Revisiting the Storage Stack in Virtualized NAS Environments

Dean Hildebrand, Anna Povzner, Renu Tewari
IBM Almaden

Vasily Tarasov
Stony Brook University

Abstract

Cloud architectures are moving away from a traditional data center design with SAN and NAS attached storage to a more flexible solution based on virtual machines with NAS attached storage. While VM storage based on NAS is ideal to meet the high scale, low cost, and manageability requirements of the cloud, it significantly alters the I/O profile for which NAS storage is designed. In this paper, we explore the storage stack in a virtualized NAS environment and highlight corresponding performance implications.

1. Introduction

There is a shift in cloud architectures from the traditional data center design based on physical machines with SAN or NAS attached storage to a more flexible and lower cost solution using virtual machines and NAS storage. In these architectures, the virtual machine's associated disks exist as files in a NAS data-store and are accessed using a file access protocol such as NFS. Storing the disks of a virtual machine as files (instead of LUNs) makes VMs easier to manage, migrate, clone, and access. The traditional NAS storage stack, however, is altered as the guest block layer now resides *above* a file access protocol. The block layer, which expects a consistent and low-latency I/O layer beneath it, must now utilize a file access protocol with a higher latency and looser close-to-open consistency semantics. Moreover, what the NAS storage system views as a file and optimizes for file operations is actually a *disk*.

While some work exists that investigates the impact of virtualized environments with traditional block-based storage [1, 2], there is no formal investigation on the impact of virtual machines with NAS storage. In this paper, we investigate how a virtualized stack affects the behavior and workload characteristics of the NAS storage system. Some key observations we make in the paper are:

- The NAS workload consists of only I/O requests and no metadata requests, i.e., no create, delete, change attribute operations. Using I/O to perform metadata operations can increase performance in certain instances.
- *All* NAS write requests require a commit to the stable storage instead of the weaker semantics of a commit-on-close or on an fsync.
- Sequential I/O at the guest level is highly likely to be transformed to random I/O at the NAS level.

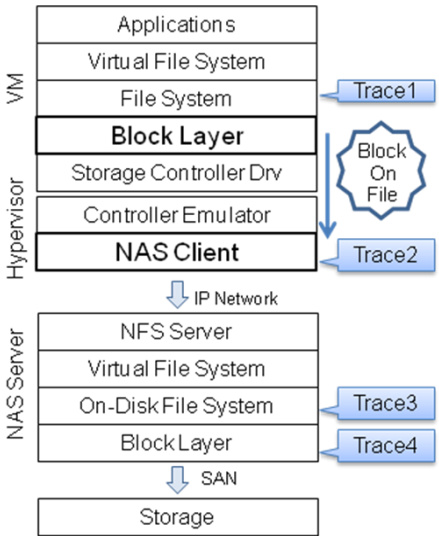

Figure 1: VM with NAS - Numerous software layers are punctuated by a block layer in the guest operating system driving the requests to the NAS file layer. Our system traces the stack at 4 points simultaneously to capture the I/O behavior at each level.

- NAS workload changes from many smaller files to a small number of considerably larger files.
- *Small* writes generate a read-modify-write update from the guest and *all* writes may generate read-modify-write update on the server side, significantly degrading performance (~20-69%).
- *Small* reads in the guest can double the amount of data read at the NAS level.

2. VM and NAS: An Overview

Virtual machines encapsulate the entire guest file system in a virtual disk. A VM residing over NAS increases the number of layers in the virtual software stack as shown in Figure 1. Virtual disks are typically stored as files, e.g., VMWare .vmdk, on the NAS store. The NAS store in turn consists of an NFS or CIFS server with a back-end file system such as ZFS, WAFL, or GPFS [3]. Hypervisors provide VMs with access to these disk images using a NAS protocol.

Figure 1 shows the VM with NAS storage stack. Application accesses to the guest file system are sent to the storage controller emulator in the hypervisor by the guest block layer. The storage controller emulator then re-issues requests from the guest block layer as file-level I/O requests to the disk image via the NAS client. The NAS client in turn performs I/O requests to the disk image file stored in the server file system.

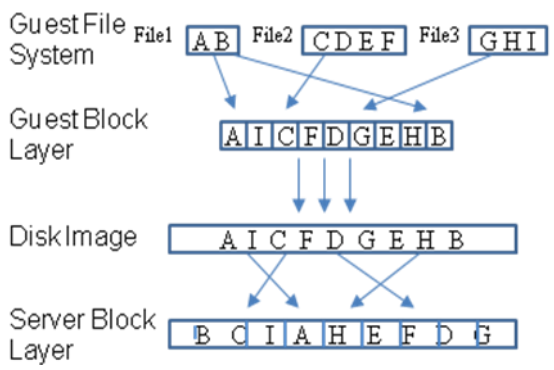

Figure 2: Data through the layers - File data in the guest file system are organized as blocks in the guest block layer. Block boundaries are lost when represented in the disk image file. Guest data blocks are re-arranged in the server block layer, possibly straddling server block boundaries.

The movement of the I/O request through the various layers of the virtualized stack has two adverse side effects. First, the increased number of layers in the software stack increases the amount of processing each request must experience (and hence increase its I/O request latency). Second, as Figure 2 demonstrates, files in the guest file system have their data block addresses stored relative to the guest block layer addresses. These data blocks are actually offsets and extents in the disk image file stored on the server file system. By storing these blocks in a file, the block layer's need to cache entire blocks causes read-modify-write operations over the NAS protocol, which degrades performance. The server file system then rearranges the original guest data blocks once again as it is stored in the server block layer. Guest data blocks can be split across multiple server file system blocks depending on the start offset of the file system within the disk image file, causing even more read-modify-write operations on the server side.

By storing the VM's disk as a file, the virtualized stack changes the workload profile of the server file system from millions of small files to a much smaller number of very large disk images. In addition, our preliminary results indicate that access to these disk images will most likely consist of small and random I/O requests with very few metadata requests, such as create, delete, and change attributes. Supporting small and random I/O requests to large files runs contrary to the current focus on optimizing the backend NAS store for create operations per second and supporting billions of files in a single directory [4].

3. Block over File: A Good Idea?

This section investigates how the I/O workload changes, and the resultant performance impact, when virtual machines are introduced into the NAS environment. We ran several benchmarks in standard NAS (*NFS)* and virtualized NAS (*VM-NFS)* environments and compared the generated I/O workloads and subsequent performance. Both environments used GPFS as a server file system running on an IBM System x3550 equipped with a five disk RAID-5. In the standard NAS environment, benchmarks ran directly on an NFS client, and in the virtualized NAS environment, benchmarks ran in a virtual machine. The virtual machine ran on VMware ESX 4.1 with a Fedora 14 guest operating system stored locally on the ESX server. The experiments were executed on an Ext2 file system, whose disk image was stored on the NFS data store. Both VM-NFS and standard NFS client were configured with 512MB of memory and connected to the NAS server with a GigE network. NFS data transfer buffer sizes were set to 64KB ("rsize") and 512KB ("wsize").

Filebench was used to generate all workloads. We used a multi-layer, correlated tracing framework to collect traces at four trace points in the VM with NAS stack (as shown in Figure 1). Through these traces, we could decipher how the guest block layer interacts with the NAS client file layer and changes the generated I/O workload at the server file system.

In this paper, we characterize and compare workloads seen by the NAS storage server in both standard NAS and virtualized NAS environments by observing types and number of operations, amount of I/O, and I/O patterns. We first investigate metadata transformations in Section 3.1 using create and stat experiments and then investigate data transformations in Section 3.2.

3.1 Metadata Workloads

In this section, we show the impact of guest file system metadata requests as they are transformed into read and write accesses to a disk image file. The optimizations made by many file systems for metadata operations such as creates become irrelevant in virtualized NAS environments since metadata is always transformed to data I/O. It is important to note that the exact type of transformation of metadata to data and its subsequent performance impact depends on the guest file system and server file system. We used a bare-bones guest file system ("Ext2") in this paper, but plan to use more advanced file systems such as Ext4 or btrfs in the future. We chose GPFS for the server file system due to both our familiarity as well as its prevalence in real-world data centers.

File creation. To evaluate file creates, we created a directory and filled it with 100,000 zero-length files. Each create file operation was followed by a close file operation. We record the number of ops/sec, the number of NFS create, getattr and mkdir operations, and the amount of data read and written by the server file system.

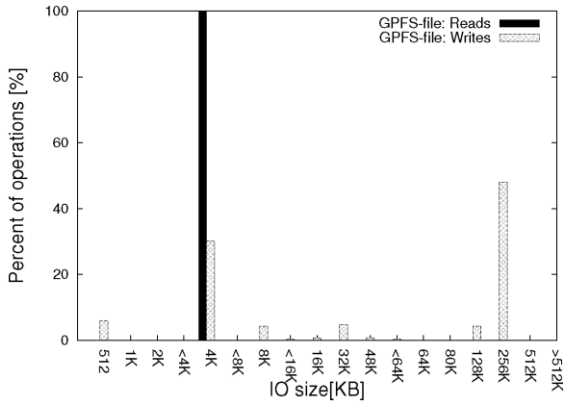

Figure 3: Virtual file create I/O workload. Read and write sizes at server file system layer in the VM-NFS environment when 100,000 files were created in a single directory.

	Single Directory		Mean Dir. Width 20	
	VM-NFS	NFS	VM-NFS	NFS
ops/sec	1408	2270	7406	2125
creates	0	100000	0	100000
getattrs	0	500216	0	500640
mkdirs	0	1	0	5105
reads	21.5MB	0MB	25.8MB	0MB
writes	21MB	0MB	15.5MB	0MB

Table 1: File create operations. Operations at server file system layer file creation in a single directory and in a directory tree with average 20 items in each directory.

As shown in Table 1, create, getattr and mkdir metadata operations observed by the NAS storage in the standard setup (*NFS*) become read and write operations in the virtualized NAS (*VM-NFS*). 100,000 create/close calls from the guest became 21.5MB worth of reads and 21MB of writes. Figure 3 shows that all reads are 4KB, while write sizes are larger, with 49% of writes being 256KB. In addition, 98% of reads and 52% of writes are sequential. High sequentiality is attributed to the Ext2 file system attempting to co-locate created inodes.

In this experiment, the transformation of metadata to data degraded VM-NFS performance to half of standard NFS performance (Table 1). However, Figure 4 shows that VM-NFS performance greatly depends on the directory tree structure. Each experiment in Figure 4 creates files in a different directory structure defined by a mean number of entries in each directory, mean *dirwidth*, starting from 20 (5105 directories, tree depth 4) to 1,000,000 (all files in one directory).

The larger and deeper directory tree results in the better VM-NFS performance. While the total amount of data transferred remains similar (Table 1), creating files in a deeper directory tree results in larger write sizes (not shown), which translates into higher write throughput. Standard NFS performance remained independent of the directory structure and was worse in most cases than VM-NFS. This can be attributed to the directory locking and lock contentions in the server

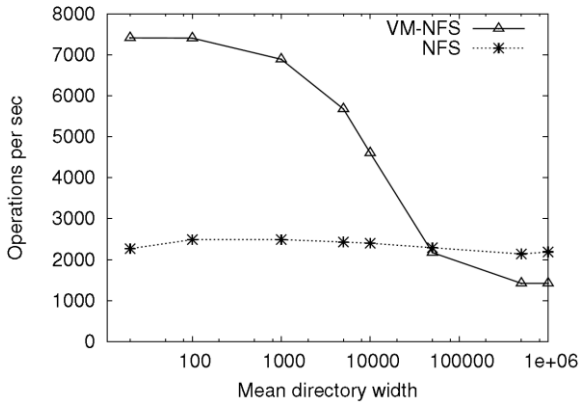

Figure 4: File create performance. Performance of file create experiment as mean *dirwidth* increases (decreasing the total number of directories from 5105 to 1).

file system that does not exist when VM-NFS performs I/O instead of file creates.

File stat. To confirm our observations, we investigate "get file status" transformations. This experiment performs stat() calls for 2 minutes on random files in a single directory of 100,000 files.

Table 2 shows that the lookup and getattr operations observed by the server file system with standard NFS became 4K read operations with VM-NFS. About a million stat() calls are transformed into 6743 4K read requests (total 26.3 MB).

Virtual *stat* performance also depends on the directory tree structure: stat() files from one directory resulted in 8622 stat/sec, but using 5105 directories/tree depth 4 (same total number of files) resulted in 12167 stat/sec. Figure 5 shows that using larger directory trees causes stat() operations to transform into read requests with smaller seek distance between reads, thus improving the *stat* performance.

Virtual *stat* performance was better than standard NFS in our experiments (Table 2). This was a surprising result, and can be attributed to several factors: NFS attribute caching timeouts; possible server file system lock contentions; and most significantly the ability for Ext2 to co-locate inodes and therefore have each read request prefetch several inodes at a time.

3.2 Read and Write Workloads

Read and write intensive workloads are also transformed when introducing VMs into the NAS environment due to the several layers, including a block layer, which data must pass through before arriving at the NAS client in the hypervisor. This section shows that workloads with smaller than 4KB reads and writes suffer the most from these additional layers due to full page read issues and read-modify-write at the guest. In addition, guest data blocks that are misaligned with respect to the server block layer decrease performance by causing additional server disk

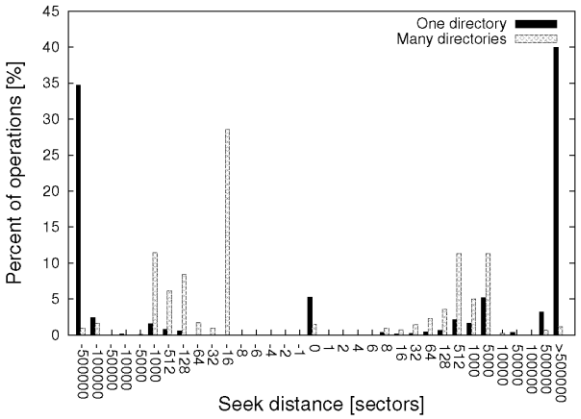

Figure 5: stat() randomness. Seek distance between reads at server file system with stat() on files in a single directory versus a directory tree with avg. 20 entries in each directory.

	stat/sec	lookup/getattr	data read
VM-NFS	8622	0	26.3MB
NFS	2656	610721	0MB

Table 2: File stat() operations. Operations at server file system when performing stat() on random files. VM-NFS executed 6743 4KB read requests.

	2KB reads		128KB reads	
	VM-NFS	NFS	VM-NFS	NFS
MB/s	0.6MB/s	2.5MB/s	46.5MB/s	50MB/s
app reads	2048M	2048M	2048M	2048M
file reads	4710M	1140M	2187M	2804M
block reads	6758M	5853M	3280M	3769M

Table 3: Random 2KB reads. Performance and amount of data transferred at server file system and block layer for random read (20 threads) from 4GB file.

accesses. Writes are further impacted because data flushed from the guest page cache must be immediately committed to disk and cannot be buffered in server file system cache before being written to disk.

Write workloads. To generate a sequential write workload, Filebench used a single thread to write a 4GB file, varying the write sizes from 2KB to 4MB.

Figure 6 shows sequential write throughput with virtualized NAS (VM-NFS) is approximately half that of standard NFS performance, with an even larger difference with 2KB writes. In both setups, the performance increased as write size increased since disk-based systems can generally handle larger requests more efficiently.

Figure 7 shows that 2KB writes in VM-NFS cause read-modify-write in the guest ("file-read")—in order to write 4GB of data, the guest must also read 4GB of data. Making matters worse, reads are performed as synchronous 4KB requests without any read-ahead from the guest (not shown). In contrast, NFS does not perform file reads when writing in 2KB chunks. The "block-read" and "block-write" parts of Figure 7 also show that 4GB of reads from the guest results in even

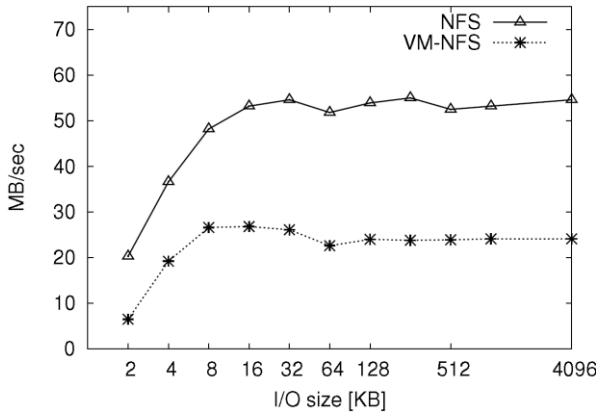

Figure 6: NFS and VM-NFS sequential write performance. Throughput when writing a 4GB file with write sizes ranging from 2KB to 4MB.

larger amount of data read at the server file system block layer due to block misalignment.

VM-NFS performance also degrades with larger write requests. As shown in Figure 7 for 128KB write, VM-NFS demonstrates more server file system block layer reads and writes. This is a result of read-modify-write operations on the server side, as guest data blocks can be split across multiple server blocks.

Read workloads. Read throughput is mostly affected in the case of the small request sizes. In this experiment, we compare performing random 2KB and 128KB reads from a 4GB file using 20 read threads, where total amount of data read in each case is 2GB. Table 3 shows that 2KB reads in VM-NFS setup suffer 4 times lower performance than NFS, while 128KB read performance is very similar to NFS.

The main reason for VM-NFS 2KB performance degradation is the full page read in the guest OS. Table 3 shows that in order to read 2GB, the guest must read 4GB from the server file system.

4. Eliminating Client-Side Block on File

As this paper states, the layering of the guest block layer on top of the NFS file layer can reduce performance, stress client networks, and reduce the server file system's ability to perform standard optimizations. Applications that were once running successfully in a SAN or NAS environment may now see unacceptable storage performance with the same storage hardware. One possible solution is for applications to use the NAS client in the guest OS (instead of a disk image), but this technique no longer virtualizes the I/O and places data centers at the mercy of features and bugs in the guest OS.

Techniques are required to allow virtual machines to continue virtualizing I/O and using disk images while giving the server file system more information of the original guest operations. One possibility is to use

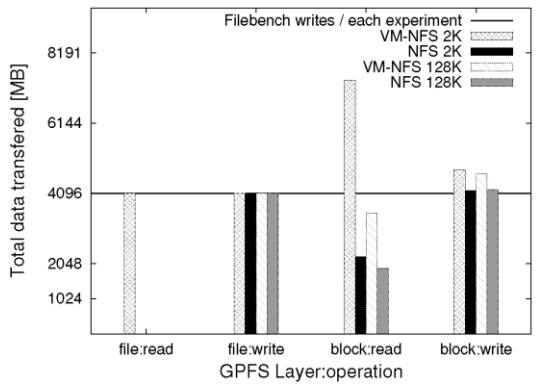

Figure 7: Sequential write data transfer. Amount of data transferred at server file system and server block layer during sequential write of 4GB file.

a para-virtualized NAS client driver in the guest OS. In this model, applications running in the guest would not use a local file system such as Ext2 but rather a NFS device that passes its operations to the hypervisor and then to a disk image in the server file system.

To simulate such an environment, we mounted the disk image in the server file system (using the loopback driver), exported it via NFS, and then mounted it from the guest using the Linux NFS client. As shown in Table 4, Guest-NFS more than doubles write throughput as compared with the standard method of translating requests from a SCSI emulator to the NFS *(VM-NFS)*. Guest-NFS sends I/O requests directly to the server, avoiding the need for read-modify-write from the hypervisor. Guest-NFS does not match the performance of standard NFS (Linux-NFS) since the loopback mounted disk image on the server reduces all write requests to 4KB.

5. Future Work

We plan to extend this work in several ways. First, more information is needed regarding the impact of virtualizing NAS applications on I/O workloads and the server file system. For instance, how does the elimination of large number of file creates and the existence of many large and sparse disk images change NAS protocols and the server file system design. In addition, we would like to work with specialized benchmarks such as VMMark [13] as well as find and replay traces from real virtual data centers.

Second, virtual disk image fragmentation can have a great impact on the randomness of read and write requests sent to the server file system. We plan to use a tool like Impressions [14] to generate fragmented disk images and study their effect on I/O workloads.

Finally, as discussed, when layering the guest block layer over the NAS file layer, the original file system operations are lost, as all requests are transformed into read and write requests to the disk image file. Hence, the server file system cannot cache

	Unalloc (MB/s)	Alloc (MB/s)
VM-NFS	36.3	11.1
Linux-NFS	98.3	92.2
Guest-NFS	66.2	50.0

Table 4: Small sequential write performance. Guest-NFS improves performance by avoiding client side read-modify-write as seen by ESX-NFS.

or prefetch data based on whether it was a metadata or data request. This is very common, for example, to ensure large data reads do not evict all directory information from the cache. We are investigating methods of restoring the original file system operations, including client side hints and possible server side reverse engineering of the I/O requests.

6. Related Work

Several NAS storage systems are available today that target virtual data centers [5-8]. While each vendor provides a different solution, we expect that all have an architecture similar to Figure 1 and are equally subject to the issues raised in this paper.

Several papers have shown that I/O performance degradation can be incurred by the use of legacy device drivers and even para-virtualized I/O drivers in the guest [9, 10]. Solutions for these device issues include direct access (pass-through) and self-virtualizing devices [11]. Since the NAS client is in the hypervisor, improvements in these I/O virtualization techniques are unlikely to significantly alter the I/O workload generated in VM with NAS architectures.

Workload characterization using traces is a popular research topic [15-19]. A few virtual machine I/O workload studies exist that show the resultant I/O workload in the hypervisor when applications are executed in the guest [1, 2, 12]. Corresponding traces and other publically available traces are usually from only a single layer of the I/O stack [20]. As far as we are aware, there are no public traces that encapsulate multiple layers of a virtualized I/O stack. We introduce a novel approach of not only focusing on virtualized NAS environments, but also showing the I/O workload at the several different levels in the stack.

7. Conclusion

While the combined use of virtual machines and NAS has the potential to reduce costs and simplify management, VMs may notice a substantial drop in I/O performance due to the introduction of several new software layers, including the introduction of a guest block layer over the NAS file layer. In this paper, we identified and demonstrated several application workloads in a virtual NAS environment that both change the I/O workload and impact performance. In addition, we demonstrate how it is possible to avoid the block on file architecture and send the original application file requests to the NAS server.

8. References

[1] I. Ahmad, "Easy and Efficient Disk I/O Workload Characterization in VMware ESX Server," in *Proceedings of IISWC07*, Boston, MA, 2007.

[2] I. Ahmad, J.M. Anderson, A.M. Holler, R. Kambo, and V. Makhija, "An analysis of disk performance in VMware ESX server virtual machines," in *Proceedings of the IEEE International Workshop on Workload Characterization*, 2003.

[3] F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," in *Proceedings of the USENIX Conference on File and Storage Technologies*, San Francisco, CA, 2002.

[4] SpecSFS, www.spec.org/sfs93.

[5] "NetApp Storage Solutions for Server Virtualization," www.netapp.com/us/solutions/infrastructure/virtualization/server.

[6] "IBM Scale Out Network Attached Storage," www-03.ibm.com/systems/storage/network/sonas

[7] "Gluster," www.gluster.com/2011/02/08/gluster-introduces-first-scale-out-nas-virtual-appliances-for-vmware-and-amazon-web-services/.

[8] "EMC," www.emc.com/solutions/business-need/virtualizing-information-infrastructure/file-virtualizations.htm.

[9] J. Santos, Y. Turner, J. Janakiraman, and I. Pratt, "Bridging the gap between software and hardware techniques for i/o virtualization," in *Proceedings of the USENIX Annual Technical Conference*, 2008.

[10] J. Sugerman, G. Venkitachalam, and B. Lim, "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor," in *Proceedings of the USENIX Annual Technical Conference*, 2001.

[11] H. Raj and K. Schwan, "High performance and scalable I/O virtualization via self-virtualized devices," in *Proceedings of the 16th international symposium on High performance distributed computing*, 2007.

[12] C. Kumar A. Gulanti, I. Ahmad, "Storage Workload Characterization and Consolidation in Virtualized Environments," in *Proceedings of the Workshop on Virtualization Performance: Analysis, Characterization, and Tools*, 2009.

[13] "VMMark 2.0," www.vmware.com/products/vmmark/overview.html.

[14] N. Agrawal, A.C. Arpaci-Dusseau, and R.H. Arpaci-Dusseau, "Generating Realistic Impressions for File-System Benchmarking," in *Proceedings of the 7th Conference on File and Storage Technologies*, San Francisco, CA, 2009.

[15] D. Roselli. Characteristics of File System Workloads. Technical Report UCB/CSD-98-1029, University of California at Berkeley, 1998.

[16] M. Zhou and A. Smith, "Analysis of Personal Computer Workloads," in *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 1999

[17] D. Roselli, J. Lorch, and T. Anderson, "A Comparison of File System Workloads," in *Proceedings of USENIX Technical Conference*, 2000.

[18] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda, "Characterization of Storage Workload Traces from Production Windows Servers," in *Proceedings of IEEE International Symposium on Workload Characterization*, 2008.

[19] N. Yadwadkar, C. Bhattacharyya, and K. Gopinath, "Discovery of Application Workloads from Network File Traces," in *Proceedings of USENIX Conference on File and Storage Technologies*, 2010.

[20] SNIA I/O Traces, Tools and Analysis Technical Working Group Trace Repository, iotta.snia.org