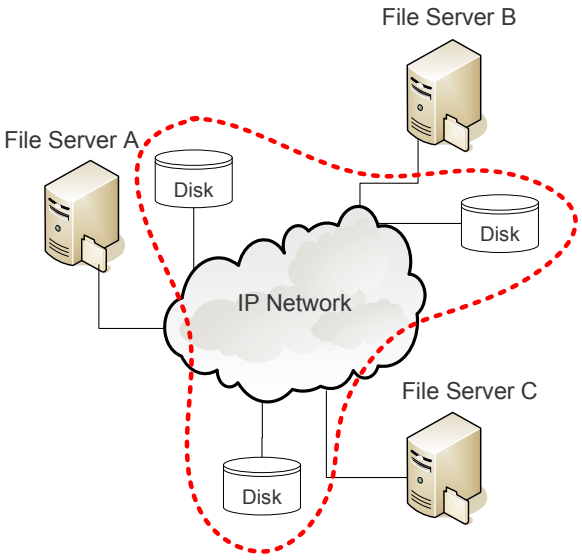
Network attachment of storage systems has also resulted in the possibility of consolidating or pooling of storage systems over wider geographic areas, for example storage systems connected over Local/Metropolitan/Wide Area Networks (LAN/MAN/WAN). It is expected that such network connectivity of storage systems will (a) enable wider (geographically) distribution and access of storage and (b) enable new storage paradigms (such as "storage as a service" [3]).

It is possible to build a large-scale storage system based on the infrastructure mentioned above. For example, grid computing applications such as TerraGrid Cluster File System [54] or IBM General Parallel File System (GPFS) [55] can employ multiple storage bricks to provide high performance and scalability. These storage bricks can be attached to clusters over LAN or over WAN [56].

Fig. 25(a) shows the example usage of this infrastructure in a University campus. There are three Department file servers each of which has its "local" (which is attached directly to the server) disk. Each disk is also attached to network so that each server can access other server's disk if it is necessary. During normal times, each server uses its local disk, but there may be cases when one server needs more storage resources than it has. Such a case can occur, for example, when CS Department server temporarily needs lots of storage space due to large-scale computing. Rather than



(a) Cooperative Storage Consolidation



(b) Storage Hosting Service

Fig. 25. Application Examples of Networked Storage Systems

buying and attaching more physical disks to CS server, it would be better if we can use other server's available storage and network bandwidth. This application scenario of cooperative storage consolidation can be generalized to storage hosting service as shown in Fig. 25(b). In Fig. 25(b), storage devices are geographically distributed and attached to the IP network. Any file server can lease available storage resource as it needs. In this configuration, the increased flexibility of storage allocation poses challenges to data distribution among disks because data distribution of each server over the network impacts its performance directly.

In a large-scale storage system, it is important to (a) allocate data efficiently because the cost of data reallocation is high due to the large scale and large network latencies and (b) redistribute data adaptively in an automated manner according to configuration changes or workload changes (after initial allocation) to improve performance.

The problem described in (a) is referred to in the literature as the file allocation problem (FAP). There is a significant body of research on this problem [57], but it is not our focus. In this study, we focus on the problem of *data redistribution* (described in (b)) in storage systems built out of networked devices. This study focuses on data migration as a redistribution action. An area of ongoing research, automated data migration requires deciding *when* to migrate, *where* to migrate, *what* data-set to migrate, and *how* to migrate (migration speed) [58]. In this study, we consider these migration decision issues in terms of balancing locality and load balance as explained below.

When data is "local" or proximate to the user[1] over the network, user can observe lower network latencies and hence can observe better performance. However, if the

[1]In this study, *user* is file system or application of storage devices.

user's data sets are larger than the capacity of the local devices, a tendency to keep data local may cause thrashing problems. Second, if the user's data set receives a significant number of I/O requests, the local devices may not be able to support the required I/O rates or data rates. On the other hand, if user's data is distributed widely over such a storage network, we may be able to exploit multiple devices to support the required I/O and data rates. Wider distribution of data also may result in better load balancing. However, accesses over the network to "remote" devices may incur extra network latency overheads and may tend to be slower than local accesses. Hence, there is a need to balance the locality of data accesses with load balancing in such a system. In this study, we consider *longer-term locality* considerations at the I/O or storage system beyond what is already exploited in memory. Data locality and load balancing issues are studied earlier in memory systems, for example [59].

Multiple users, diverse workloads, and time varying I/O demands make this problem even more challenging. When a user accesses data from multiple locations on the network, data that was local during a previous access may become remote at a later time.

In networked storage systems, we are considering data migration between devices of similar characteristics (disks). The performance difference arises due to network latencies and differences in load at the disks. When devices have equal response times, a local disk will provide better performance than a remote disk, due to network latencies. However, the response times of devices may fluctuate over time due to differences in loads and workloads. A local disk may get slower than a remote disk due to the increased load as more and more data is migrated to a local disk. The migration of data has to take these issues into account. In networked storage systems, the direction of migration of heavily referenced data may not always be in a fixed direction, from a remote device to a local device. When multiple users are considered

from multiple vantage points of the network, the data may have to migrate in different directions for different users at the same time.

Even when a single user is considered, migrating data to a local disk may not always be beneficial even when thrashing (due to large data sets) is not a problem. The local device may not be able to support the required I/O rates or data rates. It may be better to keep some data remote in order to exploit multiple devices to achieve higher I/O and data rates. Such considerations become more difficult when multiple users access storage from multiple vantage points of the network.

We propose *user-optimal* migration to address these issues. In user-optimal migration, each user or application makes migration decisions based on what is best for its application. This is contrasted with a centralized network-optimal migration of data. We use I/O request response time as a measure of performance to decide on the suitability of a device. We explain user-optimal migration with a single user and two devices (A and B), while the approach is more general. If device A is local to the user, initially the user may find A offers better performance than B. Data is then migrated to A. As more and more data is migrated to A, the load on device A may get so high that its performance may get worse than the performance of B. At that point, the migration of data to A will stop. If the response time at A continues to remain high and is found to be worse than B, eventually some active data may be migrated to B to reduce the load on A in order to reduce the response times at A. This process results in a stable operating point when the response times to both devices A and B are nearly the same (including the effects of load and network latency). Locality is exploited as more data migrates to A, but load balancing will also be considered to make sure that disk A doesn't get too heavily loaded as data is migrated to A.

When multiple users share storage devices over the network, each user employs user-optimal migration of his/her data sets to improve the performance or access

times to his/her data. Data migration tries to balance locality and balance load of each user's data accesses in such an environment. Moreover, as the load from different users fluctuates over time, user-optimal migration is expected to smooth out the loads at storage devices.

Our work makes the following contributions and differs from the previous work in the following ways:

- Propose (and evaluate) user-optimal migration policy to guide the migration process.
- Choose *active data* (that is currently read or written) for migration to reduce migration cost.
- Consider the possibility of migrating active data to remote devices in order to improve load balancing.
- Use longer-term performance metrics to guide migration decisions.
- Consider data locality on networked devices.

The remainder of this chapter is organized as follows. Section 2 gives details of the related work. Section 3 describes the algorithm of the user-optimal migration, followed by our prototype implementation and evaluation in Section 4. Section 5 points to discussions and future work. Section 6 concludes the chapter.

A. User-Optimal Migration

In this section, we first describe data placement and next describe user-optimal migration policy. We present important design issues related to user-optimal migration.

1. Data Placement

In large-scale storage systems, storage system's software may employ virtualization in order to allow data to be placed where they are needed. Such virtualization breaks direct coupling between physical and logical addresses and enables mapping of one logical set of data to one or many physical locations. Commercial products such as IBM Collective Intelligent Bricks (CIB) support this feature [51].

In such systems, an indirection map, containing mappings of physical to logical addresses, is maintained. Thus, every read and write request is processed after consulting the indirection map and determining the actual physical location of data. Similar structures have been used by others [23, 41, 42]. In our system, as the data migrates from one device to another device, there is a need to keep track of the remapping of the block addresses. When data is migrated, the indirection map needs to be updated. This is an additional cost of migration. We factor this cost into the design, implementation and evaluation of the proposed scheme.

Migrating data in units of file system blocks (usually 4KB) is easier, but requires a larger amount of information in the indirection map. In order to accommodate these two competing issues of flexibility and the need for smaller indirection maps, we allocate data in 128KB chunks. In order to allow migration of data in multiples of file system block sizes (typically, 4KB), we also maintain a block validity map. For this purpose, our indirection map contains a block validity bit of each file system block and two locations for each data chunk (e.g., part of a chunk may reside at device A and the other part may reside at device B). An entry of an indirection map, in our system, is a 5-tuple as shown below: (*logical-chunk-identifier, A<address, validity-map>, B<address, validity-map>*). We have considered multiple options for the indirection map, but the evaluation of these options is outside the scope of the

current study (primarily due to lack of space). Data caching in shared memory machines have explored similar issues of data structures, flexibility and overheads [60]. It is also noted that the indirection map itself can migrate over the network and can be cached within the memory of individual storage bricks for performance reasons.

Data is migrated to a device that has unused or unallocated storage. A background process merges inactive partial chunks between two devices to keep unallocated storage space available on each device for facilitating migration.

2. Design Issues

User-optimal migration is based on virtualized storage architecture which is explained above. In this subsection, we explain the design rationale of user- optimal migration in terms of migration decision issues.

a. When to Migrate

User-optimal routing has been proposed in [61] for adhoc wireless networks and has been applied in other contexts [62]. While switching paths over which packets are routed does not involve significant overheads, data migration in storage systems involves costs in reading (from old location), transferring (over the network) and writing data (into new location). These costs need to be taken into account in our approach of user-optimal data migration.

We factor these costs by initiating migration only when the performance difference between devices exceeds by some amount, say δ (*migration threshold*). Considering two devices on the network, data present on one device A, is migrated to another device B, when response time $r_A - r_B > \delta$, where r_i includes both the network access times and device response times. When the device is locally-attached

(directly connected to that server), r_i measures data access latency from the local disk (disk seek and rotation time plus data transfer time plus disk queuing time). For a network-attached disk, network latency and network storage protocol processing delay are additionally added. Our approach keeps track of response times of different devices over the network continuously (as explained below) to facilitate the migration. The δ parameter controls the onset of migration in the system. A small δ makes data migration more responsive to differences in response times and a very large δ leaves the data where it is allocated. In this study, we compute the δ parameter based on the observed statistics, i.e., δ =*2 * standard deviation of response times*.

Caching in memory can take advantage of short-term temporal locality of data access patterns. Data migration is intended to take advantage of longer-term temporal locality in data access patterns. Hence, we measure response times over longer periods of time to guide data migration decisions. To minimize the short-term variations while keeping track of longer-term trends in performance, we employed exponential averaging which is widely employed in network measurements, for example in estimating round trip times and queue lengths [63].

b. Where to Migrate

In general, when data can reside on multiple (more than two) devices, data currently on a device with higher response time will be migrated to the device with the lowest response time. It is possible to design several strategies that accomplishes this goal. In this study, we sort devices based on response times and prioritize migration from the device with the highest response time to the device with the lowest response time. If response times to be compared are the same or in similar range, we use randomization to break the tie.

c. What Data to Migrate

In order to reduce the cost of migration, data that is currently read or written to device A is migrated to B. If migrated data is not accessed often, migration may not be helpful in improving performance. Therefore, we need to choose part of data from active data as migration candidates. For this purpose, we maintain *migration candidate list* employing probabilistic LRU algorithm [64]. In this scheme, an accessed item if not already on this list, is entered into the list with a small probability. Hence, the list contains mostly frequently accessed items. Since the list is driven by the LRU policy, not recently accessed items fall off the list over time. As a result, the list mostly contains frequently and recently accessed data. When migration onset condition is met, we consult migration candidate list to check if current accessed data is in the list. Migration occurs only if data is in the list.

As a result, migration involves the steps of transferring the currently accessed data over the network and writing to the second device. This policy of migrating active data, more importantly, results in not causing any additional load at the device with higher response times.[2] We also considered other migration policies such as *migration on read* and *migration on write*. The former policy migrates data only when it is read and the latter one moves data only when it is written. We don't present these results here due to lack of space. The policy presented in this study is *migration on access* where data migration occurs when data is read or written.

[2]Migrating inactive data from heavily loaded device increases load further on that device because migration involves reading data from that device.

d. How to Migrate

When the response times satisfy the constraints mentioned above, active data migration is initiated. The migration rate, rate at which data is migrated, needs to be carefully chosen. High migration rate will impact normal read and write accesses by increasing the load on the network and the devices. High migration rate also may alter the data access patterns quickly and significantly to result in considerable changes to response times at different devices. This may result in oscillations if not carefully managed. Too small a migration rate may not adjust access time imbalances quickly enough to improve performance. We use *migration tokens* to control the speed of migration. The number of migration tokens controls the number of requests that are migrated at any given time. For data migration (read or write), it must get a token first. If it cannot get a token, read and write requests are processed normally; migration doesn't occur.

e. Allocation Policy

Data migration remaps initial allocation of blocks based on performance. Initial allocation influences the load balance and data locality and hence influences data migration. We consider two allocation strategies in our study: striping and sequential allocation. Striping initially allocates data evenly over the devices resulting in balanced load (in most cases). The sequential allocation method allocates data locally until storage space is exhausted on the local device, at which point data is allocated on a remote device. Sequential allocation favors local devices over load balancing.

We study user-optimal migration with both allocation methods. When coupled with striping, hot data would be migrated to local device over time, improving locality and decreasing load balance. When coupled with sequential allocation, hot data

may be migrated to remote device when the local device gets heavily loaded, thus improving load balance over time.

f. Multi-user Environment

User-optimal migration migrates data in a user-selfish manner based on data access latency observed by each user. The migration actions initiated by one user may affect the performance of other users for some time. However, it is expected that the migration actions will tend to improve the performance of all users over longer periods of time. We will study user-optimal migration in a multi-user environment to understand these issues.

B. Evaluation

For evaluation, we implemented the following systems as a Linux kernel driver. Each configuration name consists of <*allocation policy*-*migration policy*>.

- **STR-NOMIG**: Data is striped over available disks. No migration is used.
- **SEQ-NOMIG**: Data is sequentially allocated. No migration is used.
- **STR-MIG**: Data is allocated by striping and migration is performed using user-optimal migration protocol.
- **SEQ-MIG**: Data is allocated sequentially and migration is performed using user-optimal migration policy.

The next subsections present performance evaluation and comparison between each system.

1. Workload

We used SPECsfs benchmark as workload for our study. SPECsfs 3.0 is the latest version of the Standard Performance Evaluation Corp.'s benchmark that measures NFS file server's performance [65]. It is a synthetic benchmark that generates an increasing load of NFS operations against the server and measures the response time and the server throughput as load increases. It defines both the operation mix and scaling rules. The operation mix of SPECsfs 3.0 is mostly small metadata operations and reads, followed by writes. SPECsfs creates files that will later be used for measurement of the system's performance according to its scaling rules and performs transactions following the operation mix.

SPECsfs reports a curve of response time vs. *delivered* throughput (not offered load). The signature of the SPECsfs results' curve contains three key features: the *base response time*, the *slope* of the curve in the primary operating regime, and the *throughput saturation point*. At low throughput, there is a base response time which is the best response time obtainable from the system. As load on the server increases, response time tends to increase (e.g., linear relationship). This slope indicates how well the server responds as load increases. As the load further increases, there will be a throughput saturation point at which a bottleneck in the system limits the throughput. Several factors such as network, the speed of the server and client processor, the size of file cache, and the speed of the server disks determine these features [66].

In this study, we focus on data migration strategy on server disks. Thus, we configure the system to have a bottleneck in server disks to increase the likelihood of data migration. All the following results are for systems running SPECsfs 3.0, NFS version 3 using UDP.

SPECsfs is used for our evaluation because it reflects realistic workloads. It tries

to recreate a typical workload based on characterization of real traces by deriving its operation mix from much observation of production systems [65]. At the same time, using SPECsfs workload is challenging. Its operation phase consists of file creation and transactions, each of which has different characteristics; the file creation phase is mostly write operations while the transaction phase involves random reads and writes. Thus, user-optimal migration must adapt to these abrupt access pattern changes and load changes over time.

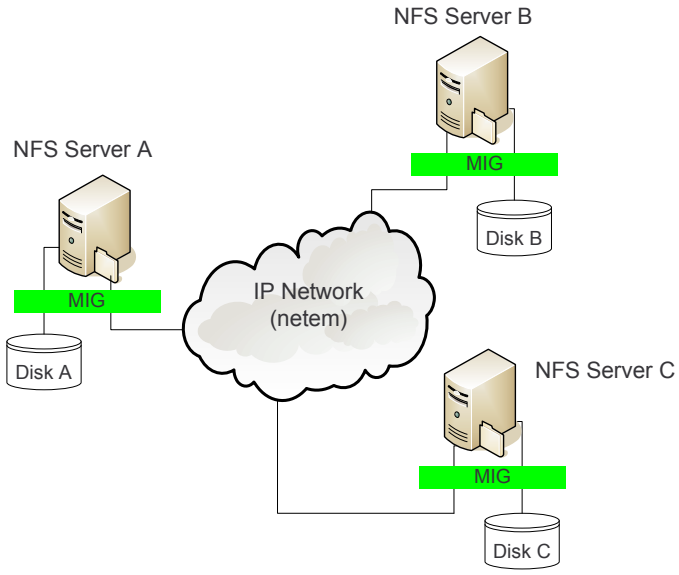


Fig. 26. Experimental Configuration

2. Experimental Setup

Fig. 26 shows our experimental configuration. All of the machines in our experiments consist of a commodity PC system equipped with a 3GHz Pentium 4 processor, 1GB of main memory. Three disks are connected to each NFS server: one locally (directly) attached disk (a 10,000 RPM Seagate SCSI disk through Adaptec SCSI card

Table VI. Three Different Network Latencies Used in This Study

RTT			
avg.	**min.**	**max.**	**mean dev.**
9.0ms	4.3ms	18.5ms	2.3ms
17.0ms	12.4ms	24.3ms	2.2ms
25.0ms	20.5ms	32.2ms	2.2ms

29160) and two network attached disks (same kind of disks connected through iSCSI[3] protocol). We use two clients (load generators) for each NFS server.

The operating system was Red Hat Linux 9 with a 2.4.30 kernel and the exported file system (of each NFS server) was an Ext2 file system. To see the impact of network latency, we used *netem* [67] which provides network emulation functionality for testing protocols by emulating the properties of wide area networks. Netem is included, by default, in Linux kernel 2.6 distribution. Three subsystems are connected through a router with network emulator (netem) in 100Mb/s LAN. We used three different network latencies as shown in Table VI (9ms RTT was used as a default, unless specifically noted). We employed δ =*2 * standard deviation of device response times + 2 * standard deviation of RTT* to control the onset of migration. Inherent variance of netem added reality to the experiments in addition to access pattern changes of SPECsfs workload.

[3]iSCSI stands for Internet SCSI, which enables to carry SCSI commands over IP networks.

3. Single-User Environment

For single-user environment, we used NFS Server A with two load generators. NFS Server A employed Disk A (*local*) and Disk B (*remote*) each of which had a 16GB partition.[4] It exports a 32GB file system which spans Disk A and Disk B to its load generators. First, we evaluate the impact of the design parameters in user-optimal migration and then consider more complex scenarios.

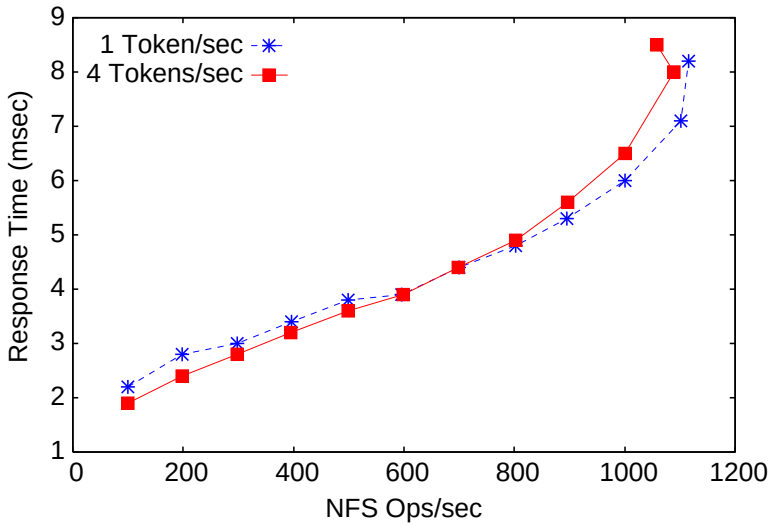


Fig. 27. Impact of Migration Rate

a. Migration Rate

For this experiment, we used striping with the user-optimal migration scheme (STR-MIG). Data is allocated by striping over local disk (Disk A) and remote disk (Disk B). If the request response time of remote disk is larger than that of local disk by δ, then data migration from remote disk to local disk is initiated and vice versa. We used

[4]For intuitive explanation, we call Disk A as local disk and Disk B as remote disk in single-user environment.

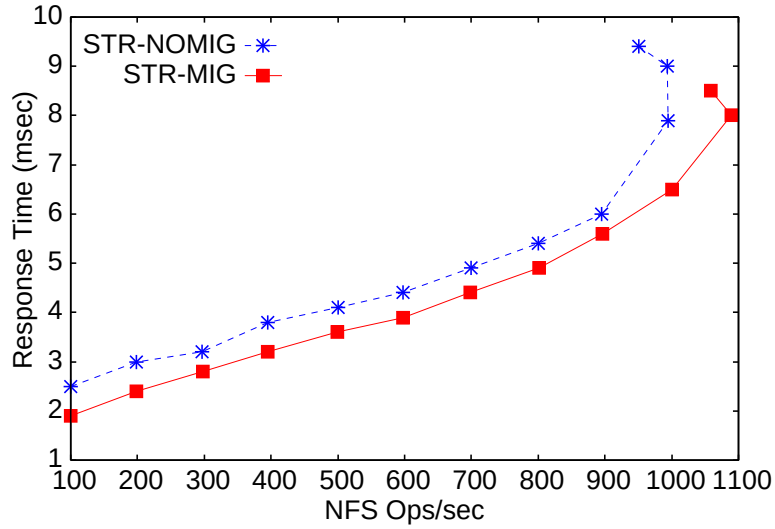
9ms of RTT. Fig. 27 shows the results with different number of migration tokens.[5] The base response time in the case of four tokens was better than that of one token case by 13.6% because a system with four tokens could reduce the number of remote disk accesses more due to a faster migration rate. However, faster migration caused earlier throughput saturation due to higher loads. In the system with one token, data migration occurred in one direction, from remote disk to local disk as the load is varied from 100 operations/sec to the saturation point. In the system with four tokens, the data was migrated from remote disk to local disk at lower NFS throughputs and from local disk to remote disk at higher NFS throughputs. The faster migration at lower throughputs to local disk resulted in higher loads at the local device at higher throughputs, which in turn, resulted in migrating data to remote disk.

Setting number of migration tokens dynamically based on workloads and network latencies seems feasible and is a subject of future study. In all the following experiments, we used the migration rate of four tokens per second.

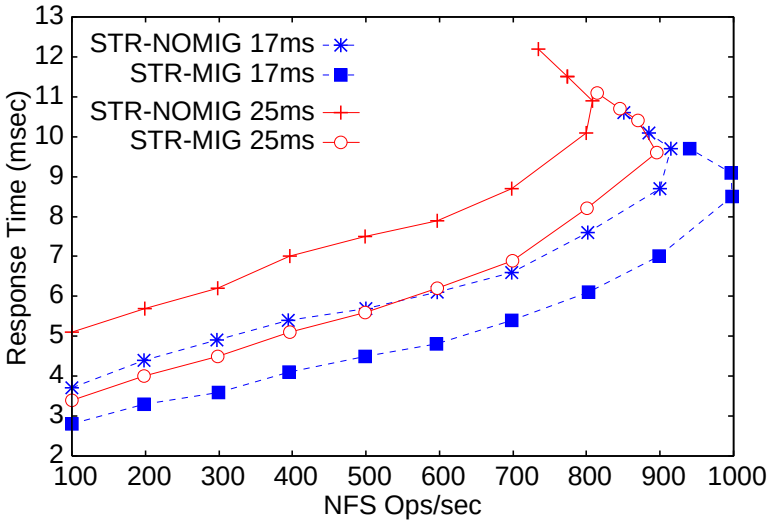
b. Striping

Fig. 28(a) shows the results of two configurations; normal striping (STR-NOMIG) and striping with user-optimal migration (STR-MIG). STR-MIG showed better base response time by 24% and at higher loads, it improved performance from 6.7% to 20% when compared to STR-NOMIG. In addition, it could increase the saturation point by 9.6% over that of STR-NOMIG. This performance improvement is attributed to the fact that striping suffers from larger remote disk access latency, while user-optimal migration could reduce the number of remote disk accesses by migrating

[5]Since the SPECsfs benchmark reports the response time vs. delivered throughput, as opposed to offered load, attempts to exceed the saturation point can result in fewer operations per second than attempted. In Fig. 27, the actual x value of data point indicates delivered throughput.



(a) Striping with User-optimal Migration



(b) Impact of Network Latency in Striping

Fig. 28. Single-user Striping

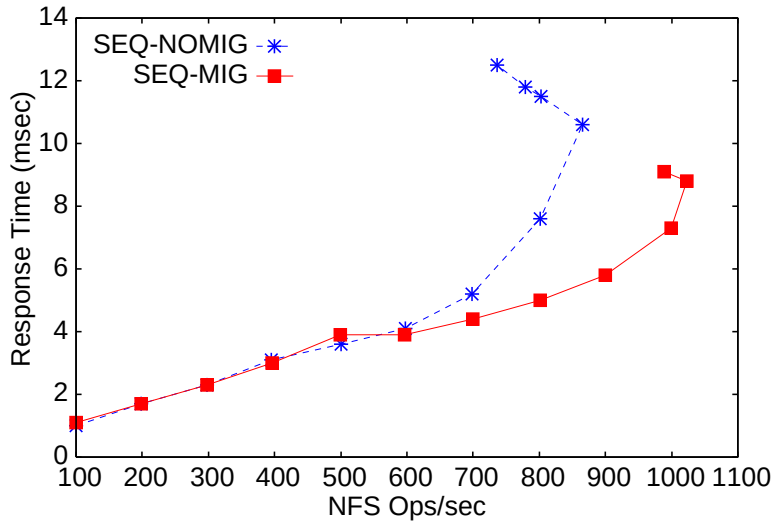
data gradually from remote disk to local disk. In normal striping (STR-NOMIG), the remote response times limit the realized throughput. User-optimal migration could increase the throughput saturation point by migrating data from remote disk.

Fig. 28(b) shows the impact of network latency. As we expected, migration showed larger performance improvement as network latency got larger. STR-MIG with 17ms RTT improved the base response time by 24.3% and improved the response time for higher loads from 8.5% to 26.5%. It increased the throughput saturation point by 9.1%. Similarly, STR-MIG with 25ms RTT improved the base response time by 33.3% and improved response times at higher loads from 7.0% to 29.8%. The throughput saturation point was increased by 10.8%. High network latency directly impacted performance of the normal striping system, while STR-MIG was less sensitive to network latency increase because data migration can effectively reduce the impact of higher network latency (by improving locality).

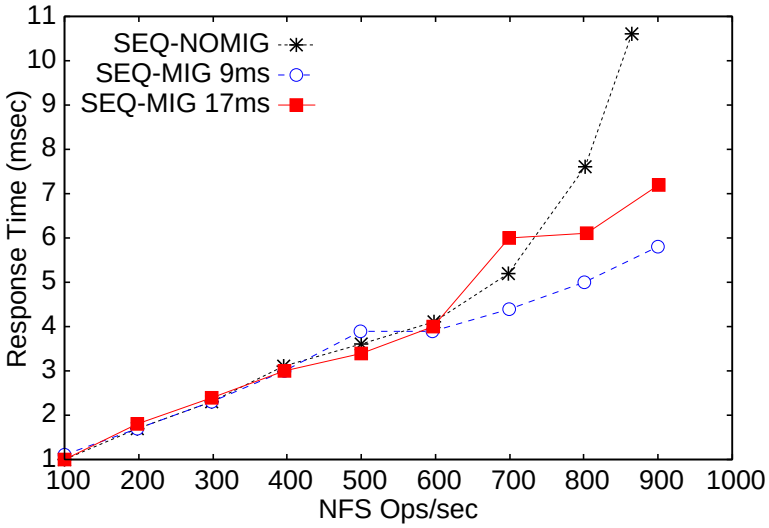
c. Sequential Allocation

Fig. 29(a) shows the results of sequential allocation (SEQ-NOMIG) and sequential allocation with user-optimal migration (SEQ-MIG). When local disk is not heavily loaded (until load 400), there is no difference between the two schemes. At load 600, SEQ-NOMIG started suffering from heavy local disk load, while SEQ-MIG migrated active data to remote disk to improve load balance.[6] This caused performance improvement in response time up to 45.3% at higher loads. At the same time, it increased the throughput saturation point by 18.2%. In contrast to normal striping, in sequential allocation, the bottleneck of the system was local disk. Larger saturation point of user-optimal migration is attributed to the fact that data migration to

[6]When migration was initiated at load 500, it incurred a little overhead on response time.



(a) Sequential Allocation with User-optimal Migration



(b) Impact of Network Latency in Sequential Allocation

Fig. 29. Single-user Sequential Allocation

remote disk could reduce the load of local disk.

Fig. 29(b) shows the impact of network latency. In SEQ-MIG with 17ms RTT, number of migrations were fewer than in SEQ-MIG with 9ms RTT. This resulted in worse response time than that of system with 9ms RTT at higher loads. In 25ms RTT, migration didn't happen; user-optimal migration decided not to migrate data because of remote disk's higher network latency. The data set was small enough to fit on the local disk and hence the network latency has no impact on the performance on the SEQ-NOMIG system.

d. Discussion of Results

From Fig. 28(a) and Fig. 29(a), we can observe that systems employing migration (STR-MIG and SEQ-MIG) exhibited better characteristics than systems without migration, with striping or sequential allocation. STR-MIG offered better response times than STR-NOMIG at most NFS throughputs. SEQ-MIG offered similar response times as SEQ-NOMIG at lower loads while improving the throughput saturation points and response times at higher loads. This adaptability comes from the flexibility of user-optimal migration policy; the direction of migration changes based on changes in loads at different disks.

Fig. 30 shows the performance comparison between two migration schemes; migration based on locality and user-optimal migration (STR-MIG and SEQ-MIG). In migration based on the locality scheme, every accessed data on remote disk is migrated to local device while *inactive* data is migrated to remote device. The performance of this scheme became worse as load increased (due to the heavily loaded local disk). When the working set size exceeded the local device size, it could only deliver very low throughput due to thrashing. User-optimal migration is observed to perform better than the locality based migration scheme and achieved higher throughput saturation

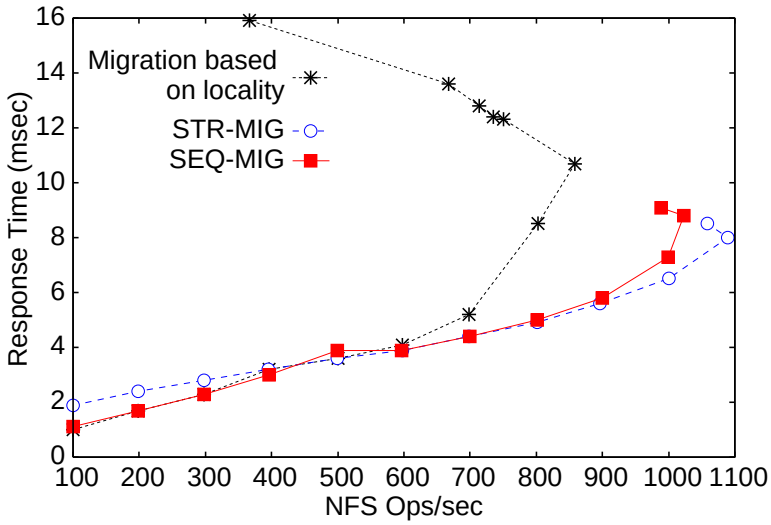


Fig. 30. The Comparison Between Different Migration Schemes

points.

4. Multi-User Environment

For multi-user environment with striping, we used three NFS Servers A, B and C with two load generators each. NFS server A exports a 32GB file system which spans Disk A, Disk B and Disk C to its load generators. NFS server B exports a 32GB file system which spans Disk B, Disk C and Disk A to its load generators. NFS server C is similar. Thus, NFS server A, B and C share storage resources (Disk A, Disk B and Disk C), and data of three servers may be intermixed on the three disks. For multi-user environment with sequential allocation, we used two NFS Servers A and B with Disk A and Disk B.

a. Striping

In this experiment, we used two configurations. In the first configuration, three NFS servers used striping simultaneously (STR-NOMIG) and in the second one, three

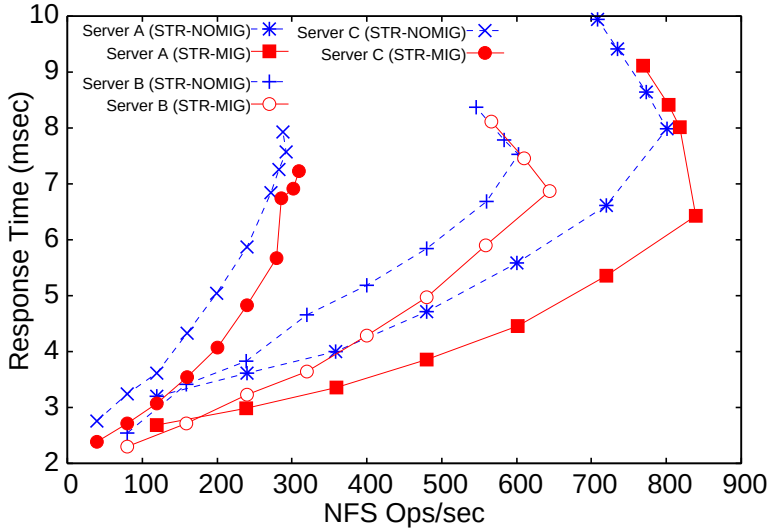
NFS servers used striping with user-optimal migration concurrently (STR-MIG). We compare results between the two configurations here.

We ran the three servers at different loads. We operated NFS server A at varying loads from 120 to 1200 and simultaneously operated NFS server B at varying loads from 80 to 800 and NFS server C from 40 to 400. Thus, the load distribution of server A, B and C was 3:2:1. This allowed us to study the impact of different loads at the different servers.[7] We used 9ms of RTT. Fig. 31(a) and 31(b) show the result of each NFS server's performance and average performance of all three servers. In Fig. 31(a), using user-optimal migration, performance improvement in NFS server A was from 7.3% to 20.1% and performance improvement in NFS server B and in NFS server C were similar to server A. We could observe that each server tried to improve locality; each server migrated its remote data to its local disk. As a result, each server could improve response times significantly compared to normal striping case. Since the data was originally striped across all the disks, the loads at all the servers together impact the response times at the disks. Fig. 31(b) compares the average performance of two configurations. Our system (STR-MIG) improved response time from 6.6% to 20.4% and increased the saturation point by 5.4% compared to STR-NOMIG.

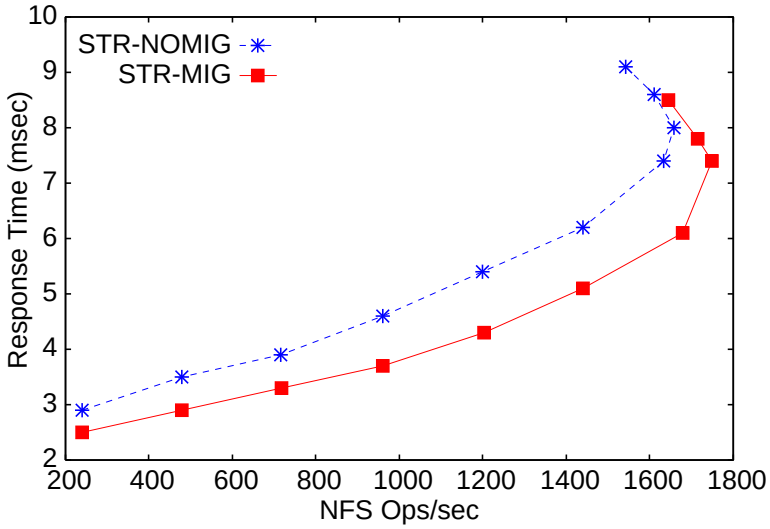
Fig. 32(a) shows the total number of requests served at Disk A, Disk B and Disk C as a function of time from the viewpoint of NFS server A. We can observe that migration from Disk B and Disk C to Disk A was continuously done during the benchmark to reduce the number of remote disk accesses (denoted as a Disk B request and Disk C request in Fig. 32(a)). Without migration, nearly equal number of requests would have been observed at the three disks.

This migration helped the system to improve locality, but it may result in lower

[7]We also conducted experiments when each server has the same load. Similar results were observed and hence not reported here.

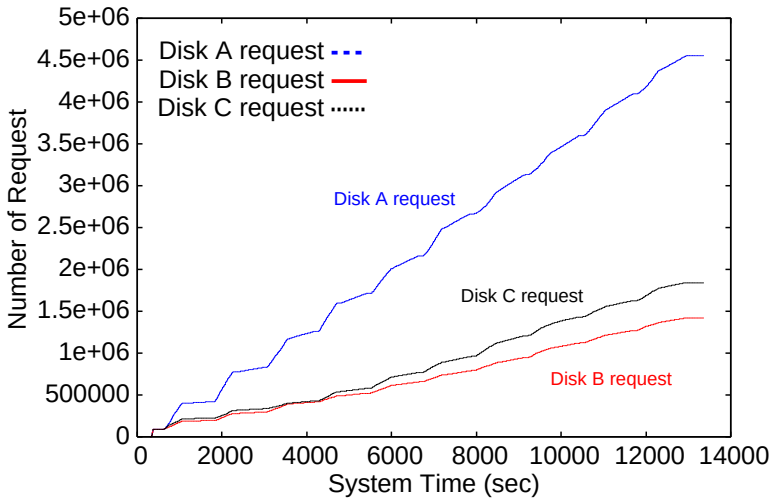


(a) The Performance of Individual Server

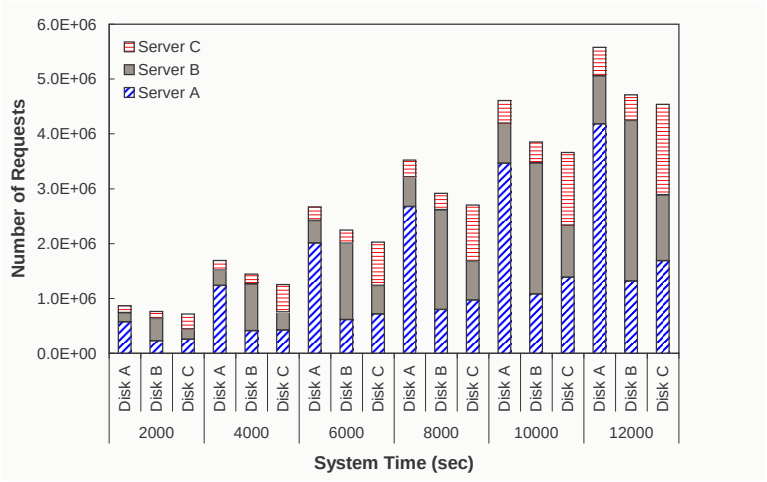


(b) Average Performance of All Servers

Fig. 31. Multi-user Striping with User-optimal Migration



(a) Number of Requests of NFS Server A



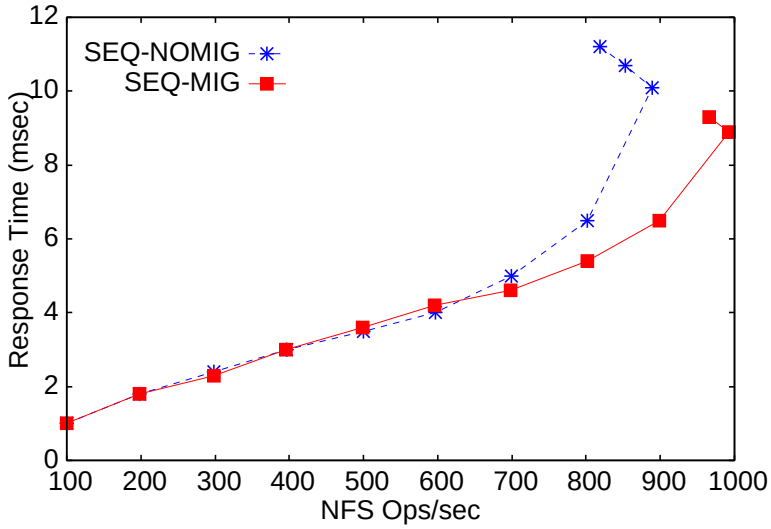
(b) Request Distribution

Fig. 32. Request Statistics

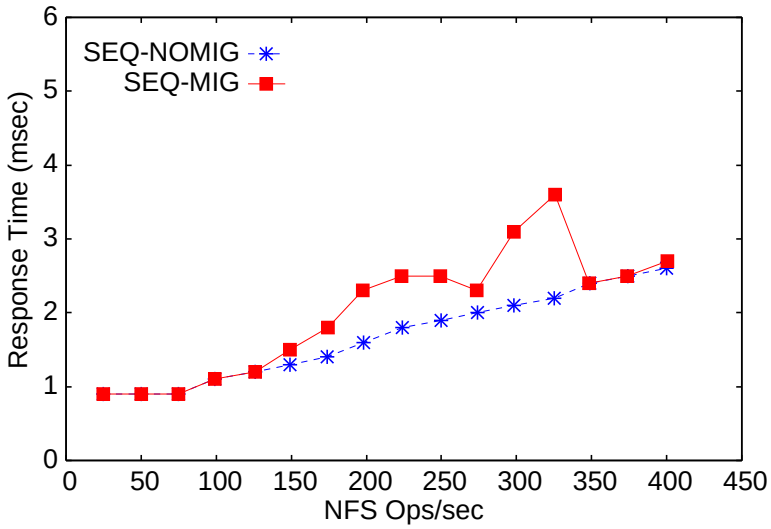
balance of load. Fig. 32(b) presents overall request distribution among the three disks during the experiment. Each group consists of three bars: the left most bar denotes number of requests issued on Disk A and the next two bars represent the number of requests issued on Disk B and Disk C from the three NFS servers. In Fig. 32(b), we can observe the following. First, the fraction of the server A requests served at disk A gradually increased indicating that server A increased its locality during the experiment. Similarly Server B increased its locality at Disk B. Disk C showed relatively balanced request distribution because server C was the most lightly loaded server. Second, in a normal striping system, the number of requests issued on Disk A, Disk B and Disk C would be the same (in most cases). It is observed that load balance was decreased by up to 12.8% in user-optimal migration. This was a consequence of the improved locality and the imbalanced load across the three servers. It is also noted that the load imbalance at the disks is not at the same level as the load imbalance at the servers (3:2:1).

b. Sequential Allocation

Similar to the previous experiment, we configured two NFS servers to use sequential allocation without migration (SEQ-NOMIG) and with migration (SEQ-MIG). Server A allocates its data first on Disk A and then on Disk B in a concatenated fashion and similarly, server B allocates its data first on Disk B and then on Disk A, i.e., locality is maximized for each server's data. We tested loads of 100 to 1100 at server A and tested loads of 25 to 400 at server B. Server B's load was kept at a quarter of server A's load in this experiment. Different loads at the servers result in different loads at the devices when data migration is not employed. Fig. 33(a) and 33(b) show the result of each server. At load 500, server A started data migration to Disk B due to the heavy load of Disk A. User-optimal migration improved the performance of the heavily



(a) NFS Server A



(b) NFS Server B

Fig. 33. Multi-user Sequential Allocation with User-optimal Migration

loaded server A. It improved throughput saturation point by 11.7% at server A. Data migration of server A impacted the performance of server B. However, response time improvement in server A (e.g., from 10.1ms to 6.5ms) was more significant than the response time increase in server B (e.g., from 2.2ms to 3.6ms).

This experiment also highlights the local good versus global good tension that is addressed by our approach. While server A's performance improved in this experiment at the cost of server B's performance, overall (global) average response time also improved at the same time (since there are more requests at server A). Over time, the response times of the two devices will be within δ (migration threshold) and migration will stop.

c. Discussion of Results

When multiple users share storage resources by striping and those devices have performance differences (due to disk hardware or network latencies), user-optimal migration could improve performance by improving locality. In multi-user workloads, the resulting improvement in locality did not decrease load balance much. If multiple users use sequential allocation and there are load differences, user-optimal migration allows a heavily loaded server to reduce its load by migrating its data to a server with less load.

C. Related Work

Data (file) migration has been studied extensively earlier in systems where disks and tapes are used [57, 68, 69, 70, 71]. As mentioned earlier, we are considering data migration among disks. Characteristics of data migration can be much different in two cases. First, the ratio of access time of local disk and remote disk is much smaller

than that of disk and tape, the decision of data migration among disks can be different from disk and tape case. Second, the performance of disk is always better than that of tape, heavily referenced data always migrate from tape to disk. In contrast, in our case, heavily referenced data may migrate in different directions depending on workloads.

Data migration has also been studied in HP's AutoRAID system where data can migrate between a mirrored device and a RAID device [42]. In these systems, hot[8] data is migrated to faster devices and cold data is migrated to slower devices to improve the access times of hot data by keeping it local to faster devices. When data sets are larger than the capacity of faster devices in such systems, thrashing may occur. Some of the systems detect thrashing and may preclude migration during such times [42]. Our data migration scheme is more general and considers migration among multiple disks over a network with dynamically changing characteristics. Data migration direction is decided by workload changes - a faster device at some time can become a slower device later and vice versa.

Aqueduct [72] deals with data migration among storage devices without application down-time. It takes a control-theoretical approach to provide QoS guarantees to client applications during a data migration, while still accomplishing the data migration in as short a time as possible. Similar to our system, Aqueduct uses I/O request response time as a performance measure, but there are significant differences from our work. First, Aqueduct is only focused on how to migrate data without significantly impacting foreground activities. Other migration decision issues mentioned in the earlier section, i.e., migration initiation, destination, data set to migrate were statically decided (e.g., by system administrator). Only migration speed is deter-

[8]Data that is currently being accessed or active is called "hot" data and data that is not currently being accessed or inactive is called "cold" data.

mined dynamically according to the impact on foreground performance. In contrast, in our system, all of the migration issues are decided dynamically based on system characteristics such as workload, disk access locality and disk load. Second, we focus on migrating active data (that is currently being accessed), which results in reduced migration cost.

The problems of data distribution in large-scale storage systems are addressed in [73, 74, 75]. This body of work is focused on data migration as a result of configuration changes such as insertions or removals of disks. Handling heterogeneity in shared-disk file systems is addressed in [76], where observed request latencies were used for adaptive file set migration among servers to balance the metadata workload on servers. Our work is different from this work in two points: (a) Our system is dealing with migrating block-level data in contrast to file-level data (e.g., file sets) and (b) we consider a multi-user environment where several applications compete for shared resources.

Request distribution, load balancing and memory cache hits are considered in LARD [77] for improving the performance of clustered file servers. D-SPTF [78] additionally considered disk head scheduling when redirecting requests in brick-based storage systems where multiple copies of a data item may be present. In this study, we are considering the case where there is single copy of a data item, i.e., no replication.

A number of studies have pointed to the need for improving the locality of accesses when devices are networked [10, 79, 80]. These studies have pointed to the importance of reducing the impact of network access latencies on applications' performance. File systems and other applications already employ caching to exploit the locality characteristics of data accesses. As mentioned earlier, we consider *longer-term locality* considerations at the I/O or storage system beyond what is already exploited in memory.

D. Discussion

As an alternative approach to ours, we could think of a policy which would implicitly create a storage hierarchy, where heavily referenced data is migrated in the local storage, and less referenced data is migrated in the more distant networked storage. This idea is tested against our system and the results are presented in Fig. 30. The primary difference from hierarchical memory systems is that the loads at the devices can exceed disk throughput capacities and queuing delays can build up. Hence, a local disk can be slower if all the needed data is on the local disk (and hence a higher load) than a remote disk (because of lower load). A second reason is that the access times of local and remote devices are a factor of 2 or 3 away, not an order of magnitude higher as in cache-memory hierarchy.

Disks have been traditionally attached to hosts or servers. Only recently, they are being attached to networks, either Storage Area Networks or IP networks. Most of current storage systems do static allocation of storage at the time of file system allocation and do not provide the flexibility that is being pursued here.

E. Summary

In this chapter, we have proposed user-optimal migration for balancing load and locality in a networked storage system. Through realistic experiments on an experimental testbed, user-optimal migration is shown to automatically and transparently balance disk access locality and load balance through migration of active data blocks. Migration is decided by longer-term performance metrics and it is shown to adapt to changes of workloads and loads in each storage device in a networked environment. User-optimal migration was shown to improve locality with an allocation policy of striping and to improve load balance with an allocation policy of sequential allocation.

We have shown that user-optimal migration is effective in multi-user environments and can effectively share common storage resources under different user loads. We have also shown that the user-optimal migration policy can outperform policies based on maximizing locality or load balance alone.

CHAPTER IV

CONCLUSIONS

A. Dissertation Summary

In this dissertation, we have considered some of the challenges posed by the emerging networked storage systems. Traditional file systems employ a static allocation approach where the entire storage space is claimed at the time of the file system creation. This results in space on the disk is not allocated well across multiple file systems. We have proposed virtual allocation employing an allocation-on-write policy for improving the flexibility of managing storage across multiple file systems. By separating the storage allocation from the file system creation, common storage space can be pooled across different file systems and flexibly managed to meet the needs of different file systems. We have shown the effectiveness of our approach through the evaluation of it.

Next, we have examined the problem of balancing locality and load in networked storage systems with multiple storage devices. Data distribution affects locality and load balance across the devices in a networked storage system. We have proposed user-optimal migration which balances disk access locality and load balance automatically and transparently through migration of active data blocks. Migration is decided by longer-term performance metrics and it is shown to adapt to changes of workloads and loads in each storage device in a networked environment. We have presented the design and an evaluation of it through realistic experiments.

We introduced the motivations of this study in Chapter I. In Chapter II, we presented the design of virtual allocation and performance evaluation of it. In Chapter III, we discussed user-optimal migration architecture with performance evaluation.

B. Further Work

In this section, we will present the discussion issues and propose further extensions for two architectures.

1. Virtual Allocation

Fig. 34 shows various ways in which VA can be deployed. As a research prototype, we implemented VA using a device driver. It is possible to deploy VA inside file system. In this case, VA allows file system to allocate its storage space flexibly, but it may not be easy to share storage space with other file systems. For more scalability, it is possible to deploy VA inside a storage controller or inside a router. This can result in flexible storage allocation across multiple operating systems.

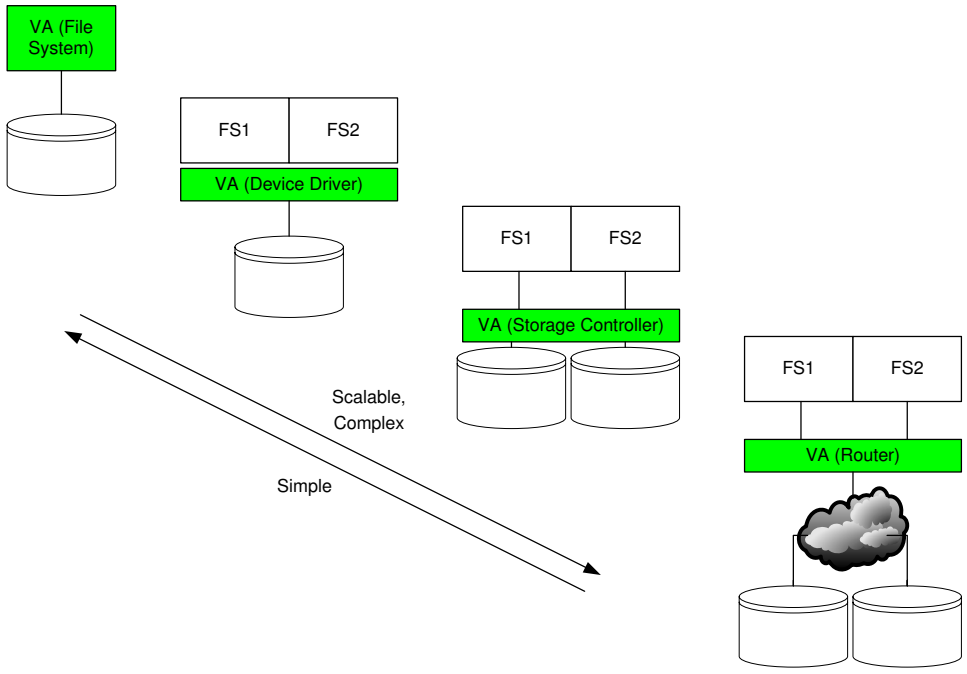


Fig. 34. Real-World Examples of VA Deployment

VA has several limitations. First, it may not support mixed workloads well

because of the allocation policy. If the small-file workload results in excessive extent fragmentations, then the performance of sequential read or write operation of a large file will suffer. This problem could be solved through the separation of large-file workloads and small-file workloads similar to [81]. Second, for large-file workloads, the environments where small-write cost is expensive as in RAID-5, VA requires additional resources such as NVRAM. Third, the deployment of VA on an existing file system requires reorganization or copying of file system data onto the VA devices.

VA can be extended easily to support snapshots. The VA block map can be extended to create entries for a number of versions of data on the disk. Each time a snapshot is created, all writes, including old blocks (previous versions of data), can be treated to require allocations of new space on the disk.

We believe that our system can be extended employing the IP connectivity of I/O devices (e.g., iSCSI [2]). The IP connectivity of I/O devices makes it possible for storage devices to be added to the network and enables storage to be treated as a discoverable resource. Virtual allocation, through its separation of storage allocation from the file system creation, potentially allows file systems to span multiple devices across the network. When storage resources need to be shared over a network, virtual allocation's block map could be extended to enable local disk caching in unallocated disk space to offset large data access latencies. Recent studies have shown the importance of caching in such networked storage systems [10, 11]. This could be pursued in the future. Another topic of interest is to investigate the interaction between multiple file systems and VA with RAID. Our allocate-on-write policy would need to be suitably modified to fit into the logical device characteristics advertised by the RAID.

2. User-Optimal Migration

In our approach, data is migrated to other locations only when there is a benefit in terms of performance. Ideally, when all the devices have nearly the same response times, data is not migrated from one device to another. To avoid oscillations and to take costs of migration into account, we set thresholds (the δ parameter) to dampen migration around this ideal operating point. Our system adapts to workload and data distribution changes to reach this operating point.

When block network storage protocol devices (like iSCSI) are used, data cannot be simultaneously shared by multiple file systems without additional locking mechanisms. In the work here, we assume a single file system or user is accessing data at a time. In such scenarios, migration will provide similar benefits as caching [82]. Andrew File System [83] employed disk caching to reduce the load on the file server and to exploit the disk access locality at each client.

In this study, we considered two allocation policies; striping and sequential allocation. Random allocation has been proposed by others [51, 53] to achieve load balancing with heterogeneous devices and for minimizing disruptions as devices are added to the system. In these systems, multiple copies of data is often employed. The remapping required for migration would have to be suitably modified to work with random allocation and multiple copies of data. This could be pursued in the future.

Our migration policy could be evaluated in more diverse workloads and tested in a wide area testbed such as Planetlab [84].

C. Concluding Remarks

In this dissertation, we presented the design and implementation of flexible storage allocation and a data migration policy in a networked storage system. We motivated

the need for these solutions and provided rationale for our design choices. Results show that our systems provide better flexibility or better performance compared to other systems. We believe that the design principles presented in this thesis are very useful and they could be easily applied to a real product.

REFERENCES

[1] D. Patterson, G. Gibson, and R. Katz, "A case for redundant array of inexpensive disks (RAID)," in *Proc. ACM SIGMOD*, September 1988, pp. 109-116.

[2] M. Krueger, R. Haagens, C. Sapuntzakis, and M. Bakke, "Small computer systems interface protocol over the Internet (iSCSI) requirements and design considerations," RFC 3347, Internet Engineering Task Force, July 2002, Available: http://www.ietf.org/rfc/rfc3347.txt; Accessed: January 27, 2007.

[3] A. Sporer, "Delivering storage as a service," Available: http://www.emc.com/pdf/service/C1086_del_storage_service_ldv.pdf; Accessed: January 27, 2007.

[4] G. Gibson, D. Nagle, K. Amiri, F. Chang, E. Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, D. Rochberg, and J. Zelenka, "File server scaling with network-attached secure disks," in *Proc. ACM SIGMETRICS*, June 1997, pp. 272-284.

[5] Adaptec, Inc., "Snap server product overview," Available: http://www.adaptec.com/en-US/products/nas; Accessed: January 27, 2007.

[6] R. Horst, "TNet: A reliable system area network," *IEEE Micro*, vol. 15, no. 1, pp. 37-45, February 1995.

[7] Seagate LLC., "Fibre channel: The digital highway made practical," Available: http://www.seagate.com/support/kb/disc/tp/fc.html; Accessed: January 27, 2007.

[8] R. Meter, "A brief survey of current work on network attached peripherals (extended abstract)," *Operating Systems Review*, vol. 30, no. 1, pp. 63-70, January 1996.

[9] *The European Datagrid Project,* Available: http://eu-datagrid.web.cern.ch/eu-datagrid; Accessed: January 27, 2007.

[10] W. Ng, and B. Hillyer, "Obtaining high performance for storage oursourcing," in *Proc. ACM SIGMETRICS*, June 2001, pp. 322-323.

[11] X. He, Q. Yang, and M. Zhang, "A caching strategy to improve iSCSI performance," in *Proc. IEEE LCN*, November 2002, pp. 278-288.

[12] *The SDSC Storage Resource Broker,* Available: http://www.npaci.edu/DICE/SRB; Accessed: January 27, 2007.

[13] L. Huang, G. Peng, and T. Chiueh, "Multi-dimensional storage virtualization," in *Proc. ACM SIGMETRICS*, June 2004, pp. 14-24.

[14] J. Menon, D. Pease, R. Rees, L. Duyanovich, and B. Hillsberg, "IBM Storage Tank - A heterogeneous scalable SAN file system," *IBM Systems Journal*, vol. 42, no. 1, pp. 250-267, 2003.

[15] M. Mesnier, G. Ganger, and E. Riedel, "Object-based storage," *IEEE Communications Magazine*, vol. 41, no. 8, pp. 84-90, August 2003.

[16] J. MacCormick, N. Murphy, M. Najork, C. Thekkath, and L. Zhou, "Boxwood: Abstractions as the foundation for storage infrastructure," in *Proc. USENIX OSDI*, December 2004, pp. 105-120.

[17] M. Rosenblum, and J. Ousterhout, "The design and implementation of a log-structured file system," *ACM Transactions on Computer Systems*, vol. 10, no. 1, pp. 26-52, February 1992.

[18] D. Hitz, J. Lau, and M. Malcom, "File system design for an NFS file server appliance," in *Proc. USENIX Winter*, January 1994, pp. 235-246.

[19] M. Sivathanu, V. Prabhakaran, F. Popovici, T. Denehy, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Semantically-smart disk systems," in *Proc. USENIX FAST*, March 2003, pp. 73-88.

[20] M. Sivathanu, L. Bairavasundaram, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Life or death at block-level," in *Proc. of USENIX OSDI*, December 2004, pp. 379-394.

[21] M. McKusick, W. Joy, S. Leffler, and R. Fabry, "A fast file system for UNIX," *ACM Transactions on Computer Systems*, vol. 2, no. 1, pp. 181-197, February 1984.

[22] G. Ganger, and Y. Patt, "Metadata update performance in file systems," in *Proc. USENIX OSDI*, November 1994, pp. 49-60.

[23] R. Wang, D. Patterson, and T. Anderson, "Virtual log based file systems for a programmable disk," in *Proc. USENIX OSDI*, February 1999, pp. 29-43.

[24] R. Meter, "Observing the effects of multi-zone disks," in *Proc. USENIX*, January 1997, pp. 19-30.

[25] J. Peacock, "The counterpoint fast file system," in *Proc. USENIX Winter*, January 1988, pp. 243-249.

[26] L. McVoy, and S. Kleiman, "Extent-like performance from a UNIX file system," in *Proc. USENIX Winter*, January 1991, pp. 33-44.

[27] K. Smith, and M. Seltzer, "A comparison of FFS disk allocation policies," in *Proc. USENIX*, January 1996, pp. 15-26.

[28] *Bonnie++ Benchmark,* Available: http://www.coker.com.au/bonnie++, Accessed: January 27, 2007.

[29] *PostMark Benchmark,* Available: http://www.acnc.com/benchmarks.html; Accessed: January 27, 2007.

[30] W. Vogels, "File system usage in Windows NT 4.0," in *Proc. ACM SOSP*, December 1999, pp. 93-109.

[31] *TPC-C Benchmark,* Available: http://www.tpc.org; Accessed: January 27, 2007.

[32] *Hammerora,* Available: http://hammerora.sourceforge.net; Accessed: January 27, 2007.

[33] D. Ellard, J. Ledlie, P. Malkani, and M. Seltzer, "Passive NFS tracing of email and research workloads," in *Proc. USENIX FAST*, March 2003, pp. 203-216.

[34] N. Zhu, J. Chen, T. Chiueh, and D. Ellard, "TBBT: Scalable and accurate trace replay for file server evaluation," in *Proc. ACM SIGMETRICS*, June 2005, pp. 392-393.

[35] D. Ellard, and M. Seltzer, "NFS tricks and benchmarking traps," in *Proc. USENIX: FREENIX Track*, June 2003, pp. 101-114.

[36] *ZCAV Benchmark,* Available: http://www.coker.com.au/bonnie++/zcav; Accessed: January 27, 2007.

[37] *ReiserFS,* Available: http://www.namesys.com; Accessed: January 27, 2007.

[38] S. Best, "JFS Overview," Available: http://www.ibm.com/developerworks/library/l-jfs.html; Accessed: January 27, 2007.

[39] Symantec Corp., "Veritas volume manager overview," Available: http://www.symantec.com/enterprise/products/overview.jsp?pcid=1020&pvid=203_1; Accessed: January 27, 2007.

[40] M. Kaczmarski, T. Jiang, and D. Pease, "Beyond backup toward storage management," *IBM Systems Journal*, vol. 42, no. 1, pp. 322-337, 2003.

[41] R. English, and A. Stepanov, "Loge: A self-organizing storage device," in *Proc. USENIX Winter*, January 1992, pp. 237-252.

[42] J. Wilkes, R. A. Golding, C. Staelin, and T. Sullivan, "The HP AutoRAID hierarchical storage system," *ACM Transactions on Computer Systems*, vol. 14, no. 1, pp. 108-136, February 1996.

[43] E. Lee, and C. Thekkath, "Petal: Distributed virtual disks," in *Proc. ACM ASPLOS*, October 1996, pp. 84-92.

[44] 3PARdata, Inc., "Thin provisioning," Available: http://www.3pardata.com/products/thinprovisioning.html; Accessed: January 27, 2007.

[45] W. Jonge, M. Kaashoek, and W. Hsieh, "The Logical Disk: A new approach to improving file systems," in *Proc. ACM SOSP*, December 1993, pp. 15-28.

[46] Vmware, Inc., "Vmware workstation," Available: http://www.itc.virginia.edu/atg/techtalks/powerpoint/vmware/sld018.htm; Accessed: January 27, 2007.

[47] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "OceanStore: An architecture for global-scale persistent storage," in *Proc. ACM ASPLOS*, November 2000, pp. 190-201.

[48] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer, "FARSITE: Federated, available, and reliable storage for an incompletely trusted environment," in *Proc. USENIX OSDI*, December 2002, pp. 1-14.

[49] L. Cox, C. Murray, and B. Noble, "Pastiche: Making backup cheap and easy," in *Proc. USENIX OSDI*, December 2002, pp. 285-298.

[50] E. Zadok, R. Iyer, N. Joukov, G. Sivathanu, and C. Wright, "On incremental file system development," *ACM Transactions on Storage Systems*, vol. 2, no. 2, pp. 161-196, May 2006.

[51] IBM Corp., "Collective Intelligent Bricks," Available: http://www.almaden.ibm.com/StorageSystems/Advanced_Storage_Systems/Intelligent_Bricks; Accessed: January 27, 2007.

[52] J. Gray, "What happens when processing, storage bandwidth are free and infinite?," *Keynote speech at ACM IOPADS*, November 1997.

[53] S. Ghemawat, H. Gobioff, and S-T. Leung, "The Google file system," in *Proc. ACM SOSP*, October 2003, pp. 29-43.

[54] Terrascale Technologies, Inc., "TerraGrid cluster file system," Available: http://www.terrascale.com/prod_over_e.html; Accessed: January 27, 2007.

[55] F. Schmuck, and R. Haskin, "GPFS: A shared-disk file system for large computing clusters," in *Proc. USENIX FAST*, January 2002, pp. 231-244.

[56] IBM Corp., "An introduction to GPFS," Available: http://www-03.ibm.com/systems/clusters/software/whitepapers/gpfs_intro.pdf; Accessed: January 27, 2007.

[57] B. Gavish, and O. Sheng, "Dynamic file migration in distributed computer systems," *Communications of the ACM*, vol. 33, no. 2, pp. 177-189, February 1990.

[58] L. Yin, S. Uttamchandani, and R. Katz, "SmartMig: Risk-modulated proactive data migration for maximizing storage system utility," in *Proc. IEEE MSST*, May 2006.

[59] E. Markatos, and T. LeBlanc, "Load balancing vs. locality management in shared-memory multiprocessors," *Tech. Report: TR399, University of Rochester*, 1991.

[60] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz, "An evaluation of directory schemes for cache coherence," in *Proc. IEEE ISCA*, May/June 1988, pp. 280-289.

[61] V. Borkar, and P. Kumar, "Dynamic Cesaro-Wardrop equilibration in networks," *IEEE Transactions on Automatic Control*, vol. 48, no. 3, pp. 382-396, March 2003.

[62] L. Qiu, R. Yang, Y. Zhang, and S. Shenker, "On selfish routing in Internet-like environments," in *Proc. ACM SIGCOMM*, August 2003, pp. 151-162.

[63] S. Floyd, and V. Jacobsen, "Random early detection gateways for congestion avoidance," *ACM/IEEE Transactions on Networking*, vol. 1, no. 4, pp. 397-413, August 1993.

[64] Smitha, and A. Reddy, "LRU-RED: An active queue management scheme to contain high bandwidth flows at congested routers," in *Proc. Globecom*, November 2001, pp. 2311-2315.

[65] *SPECsfs Benchmark,* Available: http://www.spec.org/osg/sfs97r1; Accessed: January 27, 2007.

[66] R. Martin, and D. Culler, "NFS sensitivity to high performance networks," in *Proc. ACM SIGMETRICS*, May 1999, pp. 71-82.

[67] *Netem,* Available: http://linux-net.osdl.org/index.php/Netem; Accessed: January 27, 2007.

[68] E. Miller, and R. Katz, "An analysis of file migration in a UNIX supercomputing environment," in *Proc. USENIX Winter*, January 1993, pp. 421-434.

[69] M. Lubeck, D. Geppert, K. Nienartowicz, M. Nowak, and A. Valassi, "An overview of a large-scale data migration," in *Proc. IEEE MSST*, April 2003, pp. 49-55.

[70] A. Smith, "Long term file migration: Development and evaluation of algorithms," *Communications of ACM*, vol. 24, no. 8, pp. 521-532, August 1981.

[71] V. Cate, and T. Gross, "Combining the concepts of compression and caching for a two-level file system," in *Proc. ACM ASPLOS*, April 1991, pp. 200-211.

[72] C. Lu, G. Alvarez, and J. Wilkes, "Aqueduct: Online data migration with performance guarantees," in *Proc. USENIX FAST*, January 2002, pp. 219-230.

[73] A. Brinkmann, K. Salzwedel, and C. Scheideler, "Efficient, distributed data placement strategies for storage area networks (extended abstract)," in *Proc. SPAA*, July 2000, pp. 119-128.

[74] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence, "FAB: Building distributed enterprise disk arrays from commodity components," in *Proc. ACM ASPLOS*, October 2004, pp. 48-58.

[75] R. Honicky, and E. Miller, "A fast algorithm for online placement and reorganization of replicated data," in *Proc. IEEE IPDPS*, April 2003, pp. 57.

[76] C. Wu, and R. Burns, "Handling heterogeneity in shared-disk file systems," in *Proc. ACM/IEEE SC*, November 2003, pp. 7.

[77] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum, "Locality-aware request distribution in cluster-based network servers," in *Proc. ACM ASPLOS*, October 1998, pp. 205-216.

[78] C. Lumb, R. Golding, and G. Ganger, "D-SPTF: Decentralized request distribution in brick-based storage systems," in *Proc. ACM ASPLOS*, October 2004, pp. 37-47.

[79] Y. Lu, F. Noman, and D. Du, "Simulation study of iSCSI-based storage system," in *Proc. IEEE MSST*, April 2004 , pp. 101-110.

[80] P. Radkov, L. Yin, P. Goyal, P. Sarkar, and P. Shenoy, "A performance comparison of NFS and iSCSI for IP-connected networked storage," in *Proc. USENIX FAST*, April 2004, pp. 101-114.

[81] D. Anderson, J. Chase, and A. Vahdat, "Interposed request routing for scalable network storage," in *Proc. USENIX OSDI*, October 2000, pp. 259-272.

[82] Y. Hu, and Q. Yang, "DCD - Disk Caching Disk: A new approach for boosting I/O performance," in *Proc. IEEE ISCA*, May 1996, pp. 169-178.

[83] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, and R. Sidebotham, "Scale and performance in a distributed file system," *ACM Transactions on Computer Systems*, vol. 6, no. 1, pp. 51-81, February 1988.

[84] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: An overlay testbed for broad-coverage services," *Computer Communications Review*, vol. 33, no. 3, pp. 3-12, July 2003.

VITA

Suk Woo Kang received his B.S. and M.S. degrees in Electrical Engineering from Seoul National University, Seoul, Korea in February 1996 and 1998. He worked at Daewoo Electronics Company located in Seoul between 1998 and 2003. While working at Daewoo, he mainly participated in various embedded computing system projects. He was a Ph.D. student in the Department of Electrical and Computer Engineering at Texas A&M University since September 2003, and he received his Ph.D. in May 2007. His research interests are in distributed storage systems and file systems. His address is 378 La Strada Drive #25, San Jose, CA 95123.

The typist for this dissertation was Suk Woo Kang.