Improving Storage System Flexibility Through Virtual Allocation

Sukwoo Kang, and A. L. Narasimha Reddy
Dept. of Electrical Engineering
Texas A & M University
College Station, Texas, 77843
{*swkang, reddy*}*@ee.tamu.edu*

Abstract

This paper presents *virtual allocation*, a scheme for flexible storage allocation. It 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.

1 Introduction

Storage Area Networks (SANs) and storage virtualization [1] 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 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 [2].

Traditional file systems are closely tied to the underlying storage. Storage space is allocated and owned at the time the file system is created. This directly conflicts with the notion of storage as a discoverable resource since unused space in one file system cannot be used easily by another file system running out of space. The file systems currently employ a static allocation approach where entire storage space is claimed at the time of the file system creation. This is somewhat akin to running with only physical memory in a memory system. Memory systems employ virtual memory for many reasons: to allow flexible allocation of resources, safe sharing, to allow larger programs to run, etc. Traditional file systems lack 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.

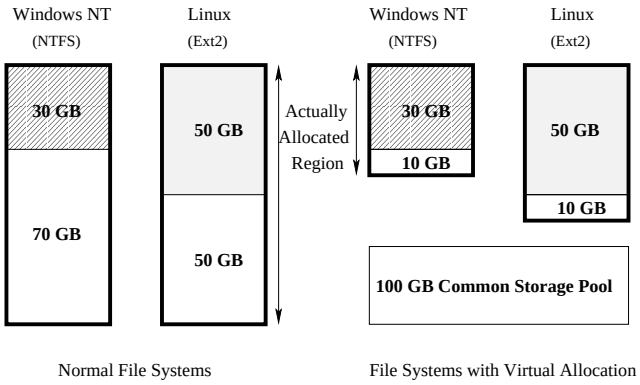


Figure 1: *Example illustrating virtual allocation. Virtual allocation lets storage space be allocated based on actual usage. Highlighted region indicates current usage and bold line represents actual allocations. In this case, with virtual allocation, we can get 100 GB (60 GB + 40 GB) unallocated storage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

A file system can be created with X GBs. Instead of allocating the entire X GBs of space, virtual allocation allows the file system to be created with only Y GBs, where Y could be much smaller than X. The remaining storage space $(Y - X)$ GBs will be unused and available as a discoverable resource. If the created file system does not exceed the actual allocation of Y GBs, the remaining space $(Y - X)$ GBs can be used by another file system, or application. 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

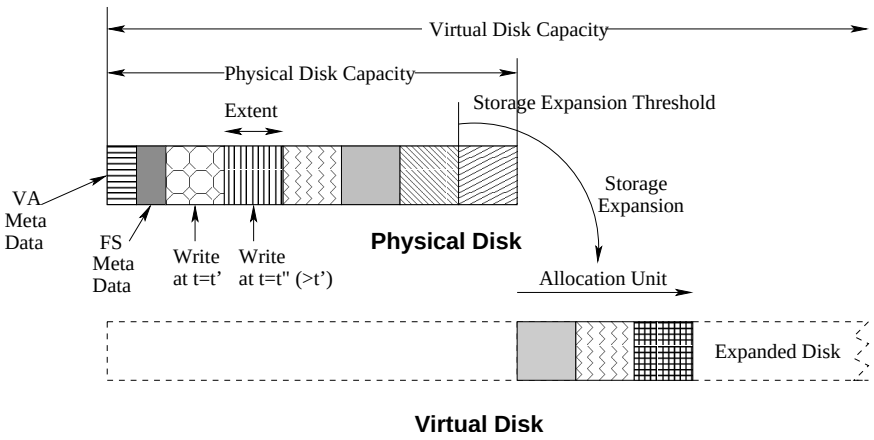


Figure 2: *Allocation strategy of virtual allocation. It employs a dynamic allocation strategy based on write-time. When the data is written, storage space is allocated by the unit of the extent.*

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.

Many of the ideas presented in this paper were pursued in different forms earlier. IBM's Storage Tank [2] separates metadata operations from data operations to detach storage allocation from file systems. IBM's Storage Tank allows a common pool of storage space to be shared across heterogeneous environments. Object based storage proposed by CMU [3] allows storage allocation to be done at the storage systems as opposed to a file system manager. Log-structured file systems (for example, LFS [7] and WAFL [8]) 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 [4] and JFS (Journaled File System) [5] support expansion of file systems. These approaches are either at the file system-level or they require new storage paradigms. In contrast, virtual allocation is a device-level approach and does not require installing new file systems. Veritas Volume Manager [12] and other existing Storage Resource Management (SRM) products such as IBM Tivoli Storage Manager [6] provide considerable flexibility in storage management, but allocate storage space based on the file system size.

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 for virtual allocation and Section 3 describes some details about our prototype implementation. Section 4 explains our experimental methodologies and results. Section 5 points to the discussions and future work and Section 6 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 *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. First, virtual allocation writes all data to disk sequentially in a

LBA	Extent Number	Device Number	PBA
100	0	Dev0	1000
500	1	Dev0	1100
1000	2	Dev0	1200
5000	3	Dev1	1000

Table 1: *Example of a block map in virtual allocation. It keeps a mapping of logical storage locations to real (physical) storage locations. It is* ***dynamically*** *constructed as data is written. LBA denotes the logical block address and PBA represents the physical block address.*

log-like structure based on the time at which data is written to the device. The figure shows an example that new data is written at time t = t′ and at time t = t′′ where t′′ > t′. This approach is similar to log-structured file systems [7, 8] 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. Once data is written to the 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 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 the 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 block address (LBA), (mapped) extent number, the physical device where the extent resides, and the physical block address of the extent (PBA). Table 1 shows an example of block map data structure. For example, the first row of Table 1 means that a *logical block 100* resides at the *extent 0* whose physical block address is *1000* of block device *Dev0*.

Each entry of the block map corresponds to one extent and whenever a new extent is written, 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 128 KB extent, the size of VA metadata is less than 96 KB per 1 GB disk space.

File systems tend to create metadata at the time of file system creation. Due to our allocation policy, file system metadata tends to be clustered in front of the allocation log.

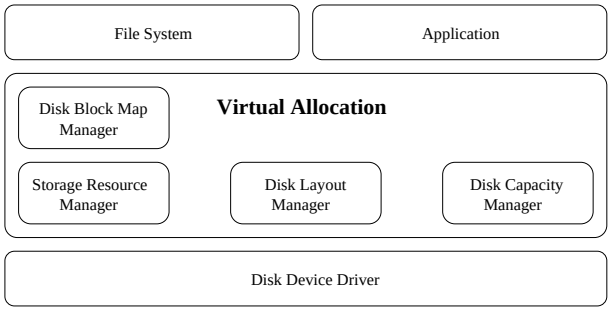


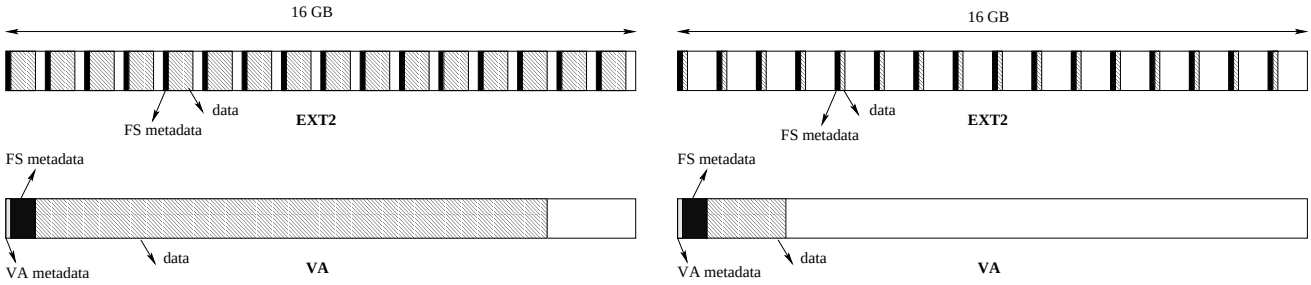
Figure 3: *Virtual allocation architecture. It is located between the O/S storage system layer and the disk device driver layer. It intercepts all incoming I/O requests and routes them to the appropriate locations based on the block map.*

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. Our allocation policy tends to result in a similar separation of the file system metadata from the file system data.

Figure 2 also shows that when the capacity of physical disk is exhausted or reaches a certain limit (*storage expansion threshold*), a physical disk can be expanded to other available storage resources. This process of *storage expansion* allows file systems or applications to potentially span multiple devices. Storage expansion is done in terms of *allocation units*. For example, if the allocation unit is 1GB, whenever storage expansion is performed, virtual allocation adds 1GB of disk space to an existing disk by finding other available storage resources.

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 a large storage device 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 thin 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 routes 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



(a) 85% file system utilization

(b) 10% file system utilization

Figure 4: *Illustration of the on-disk layout differences between EXT2 and VA. 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.*

map is consulted for mapping information. If it exists, it is routed 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 virtual allocation architecture. Disk layout manager in Figure 3 routes incoming read and write requests to appropriate locations and disk block map manager maintains this information as a block map. Disk capacity manager continuously monitors disk usage and it generates a service message to a storage resource manager once the disk usage reaches a 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.

2.2 Design Issues

VA Metadata Hardening. As mentioned earlier, our in-memory block map must be hardened to disk regularly to protect against system failures. Since VA metadata hardening requires an additional disk access, we have to carefully decide how often VA metadata will be written to the disk to minimize the overhead. If the file system has a relaxed mechanism of hardening file system (FS) metadata, more strict VA metadata hardening than that of the file system is not necessary because a FS metadata update failure will leave the file system metadata more inconsistent than VA metadata.

For example, a Linux Ext2 file system, by default, does not use synchronous writes for metadata updates. When virtual allocation is used with such a file system, it doesn't need to harden VA metadata more often than the Ext2 file system does. On the other hand, the Berkeley FFS (also known as UFS), by default, uses synchronous writes for metadata updates. Hence, VA metadata should be updated frequently enough so as not to compromise the FS metadata integrity.

In this paper, we considered two kinds of VA metadata hardening policies: a *file system-based policy* and an *extent-based policy*. A file system-based VA metadata hardening treats VA metadata similar to how the file system treats FS metadata. An extent-based VA metadata hardening provides the most strict policy i.e., VA metadata is written to the disk whenever a new extent is allocated.

Extent Size. An extent size is a configurable parameter in our system. Larger extents can retain more spatial locality than smaller ones. They can 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 tradeoffs with the size of an extent are similar to the tradeoffs with the page size in virtual memory or the block size in cache memory systems.

2.3 Spatial Locality Issues

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. The file systems put significant effort into keep-

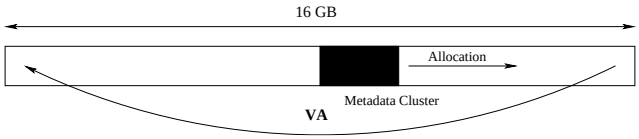


Figure 5: *Illustration of the data placement policy which places hot data (metadata cluster of VA) in the middle of a partition to reduce the average seek distance to hot data. It starts allocation from the middle of a partition.*

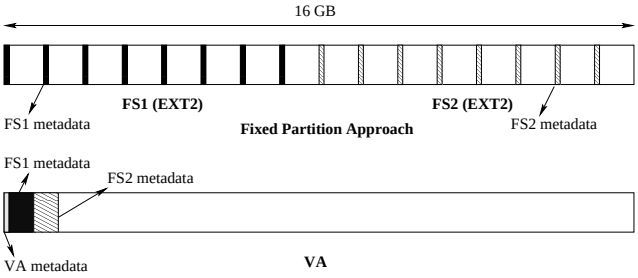


Figure 6: *The on-disk layout difference between VA and the fixed partition approach after two file systems are created. In this example, FS1 was created first and then FS2 was created.*

ing metadata in close proximity to the data. Earlier studies [21, 22, 23, 24] 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 two systems: an Ext2 file system with virtual allocation and an unmodified Ext2 file system with a normal storage system. In the remaining sections, **VA** denotes a configuration of an Ext2 file system with virtual allocation and **EXT2** represents an Ext2 file system with a normal storage system.

Figure 4 shows differences of the on-disk layout between EXT2 and VA. 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. File systems such as Linux Ext2 and BSD FFS try to distribute metadata all over the disk in order to keep the metadata in close proximity to the data. However, 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.

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 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 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 reduce.

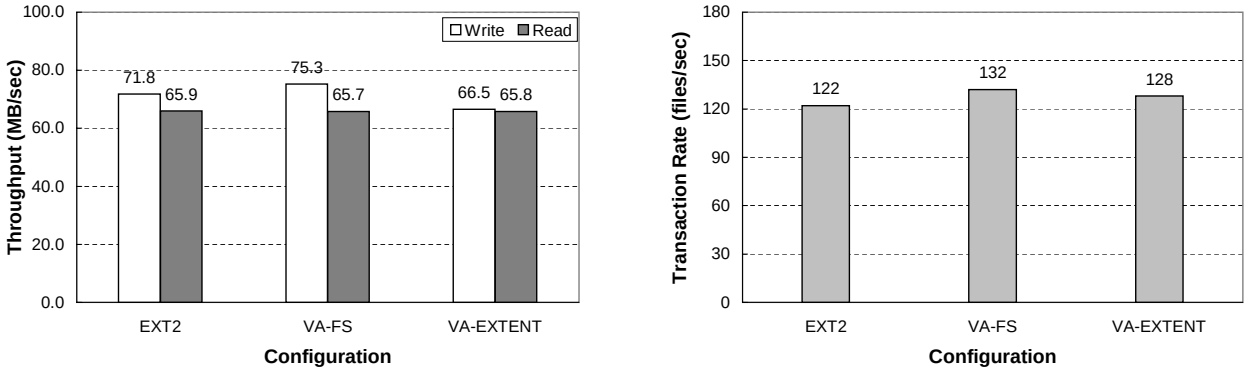
Space Allocation Policy. Figure 4 shows a linear allocation of space on the device. It is possible to employ many different policies for space allocation. For example, VA can allocate hot data (such as the metadata cluster) in a high data rate region of a disk to increase performance. Modern disk drives use a technique known as *zoned constant angular velocity* (ZCAV) to increase total disk capacity by varying the number of disk sectors per track with the distance from the spindle. A side effect of this technique is that the transfer rate also varies with sector address [16] i.e., data rate is higher in outer cylinders than in inner cylinders. By placing the metadata cluster in a high data rate region of a disk (outer cylinder), we may be able to improve the file system performance.

Even if the partition is located in the region of a disk which has the constant data rate, we can still improve performance by placing the metadata cluster in the middle of the partition. This will reduce the average seek distance to metadata and will contribute to increased performance. This is illustrated in Figure 5.

File System Aging (Fragmentation). Earlier file systems have used clustering to improve performance [9, 10]. 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 [11].

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



(a) Bonnie++ results

(b) Postmark results

Figure 7: *Impact of VA metadata hardening on performance of VA for the large-file workload and the small-file workload. Bonnie++ benchmark was used as the large-file workload and Postmark was used as the small-file workload.*

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 three 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 EXT2. The third factor is related to space allocation policy i.e., in VA, data observe different data rates compared to a normal storage system. These three 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. The virtual allocation layered driver resides between the O/S storage system and the disk device driver interface. It intercepts all incoming I/O requests and routes them dynamically based on the block map. The in-memory block map was implemented as a hash table.

4 Evaluation

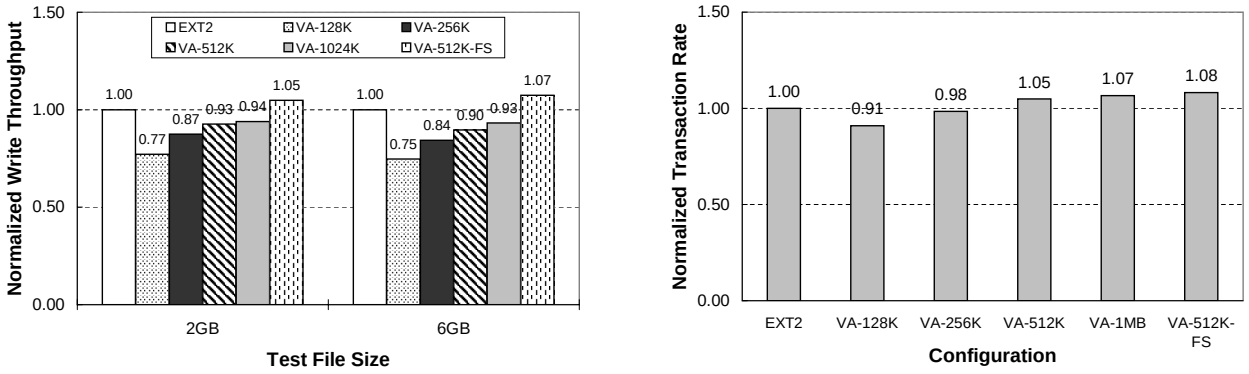
In this section, we compare two systems: EXT2 and VA. We used three workloads in our experiments. The first workload was sequential reads and writes of large files of Bonnie++ benchmark [15]. We used test files of various sizes depending on the experiment and used a 4KB chunk size. It sequentially writes a test file to a disk and then sequentially reads it by the unit of the chunk size.

The second workload we chose was Postmark [20]. 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 [18]. 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) [13]. 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 [14].

4.1 Experimental Setup

All experiments were performed on commodity PC systems equipped with a 3GHz Pentium 4 processor, 900MB of main memory, and two 10,000 RPM Seagate SCSI disks (ST3146807LW: 147GB, ST336607LW: 37GB) controlled by the Adaptec SCSI Card 29160. The operating system was Red Hat Linux 9 with a 2.4.22 kernel and the file system was an Ext2 file system. All experiments were run on



(a) Bonnie++ results

(b) Postmark results

Figure 8: *Impact of extent sizes on performance of VA for the large-file workload and the small-file workload. Bonnie++ benchmark was used as the large-file workload and Postmark was used as the small-file workload.*

an empty file system except the file system aging experiment and the disk of 37GB size was only used in the space allocation policy experiment. 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 [17] to remove a device-specific feature. To reduce ZCAV effects, we used a 7GB partition (using outer cylinders) of 147GB disk in most experiments. There was less than 0.2% performance variation due to ZCAV effects 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 [19]. If a partition of more than 7GB size was used, we presented performance results considering the bias of ZCAV effects.

4.3 Impact of Various Factors

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 with two VA metadata hardening policies, i.e., a file system-based hardening, and an extent-based hardening. 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(a) shows the results of the Bonnie++ benchmark. The figure depicts the write and the read throughputs under different configurations. Each group of bars shows the throughput for a particular configuration. The leftmost group shows the throughput of EXT2 for reference. VA-FS denotes VA with the file system-based hardening and VA-EXTENT represents VA with the extent-based hardening.

VA with the file system-based hardening increased performance by 4.8% compared to EXT2. This improvement is due to data clustering of VA, which was explained in the previous section. VA with the extent-based hardening incurred an overhead of 7.3% in write operations and incurred no overhead in read operations compared to EXT2. This overhead is due to more frequent VA metadata hardening than the hardening of metadata in the file system.

Figure 7(b) shows the transaction rate for Postmark. This figure has the same structure as Figure 7(a) and shows results with the same configurations for Postmark workloads. The figure shows that VA with the file system-based hardening improved the transaction rate by 8.2%. The extent-based hardening showed a 4.9% performance increase in the transaction rate.

These results indicate that the impact of VA metadata hardening was larger in a large-file workload. We used, by default, the extent-based hardening in all the following experiments to provide the base performance. When a comparison is needed, we also presented the results of the file system-based hardening.

Extent Size

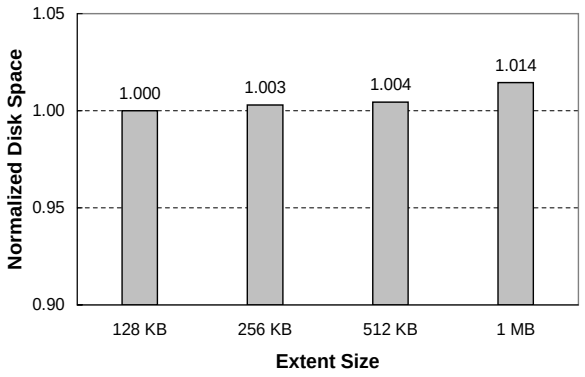


Figure 9: *Occupied disk space for different extent sizes. Each value shows a normalized disk space to the occupied disk space of 128KB extent.*

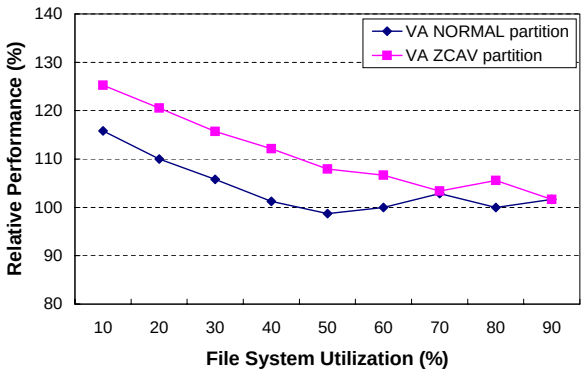


Figure 10: *Comparison of VA performance for two data placement configurations. VA NORMAL partition represents the configuration of VA on a partition which has the same data rate across the partition. VA ZCAV partition denotes the configuration of VA on a partition which has different data rates across the partition. Each value is a a percentage of the transaction rate of the corresponding EXT2.*

We measured the impact of extent sizes using Bonnie++ and Postmark. Four different extent sizes were used in this experiment i.e., 128KB, 256KB, 512KB, and 1MB. In Bonnie++ benchmark, we used test files of size 2GB and 6GB. Postmark was configured to have the same configuration as the previous experiment. We compared each performance result of VA to that of EXT2. We expected that overall VA performance would be better as the extent size got larger because larger extent size can reduce the VA metadata hardening overhead and can retain more spatial locality.

Figure 8(a) shows the impact of the extent size in a large-file workload. The bar height represents the write throughput normalized to that of EXT2. Each group of bars shows throughputs according to different test file sizes. Each group consists of six bars: beginning at the far left, the first bar shows the throughput of EXT2, and the next four bars denote the throughput of VA with different extent sizes. The last bar shows the VA throughput of 512KB extent with the file system-based hardening. All other VA data were obtained with the extent-based hardening. The read throughput of VA showed less than 1% degradation compared to EXT2 for all extent sizes, which were not shown due to the space limitations.

Larger extent size shows better performance in VA. For 2GB test file size, the throughput of 512KB extent size is higher than that of 256KB extent by 6%. VA shows worse performance than EXT2 except the VA with the file system-based hardening. VA with 512KB extent incurred 7% overhead compared to EXT2 and 1MB extent incurred 6% overhead. However, with the file system-based hardening, we got a performance improvement of 5%. The performance results from other test file sizes show similar trends.

Figure 8(b) shows the impact of the extent size in a small-file workload. The bar height represents the transaction rate normalized to that of EXT2. As we expected, the transaction rate improves as the extent size gets larger. The extent size of 128KB resulted in 9% performance degradation. VA with extent sizes of 256KB or higher performed nearly as well or better than EXT2.

Large extent size can increase overall performance of VA, 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 in Figure 9. In the case of 512KB extent size, only 0.4% disk space overhead was observed compared to that of 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 and an extent-based VA metadata hardening unless specifically noted.

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 147GB disk (*VA NORMAL partition*) and VA on a 16GB partition of 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.

The graphs in Figure 10 show 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 in-

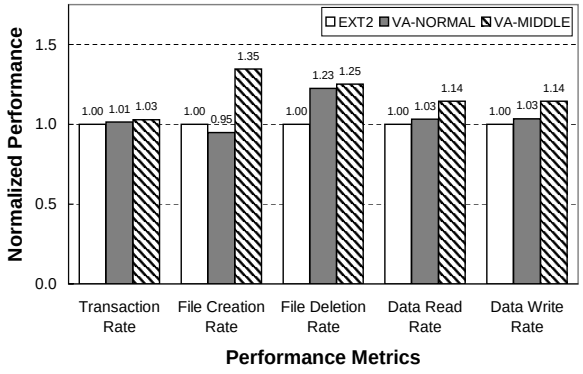


Figure 11: *Comparison of VA performance between two data placement policies, each of which places hot data (metadata cluster) in different locations of the disk. VA-NORMAL starts allocation from the beginning of a partition, so the metadata cluster is allocated in front of the partition. In contrast, VA-MIDDLE starts allocation from the middle of the partition, the metadata cluster is allocated in the middle of the partition.*

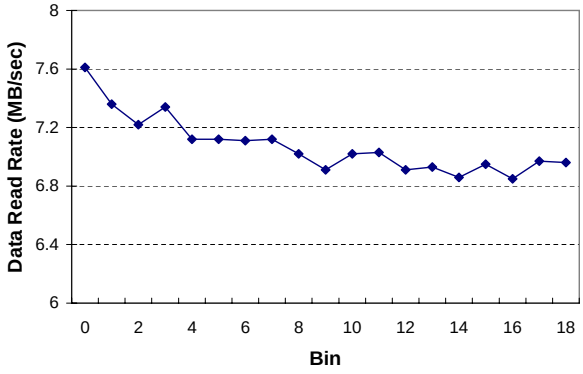


Figure 12: *Data read rate as the data get farther from the metadata cluster located before bin 0. Each value represents the data read rate of each bin.*

creases, the performance improvement decreases for both cases. However, VA on a ZCAV partition always showed better performance improvement than VA on a NORMAL partition.

The second experiment was done in the NORMAL partition which has almost the same data rate across the partition. We did experiments with two data placement policies. One policy starts allocation from the beginning of the partition and the other one starts allocation from the middle of the partition as shown in Figure 5. The latter policy will place the metadata cluster in the middle of the partition because FS metadata is created at the time of file system creation. Figure 11 shows various performance metrics of the two policies. VA-NORMAL denotes the former data placement policy, and VA-MIDDLE represents the latter policy. As shown in Figure 11, VA-MIDDLE improved the transaction rate by 2%, file creation rate by 40%, file deletion rate by 2%, data read rate by 11% and 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.

Metadata and Data Separation

We modified Postmark to report on the data read rate of files based on their proximity to metadata. Files in the system are divided into 19 bins, where *bin 0* is closest to metadata and *bin 18* is farthest. The data read rate for files in each bin is reported in Figure 12. *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*.

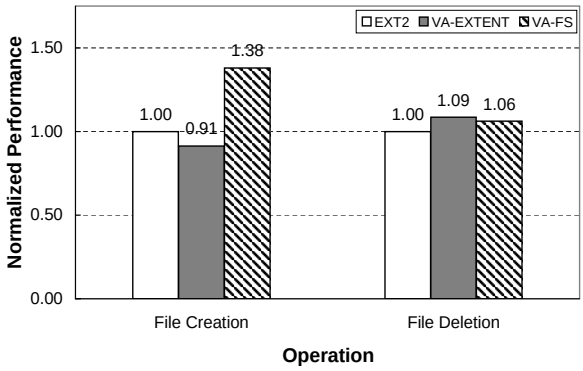


Figure 13: *Performance of file system create and delete operations. In this experiment, 50,000 files (between 8KB and 64KB) were created in 200 directories and deleted.*

Metadata Clustering

We configured Postmark to measure the speeds of file system create and delete operations. Postmark was set to have a 30% file system utilization in a 7GB partition. Figure 13 shows the results of this experiment. With the extent-based hardening, VA incurred an overhead of 9% in the file creation phase and showed 9% improvement in the file deletion phase. VA with the file system-based hardening showed 38% performance increase in the file creation phase and 6% improvement in the file deletion phase. In VA, VA metadata hardening and seek distances affect the performance of file create operations. In the file deletion phase, the seek operations to access metadata determines the performance. VA got faster deletion performance due to metadata clustering than EXT2.

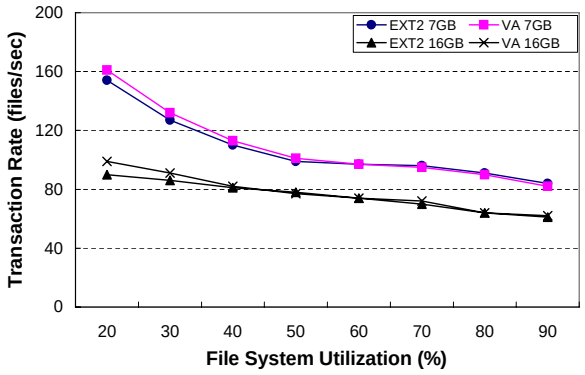


Figure 14: *Performance of VA compared to EXT2 for two partition sizes and different file system utilizations. Two partition sizes i.e., 7GB and 16GB were used. The transaction rate was measured from 20% to 90% file system utilization.*

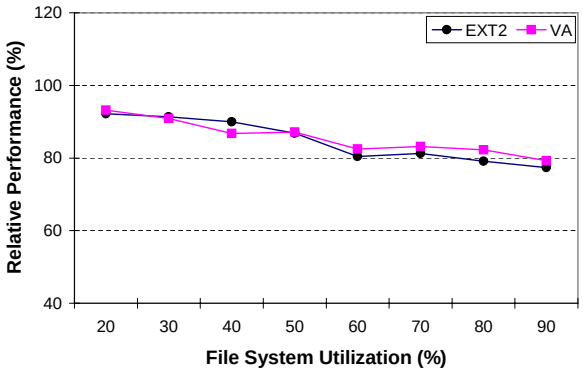


Figure 15: *Impact of file system aging on EXT2 and VA performance. Each value represents a percentage of the transaction rate of the corresponding empty file system.*

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 and compared it to that of EXT2 with the same configuration.

Figure 14 shows the results of Postmark. A large partition with smaller utilization works best in VA because data is relatively more clustered than EXT2. VA showed about 7.6% performance improvement (excluding the ZCAV bias) with 16GB partition at 20% file system utilization compared to EXT2. With the file system utilization more than 50%, VA still shows equal or better performance than EXT2 by 2 to 3%. 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.

File System Aging

In this experiment, we measured the impact of fragmentation of the file system due to aging on VA 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 [11].

First, it creates a number of directories and populates them with files up to 20% of file system utilization. It measures the transaction rate the same way as done in Postmark. Then it populates more files to have 30% of file system usage and measures the transaction rate again. It repeats these procedures up to 90% of file system utilization.

The graphs in Figure 15 show the performance of each file system utilization as a percentage of the transaction rate of the corresponding empty file system. As shown in Figure 15, as the file system ages, VA incurs a performance impact similar to or smaller than that of EXT2 for all file system utilizations.

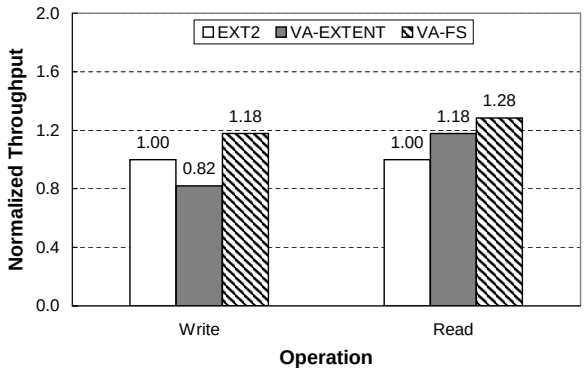
Multiple File Systems

Next, we considered the use of multiple file systems with virtual allocation. Specifically, we compared the fixed partition approach with our approach. In this experiment, we created two 3GB partitions and created an Ext2 file system on each partition. We ran Bonnie++ and Postmark on two file systems concurrently and compared the results between VA and the fixed partition approach.

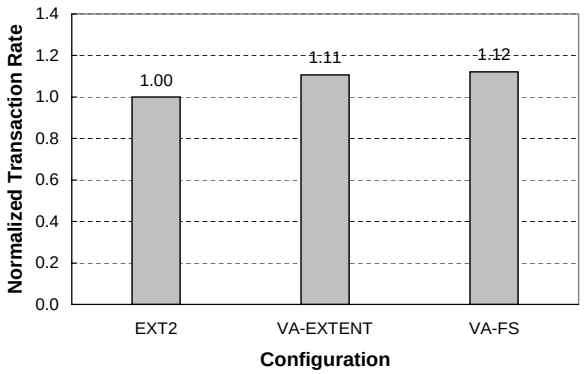
Figure 16(a) shows the average throughput of two file systems from Bonnie++ benchmark. In a large-file write workload, we observed that VA with the extent-based hardening incurred the overhead of 18% compared to the fixed partition approach and VA with the file system-based hardening showed better performance by 18%.

The read operation of VA shows large performance improvement i.e., by 18% with the extent-based hardening and by 28% with the file system-based hardening. 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 a lot of disk seek operations between the two partitions, whereas VA can reduce this seek distance significantly due to its allocate-on-write policy.

Figure 16(b) shows the results of Postmark. VA improved performance by 11% with the extent-based hardening and by 12% with the file system-based hardening.



(a) Bonnie++ results



(b) Postmark results

Figure 16: *Performance of multiple file systems with VA. Two 3GB partitions were used and two Ext2 file systems were created on each partition. Figure 16(a) shows the average read and write throughput of two file systems from Bonnie++. It was concurrently run on two file systems. Figure 16(b) shows the average transaction rate of two file systems from Postmark after being concurrently run on two file systems.*

These performance improvements are due to the reduction of seek distances similar to the case of the large-file workload. These results show that using VA has significant advantages over the fixed partition approach for these kinds of workloads i.e., we get substantial performance increases and at the same time, we increase the flexibilities of allocation across multiple file systems.

4.4 Results for the Other Workloads

Figure 17 shows the results of TPC-C workloads. The bar height represents the normalized elapsed time to that of EXT2 for various database operations. VA showed comparable performance in all kinds of database operations compared to EXT2 because the on-disk layout of this workload was similar. Transactions were done on one large file (a table-space), so most of the data was clustered in both the systems.

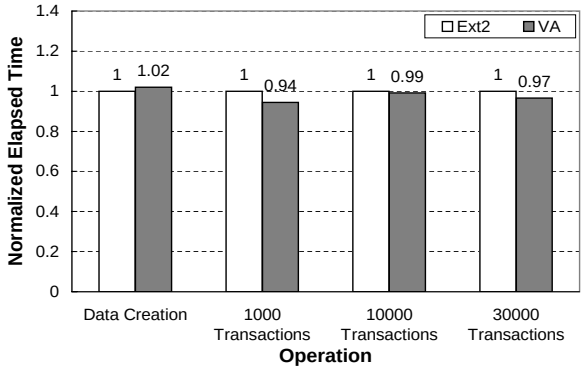


Figure 17: *Database workload performance of VA and EXT2. TPC-C workload was used.*

5 Discussions & Future Work

Storage allocation is persistent i.e., there is currently no mechanism for reclaiming allocated space of deleted files. In order to build a truly virtual storage system (similar to virtual memory), such a capability will be required. Since the ability to shrink a file system is rare, it will be a good research topic to figure out how to do this more efficiently than an exhaustive enumeration through all inodes and indirect blocks looking to see if any references to an extent still exist for reclaiming storage space.

The IP connectivity of I/O devices [25] 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 [27, 28]. We plan to pursue this topic in the future. We will also investigate the interaction between virtual allocation, and RAID, mirroring and striping typically employed within storage systems [26]. Our allocate-on-write policy would need to be suitably modified to fit into the logical device characteristics advertised by the LVM. We will in-

vestigate the interaction of VA with journaling file systems such as an Ext3 file system and log-structured file systems such as WAFL [8].

In Linux, it is possible to trace and identify data types if the data is metadata or data at the device-level. This will allow us a variety of data placement policies i.e., we can choose a data placement policy to increase the throughput (for normal data) or to increase data availability (for metadata), etc. Such an identification of metadata updates also enables the file system-based VA hardening which has been shown to significantly improve performance of VA. Similarly, identification of file name extensions may also enable file type-specific data allocation policies (for example, allocation of .avi and .mpg files on high throughput disks).

6 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.

7 Acknowledgments

We thank Uday Gupta and Jerry Cotter at EMC. Discussions with them motivated this work. We thank Jim Wyllie at IBM Almaden Research center and Pat Shuff for their feedback.

References

[1] L. Huang, G. Peng, and T. Chiueh. Multi-Dimensional Storage Virtualization. *In Proceedings of ACM SIGMETRICS 2004,* June 2004.

[2] 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. 2,* July 2003.

[3] M. Mesnier, G. Ganger, and E. Riedel. Object-Based Storage. *IEEE Communications Magazine, Vol. 41 No. 8 pp. 84-90,* August 2003.

[4] ReiserFS. Journaling file system for Linux based on balance tree algorithms. *http://www.namesys.com.*

[5] S. Best. JFS Overview. *http://www-106.ibm.com/developerworks/library/jfs.html.*

[6] M. Kaczmarski, T. Jiang, and D. Pease. Beyond Backup Toward Storage Management. *IBM Systems Journal, Vol. 42, No. 2,* July 2003.

[7] 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.

[8] D. Hitz, J. Lau, and M.A. Malcom. File System Design for an NFS File Server Appliance. *In Proceedings of the USENIX Winter Technical Conference,* San Francisco, CA, January 1994.

[9] J. Peacock. The Counterpoint Fast File System. *In Proceedings of the USENIX Winter Conference,* Dallas, Texas, January 1988.

[10] L. McVoy and S. Kleiman. Extent-like performance from a UNIX file system. *In Proceedings of the USENIX Winter Technical Conference,* Dallas, Texas, January 1991.

[11] K. Smith and M. Seltzer. A Comparison of FFS Disk Allocation Policies. *In USENIX Annual Technical Conference, pp. 15-26,* January 1996.

[12] VERITAS. Volume Manager for Windows with MSCS. VERITAS White Paper, http://www.veritas.com/van/products/volumemanagerwin.html, 2002.

[13] Transaction Processing Performance Council. *http://www.tpc.org.*

[14] Hammerora. The open source oracle load test tool. *http://hammerora.sourceforge.net.*

[15] Bonnie++. A benchmark suite aimed at performing a number of tests of hard drive and file system performance. *http://www.coker.com.au/bonnie++.*

[16] R. Meter. Observing the Effects of Multi-Zone Disks. *In Proceedings of the USENIX Annual Technical Conference,* 1997.

[17] D. Ellard and M. Seltzer. NFS Tricks and Benchmarking Traps. *In Proceedings of the Annual USENIX Technical Conference, FREENIX Track, pages 101–114,* June 2003.

[18] W. Vogels. File System Usage in Windows NT 4.0. *In Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP '99)*, Kiawah Island, South Carolina, December 1999.

[19] R. Coker ZCAV: a Hard Drive Testing Program. *http://www.coker.com.au/bonnie++/zcav.*

[20] J. Katcher. PostMark: a New Filesystem Benchmark. *Technical Report TR3022, Network Appliance*, 1997. *http://www.netapp.com/tech_library/3022.html.*

[21] M. McKusick, W. Joy, S. Leffler, R. Fabry, A Fast File System for UNIX. *ACM Transactions on Computer Systems, 2(3):181-197*, August 1984.

[22] G. Ganger, Y. Patt. Metadata Update Performance in File Systems. *USENIX Symposium on Operating Systems Design and Implementation, pp. 49-60*, November 1994.

[23] R. Wang, D. Patterson, and T. Anderson. Virtual Log Based File Systems for a Programmable Disk. *Symposium on Operating Systems Design and Implementation*, New Orleans, LA, February 1999.

[24] M. Sivathanu, V. Prabhakaran, F. Popovici, T. Denehy, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. Semantically-Smart Disk Systems. *In Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST'03)*, March 2003.

[25] M. Krueger, R. Haagens, C. Sapuntzakis, and M. Bakke. RFC 3347:Small Computer Systems Interface protocol over the Internet (iSCSI) Requirements and Design Considerations, 2002.

[26] D. Patterson, G. Gibson, and R. Katz. A Case for Redundant Array of Inexpensive Disks (RAID). *In Proceedings of ACM SIGMOD*, 1988.

[27] W. Ng, B. Hillyer, E. Shriver, E. Gabber, B. Ozden. Obtaining High Performance for Storage Outsourcing. *In Proceedings of Conference on File and Storage Technologies (FAST'02)*, January 2002.

[28] X. He, Q. Yang and M. Zhang. A Caching Strategy to Improve iSCSI Performance. *IEEE Conference on Local Computer Networks*, November 2002.