An Approach to Virtual Allocation in Storage Systems

SUKWOO KANG and A. L. NARASIMHA REDDY
Texas A&M University

This paper presents *virtual allocation*, a scheme 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. This paper presents the design of virtual allocation and an evaluation of it through benchmarks. To illustrate our approach, we implemented a prototype system on PCs running Linux. We present the results from the prototype implementation and its evaluation.

Categories and Subject Descriptors: D.4.2 [**Operating Systems**]: Storage Management; D.4.8 [**Operating Systems**]: Performance

General Terms: Design, Experimentation, Measurement, Performance

Additional Key Words and Phrases: storage systems, storage allocation, storage management, file systems

1. INTRODUCTION

Storage Area Networks (SANs) and storage virtualization [Huang et al. 2004] 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 [Menon et al. 2003]. 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 computer can only create files on devices it owns.

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

Author's address: 111 Weisenbaker Research Center, Texas A&M University, College Station, TX 77843.; email: {swkang,reddy}@ee.tamu.edu.

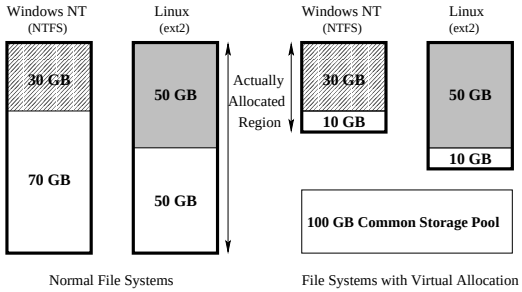


Fig. 1. **Example illustrating virtual allocation.** Virtual allocation lets storage space be allocated based on actual usage. Highlighted region indicates current usage and bold line represents actual allocations. In this case, with virtual allocation, we can get 100 GB (60 GB + 40 GB) unallocated storage space across different operating systems and different file systems.

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 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.

Figure 1 shows two example file systems with and without virtual allocation. In Figure 1, 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. 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.

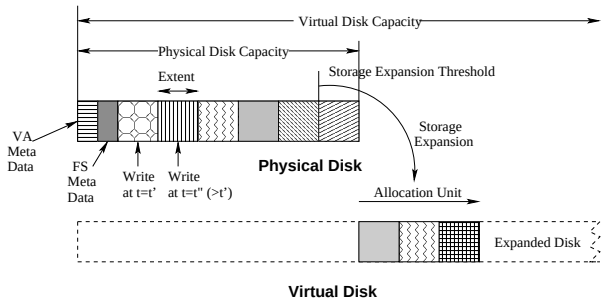


Fig. 2. **Allocation strategy of virtual allocation.** It employs a dynamic allocation strategy based on write-time. When the data is written, storage space is allocated by the unit of the extent.

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 paper 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 [Mesnier et al. 2003; MacCormick et al. 2004].

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.

The rest of the paper is organized as follows. Section 2 presents the design rationale of virtual allocation. Section 3 describes some details about our prototype implementation and Section 4 explains our experimental methodologies and results. Section 5 describes related work and Section 6 points to discussion and future work. Section 7 concludes the paper.

2. DESIGN

2.1 Overview

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. Figure 2 illustrates the storage allocation strategy of virtual allocation. The figure shows an example in which

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

Table I. **Example of a block map in virtual allocation.** It keeps a mapping of logical storage locations (LDEV-ID, LBA) to real (physical) storage locations (PDEV-ID, PBA). It is *dynamically* constructed as data is written.

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 [Rosenblum and Ousterhout 1992; Hitz et al. 1994] 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.[1] 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 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 Figure 2. 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 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 Figure 2. IBM Storage Tank takes an approach of separating metadata and normal data at the file system-level [Menon et al. 2003]. Our allocation policy tends to result in a similar separation of the file system metadata from the file system data.

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

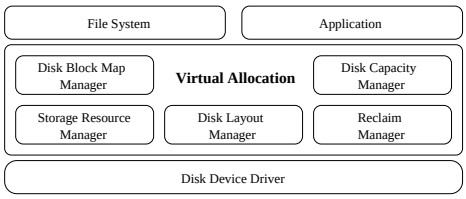


Fig. 3. **Virtual allocation architecture.** It is located between the O/S storage system layer and the disk device driver layer. It intercepts all incoming I/O requests and directs them to the appropriate locations based on the block map.

Virtual allocation makes it easy for file systems to span multiple disks. Figure 2 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 inserted below any file system. Figure 3 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.

Figure 3 shows the components of the virtual allocation architecture. The disk layout manager in Figure 3 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

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. **An example of the extent-based policy in VA with the ext2 file system.** It is a time line of operations when the new data block B is allocated to a file I and written to disk.

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.

2.2 Design Issues

2.2.1 *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 paper, 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[2], but it is expected to improve performance. We consider two file systems on top of VA to illustrate the metadata hardening policies, 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 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

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

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

Table III. **An example of the file system-based policy in VA with the ext3 file system.** It is a time line of operations when the new data blocks B_1 and B_2 are allocated to a file I and I is written to disk.

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.

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 [Sivathanu et al. 2003].[3]

2.2.2 *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.

2.2.3 *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.

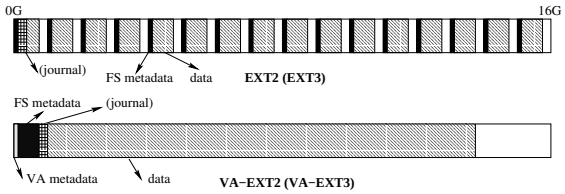
Without reclamation, an application can write large files and delete them to turn virtual allocation into static allocation. Second, as the files are deleted and created through normal activity, if a large amount of block space is allocated while utilizing only a small portion of that block space, the 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 [Sivathanu et al. 2003; Sivathanu et al. 2004].[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.

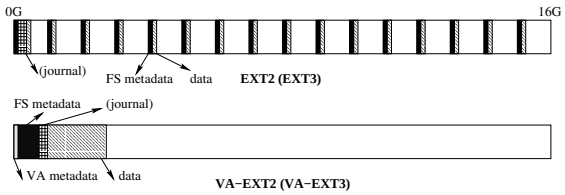
2.2.4 *Chunk Size in RAID.* When our system is used with RAID, VA is layered on top of RAID. For each read or write request, VA first remaps the block address, then RAID software or RAID hardware layer handles this request according to its RAID policy, e.g., RAID-5.

RAID systems do considerable work in choosing the chunk size for optimizing the performance [Patterson et al. 1988]. In this paper, 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 paper, we consider RAID-5 systems and study the impact

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



(a) 85% file system utilization



(b) 10% file system utilization

Fig. 4. **Illustration of the on-disk layout differences between EXT2 (EXT3) and VA-EXT2 (VA-EXT3).** In these examples, the partition size is 16GB. Figure 4(a) shows the on-disk layout at 85% file system utilization and Figure 4(b) shows the on-disk layout at 10% utilization. The only difference of EXT3 from EXT2 is a journal shown in parenthesis.

of the choice of chunk/extent size on the performance of a system employing VA along with RAID.

2.3 Spatial Locality Issues

2.3.1 *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 [McKusick et al. 1984; Ganger and Patt 1994; Wang et al. 1999; Sivathanu et al. 2003] 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 paper, 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. 5. **Illustration of the data placement policy.** It starts allocation from the middle of a partition.

Figure 4 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. Figure 4 depicts the journal stored as a file, which is the default in ext3. Figure 4(a) shows the on-disk layout of a 16GB partition at 85% file system utilization and Figure 4(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.

2.3.2 *Data Clustering.* Due to our storage allocation policy, all written data are clustered in VA, as shown in Figure 4. Data clustering will improve performance of VA-EXT2 compared to EXT2 by 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 Figure 4(b). As the file system utilization gets higher, the benefit of clustering in VA is likely to decrease.

2.3.3 *Space Allocation Policy.* Figure 4 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 [Meter 1997]. 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 Figure 5 and wraparound. This may reduce average seek distances to metadata by placing metadata in the middle of the disk.

2.3.4 *File System Aging (Fragmentation).* Earlier file systems have used clustering to improve performance [Peacock 1988; McVoy and Kleiman 1991]. 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 [Smith and Seltzer 1996].

2.3.5 *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

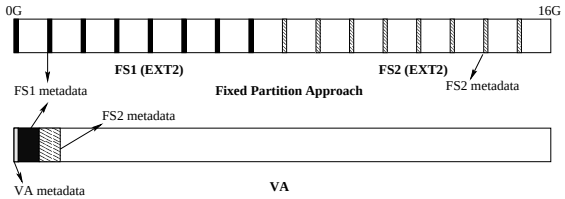


Fig. 6. **The on-disk layout difference between VA and the fixed partition approach.** In this example, two file systems are created; FS1 is created first and then FS2 is created.

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.

Figure 6 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 Figure 6, 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.

2.4 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.

3. 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.

4. 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 three workloads in our experiments. The first workload was sequential reads and writes of large files of Bonnie++ benchmark [Coker 2001]. 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 [Katcher 1997]. 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 [Vogels 1999]. 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) [Levine 1997]. 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 [Shaw 2002].

4.1 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.

4.2 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 [Ellard and Seltzer 2003]. 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 benchmark [Coker 2000].

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 ad-

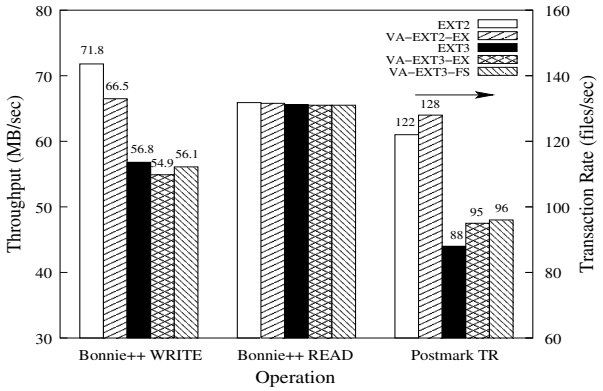


Fig. 7. **Impact of VA metadata hardening on performance of VA for the large-file workload (Bonnie++) and the small-file workload (Postmark)**. In configurations, *-EX represents VA with the extent-based hardening and *-FS denotes VA with the file system-based hardening.

dresses by employing a Postmark benchmark.[4] 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.

4.3 Impact of Various Factors

4.3.1 *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.

Figure 7 shows the results of both benchmarks. The figure depicts the write and the read throughputs of Bonnie++ and the transaction rate of Postmark under different configurations. Each group of 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 7.3% in write operations and incurred no measureable overhead in read operations compared to EXT2. Similarly, VA-EXT3 with the extent-based hardening incurred an overhead of 3.3% in write operations and in-

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

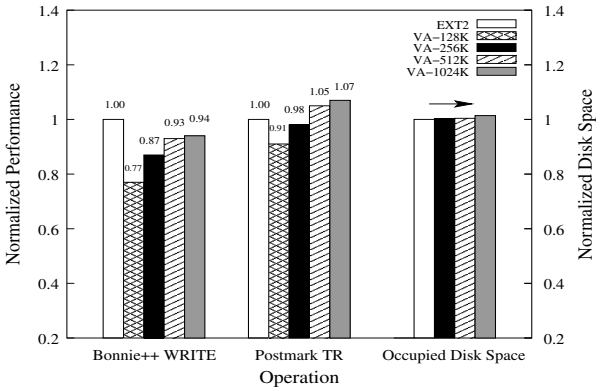


Fig. 8. **Impact of extent sizes on performance of VA.** The third group of bars shows a normalized disk space to the occupied disk space of the 128KB extent for each Postmark experiment.

curred no measurable overhead in read operations compared to EXT3. VA-EXT3 with the file system-based hardening reduced the overhead of extent-based hardening in write operations by 2.1%, which caused overall 1.2% overhead compared to EXT3. This reduction is due to the less frequent VA metadata hardening (due to aggregation), which was explained in the previous section. 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.

In Postmark results (the third group of bars in Figure 7), we observed that VA-EXT2 with the extent-based hardening showed a 4.9% performance increase in the transaction rate and VA-EXT3 with the extent-based hardening showed a 8.4% performance increase. VA-EXT3 with the file system-based hardening further increased performance up to 9.5%. 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.

4.3.2 *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. Figure 8 shows the impact of the extent size in both workloads. The first group of bars shows the write throughput of

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. **Reclaim operation of periodic purges.** This table shows the number of live blocks of EXT2 and corresponding number of live extents of VA-EXT2 for each simulation phase.

Bonnie++[5] and the second group of bars denotes the transaction rate of Postmark, each of which is normalized to that of EXT2. 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 256KB or larger incurred no overhead or performed better than EXT2. The nature of I/Os, large (Bonnie++) versus small (Postmark) contributed to these differences.[6]

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. The third group of bars in Figure 8 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 noted.

4.3.3 *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 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

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

[6]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.

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)

Table V. **Reclaim operation of usual file system activity.** This table shows the number of live blocks and touched blocks of EXT2 and the corresponding number of live extents of VA-EXT2 for each simulation phase.

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 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

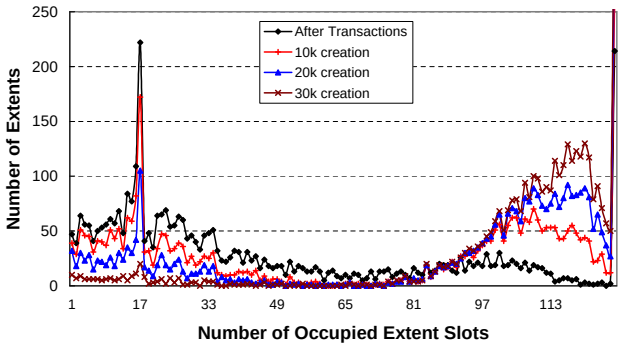


Fig. 9. **The filled status of extents.** Each value represents a number of extents each of which has the corresponding number of (file system) live blocks. In this experiment, an extent (512KB) is fully filled if it has 128 (file system) live blocks (4KB).

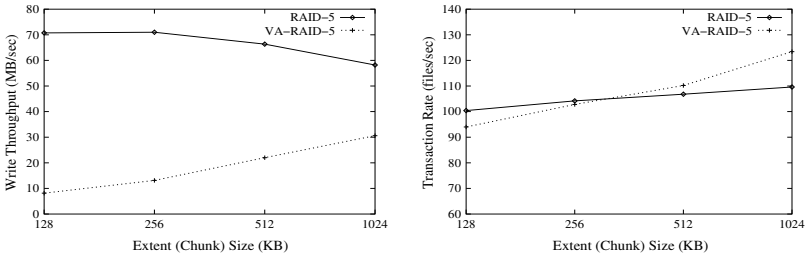
increasing the number of touched blocks, which means that EXT2 efficiently reused previously allocated blocks. VA-EXT2 allocated 90 new extents (11,520 blocks) and reused partially live extents for other blocks. These results indicate that the file system efficiently reuses blocks so that partially live 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. Figure 9 presents the extent statistics after doing transactions (§4) to after creating 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.

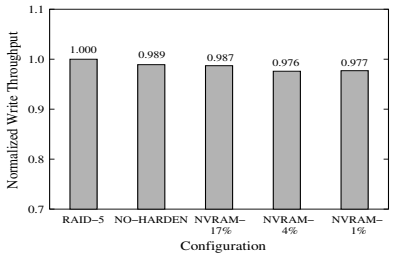
4.3.4 *RAID.* We measured the performance of VA with RAID using the same benchmarks and configurations as the extent size experiment. Figure 10(a) shows the write throughput of RAID-5 and VA-RAID-5 systems.[7] As the extent size increases, the write throughput of VA-RAID-5 increases, but even with the 1MB extent size, the write throughput corresponds to only 52.5% of RAID-5. This overhead is due to the expensive small-write cost of VA metadata hardening.

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. Figure 10(c) 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

[7]RAID-5 and VA-RAID-5 systems showed nearly the same read throughput, which was not shown due to space constraints.



(a) Bonnie++ (b) Postmark



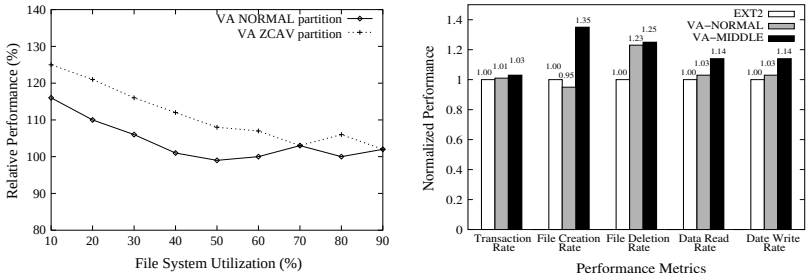
(c) Impact of NVRAM

Fig. 10. **Performance of VA-RAID-5.** In figure 10(c), NO-HARDEN denotes the configuration where VA metadata hardening is turned off. NVRAM-* represents the configuration using NVRAM. The percentage is the ratio of dedicated NVRAM size to total amount of VA metadata.

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.

Figure 10(b) 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.

4.3.5 *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 par-*



(a) Space allocation policy 1 (b) Space allocation policy 2

Fig. 11. **Impact of space allocation policies.** In Figure 11(b), VA-MIDDLE represents a configuration where allocation starts from the middle of the partition.

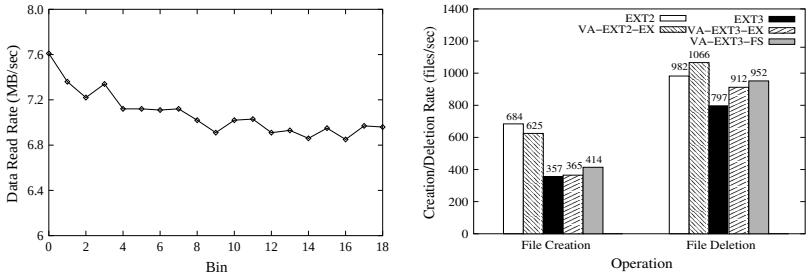
tition). The data rate difference of the location of 0GB and 16GB of each partition was 2.4% and 14.0% on the two disks (due to ZCAV effects).

The graph in Figure 11(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 Figure 5. 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. Figure 11(b) shows various performance metrics of the two policies. As shown in Figure 11(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.

4.3.6 *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.[8] The data read rate for files in each bin is reported in Figure 12(a).

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



(a) Metadata and data separation (b) Metadata clustering

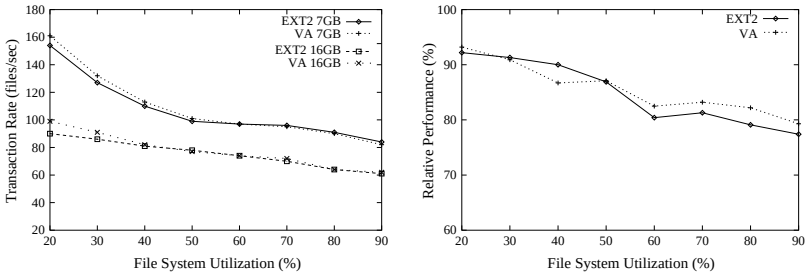
Fig. 12. **Impacts of metadata and data separation and metadata clustering.** In Figure 12(a), the (VA and FS) metadata cluster is located before bin 0.

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*.

4.3.7 *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. Figure 12(b) 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.

4.3.8 *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.

Figure 13(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.



(a) File system utilization (b) File system aging

Fig. 13. **Impacts of file system utilization and aging.** Figure 13(b) shows performance degradation of VA and EXT2 as file system ages.

Therefore, the two systems show similar performance at larger utilizations. The performance results under the partition size of 7GB show similar trends.

4.3.9 *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 [Smith and Seltzer 1996].

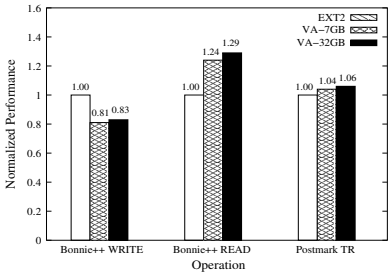
First, it creates a number of directories and populates them with files up to 10% file system utilization. It measures the transaction rate the same way as is done in Postmark. Then it populates more files to make a file system usage of 20% and measures the transaction rate again. It repeats these procedures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

The graph in Figure 13(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 Figure 13(b), as the file system ages, VA-EXT2 incurs a performance impact similar to that of EXT2 at different file system utilizations.

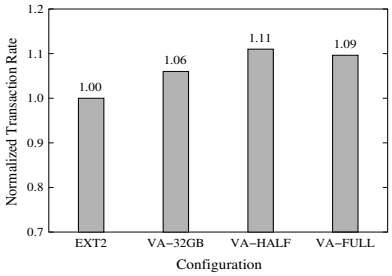
4.3.10 *Multiple File Systems.* Next, we considered the use of multiple file systems with virtual allocation. Specifically, We compared the fixed partition approach with our approach by performing two experiments. In the first experiment, we used two configurations to see the impacts of disk arm movements by employing two benchmarks (Bonnie++ and Postmark). First, we created two 3.5GB partitions on the 147GB disk and created an ext2 file system[9] 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.

Figure 14(a) shows the average throughput of two file systems from Bonnie++

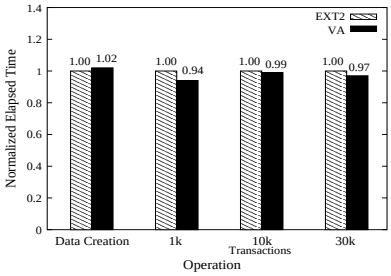
[9]We performed the same experiments using an ext3 file system with various journaling modes and got similar results to those of EXT2, which were not shown due to space constraints.



(a) Multiple file systems 1



(b) Multiple file systems 2



(c) TPC-C workload

Fig. 14. **Performance of multiple file systems with VA and database workload.** Figure 14(a) and 14(b) show performance results of multiple file systems with VA. Figure 14(c) shows database workload performance of VA.

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 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.

Figure 14(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.

4.3.11 *Results for the Other Workloads.* Figure 14(c) 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.

5. 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).

File system-level: IBM's Storage Tank [Menon et al. 2003] 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 [Rosenblum and Ousterhout 1992] and WAFL [Hitz et al. 1994]) 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 [Reiser 2004] and JFS (Journaled File System) [Best 2000] support expansion of file systems. None of these systems allocates storage space based on the actual usage.

Block-level: Veritas Volume Manager [Veritas 2002] and other existing Storage Resource Management (SRM) products such as IBM Tivoli Storage Manager [Kaczmarski et al. 2003] provide considerable flexibility in storage management, but allocate storage space based on the file system size. Loge [English and Stepanov 1992] 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 paper 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 [Wilkes et al. 1996]. Petal [Lee and Thekkath 1996] 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 [3PARdata 2003], which provides storage allocation flexibility as our system does. However, the details about their architecture, implementation and performance are unknown.

More expressive interfaces (between the file system and the storage system): Object-based storage proposed by CMU [Mesnier et al. 2003] allows storage allocation to be done at the storage systems as opposed to a file system manager. Logical disk [de Jonge et al. 1993] defined a new interface to disk storage that separates file management and disk management. Boxwood [MacCormick et al. 2004] 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).

Other system environments: VMWare's virtual machine [VMware 2000] 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 [Kubiatowicz et al. 2000; Adya et al. 2002; Cox et al. 2002] 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.

6. DISCUSSION & FUTURE WORK

In this paper, we employed *gray-box* approach proposed by Wisconsin [Sivathanu et al. 2003; Sivathanu et al. 2004] for the file system-based hardening and the reclaim operation. As an alternative, we could use stackable file system [Zadok et al. 2006]. 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.

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 [Anderson et al. 2000]. 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 [Krueger et al. 2002]). 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 [Ng et al. 2002; He et al. 2002]. We plan to pursue this topic in the future. We will also 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.

7. CONCLUSION

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 available 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.

REFERENCES

3PARDATA. 2003. Thin provisioning. *http://www.3pardata.com/products/thinprovisioning.html*.

ADYA, A., BOLOSKY, W. J., CASTRO, M., CERMAK, G., CHAIKEN, R., DOUCEUR, J. R., HOWELL, J., LORCH, J. R., THEIMER, M., AND WATTENHOFER, R. 2002. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the 5th Symposium on Operating System Design and Implementation*.

ANDERSON, D. C., CHASE, J. S., AND VAHDAT, A. 2000. Interposed request routing for scalable network storage. In *Proceedings of the 4th Symposium on Operating System Design and Implementation*. 259–272.

BEST, S. 2000. Jfs overview. *http://www-106.ibm.com/developerworks/library/1-jfs.html*.

COKER, R. 2000. Zcav: a hard drive testing program. *http://www.coker.com.au/bonnie++/zcav*.

COKER, R. 2001. Bonnie++: A benchmark suite of hard drive and file system performance. *http://www.coker.com.au/bonnie++*.

COX, L. P., MURRAY, C. D., AND NOBLE, B. D. 2002. Pastiche: Making backup cheap and easy. In *Proceedings of the 5th Symposium on Operating System Design and Implementation*.

DE JONGE, W., KAASHOEK, M. F., AND HSIEH, W. C. 1993. The logical disk: A new approach to improving file systems. In *Proceedings of the Fourteenth ACM Symposium on Operating System Principles*. 15–28.

ELLARD, D. AND SELTZER, M. I. 2003. Nfs tricks and benchmarking traps. In *Proceedings of the USENIX Annual Technical Conference, FREENIX Track*. 101–114.

ENGLISH, R. M. AND STEPANOV, A. A. 1992. Boxwood: Abstractions as the foundation for storage infrastructure. In *Proceedings of the USENIX Winter 1992 Technical Conference*. 237–251.

GANGER, G. R. AND PATT, Y. N. 1994. Metadata update performance in file systems. In *Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation*. 49–60.

HE, X., YANG, Q., AND ZHANG, M. 2002. A caching strategy to improve iscsi performance. In *Proceedings of the 27th Annual IEEE Conference on Local Computer Networks*. 278–288.

HITZ, D., LAU, J., AND MALCOLM, M. A. 1994. File system design for an nfs file server appliance. In *Proceedings of the USENIX Winter 1994 Technical Conference*. 235–246.

HUANG, L., PENG, G., AND CKER CHIUEH, T. 2004. Multi-dimensional storage virtualization. In *Proceedings of the International Conference on Measurements and Modeling of Computer Systems*. ACM, New York, 14–24.

KACZMARSKI, M., JIANG, T., AND PEASE, D. A. 2003. Beyond backup toward storage management. *IBM Systems Journal 42,* 2, 322–337.

KATCHER, J. 1997. Postmark: a new filesystem benchmark. *http://www.netapp.com/tech_library/3022.html*.

KRUEGER, M., HAAGENS, R., SAPUNTZAKIS, C., AND BAKKE, M. 2002. Rfc 3347: Small computer systems interface protocol over the internet (iscsi) requirements and design considerations.

KUBIATOWICZ, J., BINDEL, D., CHEN, Y., CZERWINSKI, S. E., EATON, P. R., GEELS, D., GUMMADI, R., RHEA, S. C., WEATHERSPOON, H., WEIMER, W., WELLS, C., AND ZHAO, B. Y. 2000. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*. 190–201.

LEE, E. K. AND THEKKATH, C. A. 1996. Petal: Distributed virtual disks. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*. 84–92.

LEVINE, C. 1997. Tpc-c: The oltp benchmark. In *Proceedings ACM SIGMOD International Conference on Management of Data (Industrial Session 5)*.

MACCORMICK, J., MURPHY, N., NAJORK, M., THEKKATH, C. A., AND ZHOU, L. 2004. Boxwood: Abstractions as the foundation for storage infrastructure. In *Proceedings of the 6th Symposium on Operating System Design and Implementation*. 105–120.

McKusick, M. K., Joy, W. N., Leffler, S. J., and Fabry, R. S. 1984. A fast file system for unix. *ACM Trans. Comput. Syst. 2,* 3, 181–197.

McVoy, L. W. and Kleiman, S. R. 1991. Extent-like performance from a unix file system. In *Proceedings of the USENIX Winter Conference*. 33–44.

Menon, J., Pease, D. A., Rees, R. M., Duyanovich, L., and Hillsberg, B. 2003. Ibm storage tank - a heterogeneous scalable san file system. *IBM Systems Journal 42,* 2, 250–267.

Mesnier, M., Ganger, G. R., and Riedel, E. 2003. Object-based storage. *IEEE Communications Magazine 41,* 8, 84–90.

Meter, R. V. 1997. Observing the effects of multi-zone disks. In *Proceedings of the USENIX Annual Technical Conference*.

Ng, W. T., Hillyer, B., Shriver, E. A. M., Gabber, E., and Özden, B. 2002. Obtaining high performance for storage outsourcing. In *Proceedings of the FAST '02 Conference on File and Storage Technologies*. 145–158.

Patterson, D. A., Gibson, G. A., and Katz, R. H. 1988. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*. 109–116.

Peacock, J. K. 1988. The counterpoint fast file system. In *Proceedings of the USENIX Winter Conference*. 243–249.

Reiser, H. 2004. Reiserfs: Journaling file system for linux based on balance tree algorithms. *http://www.namesys.com*.

Rosenblum, M. and Ousterhout, J. K. 1992. The design and implementation of a log-structured file system. *ACM Trans. Comput. Syst. 10,* 1 (Feb.), 26–52.

Shaw, S. 2002. Hammerora: The open source oracle load test tool. *http://hammerora.sourceforge.net*.

Sivathanu, M., Bairavasundaram, L. N., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. 2004. Life or death at block-level. In *Proceedings of the 6th Symposium on Operating System Design and Implementation*. 379–394.

Sivathanu, M., Prabhakaran, V., Popovici, F. I., Denehy, T. E., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. 2003. Semantically-smart disk systems. In *Proceedings of the FAST '03 Conference on File and Storage Technologies*.

Smith, K. A. and Seltzer, M. I. 1996. A comparison of ffs disk allocation policies. In *Proceedings of the USENIX Annual Technical Conference*. 15–26.

Veritas. 2002. Volume manager for windows with mscs. *http://www.veritas.com/van/products/volumemanagerwin.html*.

VMware. 2000. Vmware workstation. *http://www.itc.virginia.edu/atg/techtalks/powerpoint/vmware/sld018.htm*.

Vogels, W. 1999. File system usage in windows nt 4.0. In *Proceedings of the 17th ACM Symposium on Operating System Principles*. 93–109.

Wang, R. Y., Anderson, T. E., and Patterson, D. A. 1999. Virtual log based file systems for a programmable disk. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation*. 29–43.

Wilkes, J., Golding, R. A., Staelin, C., and Sullivan, T. 1996. The hp autoraid hierarchical storage system. *ACM Trans. Comput. Syst. 14,* 1, 108–136.

Zadok, E., Iyer, R., Joukov, N., Sivathanu, G., and Wright, C. P. 2006. On incremental file system development. *ACM Transactions on Storage (TOS) 2,* 3. To appear.