Virtual Machine Workloads: The Case for New Benchmarks for NAS

Appears in the proceedings of the 11th USENIX conference on File and Storage Technologies (FAST'13)

Vasily Tarasov[1], Dean Hildebrand[2], Geoff Kuenning[3], Erez Zadok[1]
[1]*Stony Brook University,* [2]*IBM Research—Almaden,* [3]*Harvey Mudd College*

Abstract

Network Attached Storage (NAS) and Virtual Machines (VMs) are widely used in data centers thanks to their manageability, scalability, and ability to consolidate resources. But the shift from physical to virtual clients drastically changes the I/O workloads seen on NAS servers, due to guest file system encapsulation in virtual disk images and the multiplexing of request streams from different VMs. Unfortunately, current NAS workload generators and benchmarks produce workloads typical to physical machines.

This paper makes two contributions. First, we studied the extent to which virtualization is changing existing NAS workloads. We observed significant changes, including the disappearance of file system meta-data operations at the NAS layer, changed I/O sizes, and increased randomness. Second, we created a set of versatile NAS benchmarks to synthesize virtualized workloads. This allows us to generate accurate virtualized workloads *without* the effort and limitations associated with setting up a full virtualized environment. Our experiments demonstrate that the relative error of our *virtualized benchmarks*, evaluated across 11 parameters, averages less than 10%.

1 Introduction

By the end of 2012 almost half of all applications running on x86 servers will be virtualized; in 2014 this number is projected to be close to 70% [8,9]. Virtualization, if applied properly, can significantly improve system utilization, reduce management costs, and increase system reliability and scalability. With all the benefits of virtualization, managing the growth and scalability of storage is emerging as a major challenge.

In recent years, growth in network-based storage has outpaced that of direct-attached disks; by 2014 more than 90% of enterprise storage capacity is expected to be served by Network Attached Storage (NAS) and Storage Area Networks (SAN) [50]. Network-based storage can improve availability and scalability by providing shared access to large amounts of data. Within the network-based storage market, NAS capacity is predicted to increase at an annual growth rate of 60%, as compared to only 22% for SAN [43]. This faster NAS growth is explained in part by its lower cost and its convenient file system interface, which is richer, easier to manage, and more flexible than the block-level SAN interface.

The rapid expansion of virtualization and NAS has lead to explosive growth in the number of virtual disk images being stored on NAS servers. Encapsulating file systems in virtual disk image files simplifies the implementation of features such as migration, cloning, and snapshotting, since they naturally map to existing NAS functions. In addition, non-virtualized hosts can co-exist peacefully with virtualized ones that use the same NAS interface, which permits a gradual migration of services from physical to virtual machines.

Storage performance plays a crucial role when administrators select the best NAS for their environment. One traditional way to evaluate NAS performance is to run a file system benchmark, such as SPECsfs2008 [38]. Vendors periodically submit the results of SPECsfs2008 to SPEC; the most recent submission was in November 2012. Because widely publicized benchmarks such as SPECsfs2008 figure so prominently in configuration and purchase decisions, it is essential to ensure that the workloads they generate represent what is observed in real-world data centers.

This paper makes two contributions: an analysis of changing virtualized NAS workloads, and the design and implementation of a system to generate realistic virtualized NAS workloads. We first demonstrate that the workloads generated by many current file system benchmarks do not represent the actual workloads produced by VMs. This in turn leads to a situation where the performance results of a benchmark deviate significantly from the performance observed in real-world deployments. Although benchmarks are never perfect models of real workloads, the introduction of VMs has exacerbated the problem significantly. Consider just one example, the percentage of data and meta-data operations generated by physical and virtualized clients. Table 1 presents the results for the SPECsfs2008 and Filebench web-server benchmarks that attempt to provide a "realistic" mix of meta-data and data operations. We see that meta-data procedures, which dominated in physical workloads, are

NFS procedures	**Physical clients (SPECsfs2008/Filebench)**	**Virtualized clients**
Data	28% / 36%	99%
Meta-data	72% / 64%	<1%

Table 1: The striking differences between virtualized and physical workloads for two benchmarks: SPECsfs2008 and Filebench (Web-server profile). Data operations include `READ` *and* `WRITE`*. All other operations (e.g.,* `CREATE`, `GETATTR`, `READDIR`*) are characterized as meta-data.*

almost non-existent when VMs are utilized. The reason is that VMs store their guest file system inside large disk image files. Consequently, all meta-data operations (and indeed all data operations) from the applications are converted into simple reads and writes to the image file.

Meta-data-to-data conversion is just one example of the way workloads shift when virtual machines are introduced. In this paper we examine, by collecting and analyzing a set of I/O traces generated by current benchmarks, how NAS workloads change when used in virtualized environments. We then leverage multi-dimensional trace analysis techniques to convert these traces to benchmarks [13, 40]. Our new virtual benchmarks are flexible and configurable, and support single- and multi-VM workloads. With multi-VM workloads, the emulated VMs can all run the same or *different* application workloads (a common consequence of resource consolidation). Further, users do not need to go through a complex deployment process, such as hypervisor setup and per-VM OS and application installation, but can instead just run our benchmarks. This is useful because administrators typically do not have access to the production environment when evaluating new or existing NAS servers for prospective virtualized clients. Finally, some benchmarks such as SPECsfs cannot be usefully run inside a VM because they do not support file-level interfaces and will continue to generate a physical workload to the NAS server; this means that new benchmarks can be the only viable evaluation option. Our benchmarks are capable of simulating a high load (i.e., many VMs) using only modest resources. Our experiments demonstrate that the accuracy of our benchmarks remains within 10% across 11 important parameters.

2 Background

In this section, we present several common data access methods for virtualized applications, describe in depth the changes in the virtualized NAS I/O stack (VM-NAS), and then explain the challenges in benchmarking NAS systems in virtualized environments.

2.1 Data Access Options for VMs

Many applications are designed to access data using a conventional POSIX file system interface. The methods that are currently used to provide this type of access in a VM can be classified into two categories: (1) emulated block devices (typically managed in the guest by a local file system); and (2) guest network file system clients.

Figure 1 illustrates both approaches. With an emulated block device, the hypervisor emulates an I/O controller with a connected disk drive. Emulation is completely transparent to the guest OS, and the virtual I/O controller and disk drives appear as physical devices to the OS. The guest OS typically formats the disk drive

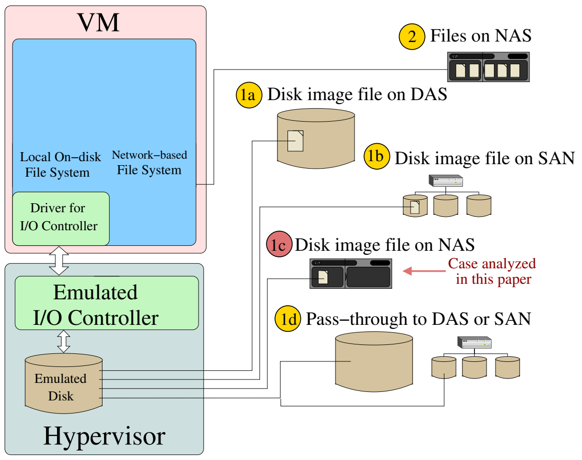

VM
Local On–disk File System
Network–based File System
Driver for I/O Controller
Emulated I/O Controller
Emulated Disk
Hypervisor
2 Files on NAS
1a Disk image file on DAS
1b Disk image file on SAN
1c Disk image file on NAS
Case analyzed in this paper
1d Pass–through to DAS or SAN

Figure 1: VM data-access methods. Cases 1a–1d correspond to the emulated-block-device architecture. Case 2 corresponds to the use of guest network file system clients.

with a local file system or uses it as a raw block device. When an emulated block device is backed by file-based storage, we call the backing files *disk image files*.

2.1.1 Emulated Block Devices

Figure 1 shows several options for implementing the back end of an emulated block device:

1a. A file located on a local file system that is deployed on Direct Attached Storage (DAS). This approach is used, for example, by home and office installations of VMware Workstation [39] or Oracle VirtualBox [44]. Such systems often keep their disk images on local file systems (e.g., Ext3, NTFS). Although this architecture works for small deployments, it is rarely used in large enterprises where scalability, manageability, and high availability are critical.

1b. A disk image file is stored on a (possibly clustered) file system deployed over a Storage Area Network (SAN) (e.g., VMware's VMFS file system [46]). A SAN offers low-latency shared access to the available block devices, which allows high-performance clustered file systems to be deployed on top of the SAN. This architecture simplifies VM migration and offers higher scalability than DAS, but SAN hardware is more expensive and complex to administer.

1c. A disk image file stored on Network Attached Storage (NAS). In this architecture, which we call *VM-NAS*, the host's hypervisor passes I/O requests from the virtual machine to an NFS or SMB client, which in turn then accesses a disk image file stored on an external file server. The hypervisor is completely unaware of the storage architecture behind the NAS interface. NAS provides the scalability, reliability, and data mobility needed for efficient VM management. Typically, NAS solutions are cheaper than SANs due to their use of IP networks, and

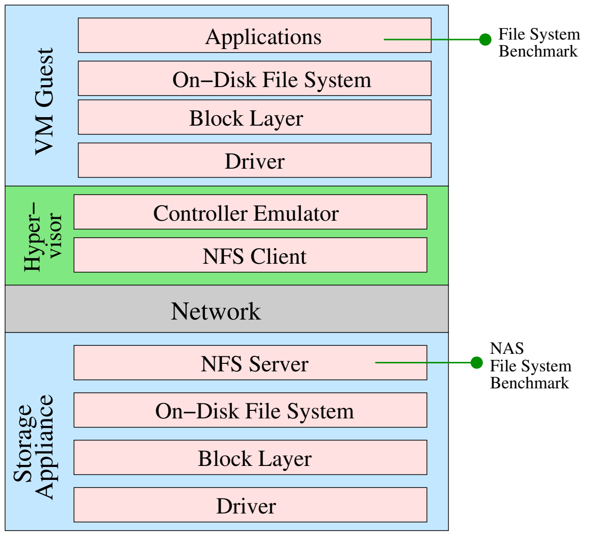


Figure 2: VM-NAS I/O Stack: VMs access and store virtual disk images on NAS.

are simpler to configure and manage. These properties have increased the use of NAS in virtual environments and encouraged several companies to create solutions for disk image files management at the NAS [6, 36, 41].

1d. Pass-through to DAS or SAN. In this case, virtual disks are backed up by a real block device (not a file), which can be on a SAN or DAS. This approach is less flexible than disk image files, but can offer lower overhead because one level of indirection—the host file system—is eliminated.

2.1.2 Network Clients in the Guest

The other approach for providing storage to a virtual machine is to let a network-based file system (e.g., NFS) provide access to the data directly from the guest (case 2 in Figure 1). This model avoids the need for disk image files, so no block-device emulation is needed. This eliminates emulation overheads, but lacks many of the benefits associated with virtualization, such as consistent snapshots, thin provisioning, cloning, disaster recovery. Also, not every guest OS supports every NAS protocol, which fetters the ability of a hypervisor and its storage system to support all guest OS types. Further, cloud management architectures such as VMware's vCloud and OpenStack do not support this design [32, 42].

2.2 VM-NAS I/O Stack

In this paper we focus on the VM-NAS architecture, where VM disks are emulated by disk image files stored on NAS (case 1c in Section 2.1.1 and in Figure 1). To the best of our knowledge, even though this architecture is becoming popular in virtual data centers [43, 50], there has been no study of the significant transformations in typical NAS I/O workloads caused by server virtualization. This paper is a first step towards a better understanding of NAS workloads in virtualized environments and the development of suitable benchmarks for NAS to be used in industry and academia.

When VMs and NAS are used together, the corresponding I/O stack becomes deeper and more complex, as seen in Figure 2. As they pass through the layers, I/O requests significantly change their properties. At the top of the stack, applications access data using system calls such as `create`, `read`, `write`, and `unlink`. These system calls invoke the underlying guest file system, which in turn converts application calls into I/O requests to the block layer. The file system maintains data and metadata layouts, manages concurrent accesses, and often caches and prefetches data to improve application performance. All of these features change the pattern of application requests.

The guest OS's block layer receives requests from the file system and reorders and merges them to increase performance, provide process fairness, and prioritize requests. The I/O controller driver, located beneath the generic block layer, imposes extra limitations on the requests in accordance with the virtual device's capabilities (e.g., trims requests to the maximum supported size and limits the NCQ queue length [51]).

After that, requests cross the software-hardware boundary for the first time (here, the hardware is emulated). The hypervisor's emulated controller translates the guest's block-layer requests into reads and writes to the corresponding disk image files. Various request transformations can be done by the hypervisor to optimize performance and provide fair access to the data from multiple VMs [18].

The hypervisor contains its own network file system client (e.g., NFS), which can cache data, limit read and write sizes, and perform other request transformations. In this paper we focus on NFSv3 because it is one of the most widely used protocols. However, our methodology is easily extensible to SMB or NFSv4, and we plan to perform expanded studies in the future. In the case of NFSv3, both the client and the server can limit read- and write-transfer sizes and modify write-synchronization properties. Because the hypervisor and its NFS client significantly change I/O requests, it is not sufficient to collect data at the block layer of the guest OS; we collect our traces at the entrance to the NFS server.

After the request is sent over a network to the NAS server, the same layers that appear in the guest OS are repeated in the server. By this time, however, the original requests have already undergone significant changes performed by the upper layers, so the optimizations applied by similar layers at the server can be considerably different. Moreover, many NAS servers (e.g., NetApp [20]) run a proprietary OS that uses specialized request-handling algorithms, additionally complicating the overall system behavior. This complex behavior has a direct effect on measurement techniques, as we discuss next in Section 2.3.

2.3 VM-NAS Benchmarking Setup

Regular file system benchmarks usually operate at the application layer and generate workloads typical to one or a set of applications (Figure 2). In non-virtualized deployments these benchmarks can be used without any changes to evaluate the performance of a NAS server, simply by running the benchmark on a NAS client. In virtualized deployments, however, I/O requests can change significantly before reaching the NAS server due to the deep and diverse I/O stack described above. Therefore, benchmarking these environments is not straightforward.

One approach to benchmarking in a VM-NAS setup is to deploy the entire virtualization infrastructure and then run regular file system benchmarks inside the VMs. In this case, requests submitted by application-level benchmarks will naturally undergo the appropriate changes while passing through the virtualized I/O stack. However, this method requires a cumbersome setup of hypervisors, VMs, and applications. Every change to the test configuration, such as an increase in the number of VMs or a change of a guest OS, requires a significant amount of work. Moreover, the approach limits evaluation to the available test hardware, which may not be sufficient to run hypervisors with the hundreds of VMs that may be required to exercise the limits of the NAS server.

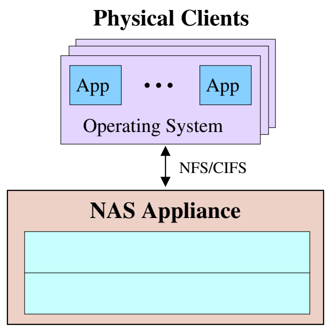
To avoid these limitations and regain the flexibility of standard benchmarks, we have created *virtualized benchmarks* by extracting the workload characteristics *after* the requests from the original *physical benchmarks* have passed though the virtualization and NFS layers. The generated benchmarks can then run directly against the NAS server without having to deploy a complex infrastructure. Therefore, the benchmarking procedure remains the same as before—easy, flexible, and accessible.

One approach to generating virtualized benchmarks would be to emulate the changes applied to each request as it goes down the layers. However, doing so would require a thorough study of the request-handling logic in the guest OSes and hypervisors, with further verification through multi-layer trace collection. Although this approach might be feasible, it is time-consuming, especially because it must be repeated for many different OSes and hypervisors. Therefore, in this paper we chose to study the workload characteristics at a single layer, namely where requests enter the NAS server. We collected traces at this layer and then characterized selected workload properties. The information from a single layer is enough to create the corresponding NAS benchmarks by reproducing the extracted workload features. Workload characterization and the benchmarks that we create are tightly coupled with the configuration of the upper layers: application, guest OS, local file system, and hypervisor. In the future, we plan to perform a sensitivity analysis of I/O stack configurations to deduce the parameters that account for the greatest changes to the I/O workload.

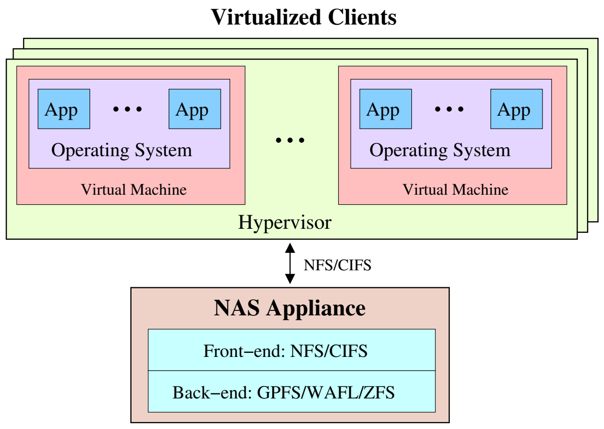
3 NAS Workload Changes

In this section we detail seven categories of NAS workload changes caused by virtualization. Specifically, we compare the two cases where a NAS server is accessed by a (1) *physical*; or (2) a *virtualized* client, and describe the differences in the I/O workload. These changes are the result of migrating an application from a physical server, which is configured to use an NFS client for direct data access, to a VM that stores data in a disk image file that the hypervisor accesses from an NFS server. Figure 3 demonstrates the difference in the two setups, and Table 2 summarizes the changes we observed in the I/O workload. The changes are listed from the most noticeable and significant to the least. Here, we discuss the changes qualitatively; quantitative observations are presented in Section 4.

First, and unsurprisingly, the number and size of files stored in NAS change from many relatively small files to a few (usually just one) large file(s) per VM—the disk image file(s). For example, the default Filebench file server workload defines 10,000 files with an average size of 128KB, which are spread over 500 directories. However, when Filebench is executed in a VM, there is only one large disk image file. (Disk image files are usually sized to the space requirements of a particular application; in our setup the disk image file size was set to the default 16GB for the Linux VM, and to 50GB for the Windows VM, because the benchmark we used in Windows required at least 50GB.) For the same reason, directory depth decreases and becomes fairly consistent: VMware ESX typically has a flat namespace; each VM has one directory with the disk image files stored inside it. Back-end file systems used in NAS are often optimized for common file sizes and directory depths [2, 30, 31, 34], so this workload change can significantly affect their performance. For example, to improve write performance for small files, one popular technique is to store data in the inode [16], a feature that would be wasted on virtualized clients. Further, disk image files in NAS environments are typically sparse, with large portions of the files unallocated, i.e., the physical file size can be much smaller than its logical size. In fact, VMware's vSphere—the main tool for managing the VMs in VMware-based infrastructures—supports only the creation of sparse disk images over NFS. A major implication of this change is that back-



(a) Physical



(b) Virtualized

Figure 3: Physical and Virtualized NAS architectures. With physical clients, applications use a NAS client to access the NAS appliance directly. With virtualized clients, applications access the NAS appliance via a virtualized block device.

#	Workload Property	Physical NAS Clients	Virtual NAS Clients
1	File and directory count	Many files and directories	Single file per VM
	Directory tree depth	Often deeply nested directories	Shallow and uniform
	File size	Lean towards many small files	Multi-gigabyte sparse disk image files
2	Meta-data operations	Many (72% in SPECsfs2008)	Almost none
3	I/O synchronization	Asynchronous and synchronous	All writes are synchronous
4	In-file randomness	Workload-dependent	Increased randomness due to guest file system encapsulation
	Cross-file randomness	Workload-dependent	Cross-file access replaced by in-file access due to disk image files
5	I/O Sizes	Workload-dependent	Increased or decreased due to guest file system fragmentation and I/O stack limitations
6	Read-modify-write	Infrequent	More frequent due to block layer in guest file system
7	Think time	Workload-dependent	Increased because of virtualization overheads

Table 2: Summary of key I/O workload changes between Physical and Virtualized NAS architectures.

end file systems for NAS can lower their focus on optimizing, for example, file append operations, and instead focus on improving the performance of block allocation within a file.

The second change caused by the move to virtualization is that all file system meta-data operations become data operations. For example, with a physical client there is a one-to-one mapping between file creation and a `CREATE` over the wire. However, when the application creates a file in a VM, the NAS server receives a series of writes to a corresponding disk image: one to a directory block, one to an inode block, and possibly one or more to data blocks. Similarly, when an application accesses files and traverses the directory tree, physical clients send many `LOOKUP` procedures to a NAS server. The same application behavior in a VM produces a sequence of `READ`s to the disk image. Current NAS benchmarks generate a high number of meta-data operations (e.g., 72% for SPECsfs2008), and will bias the evaluation of a NAS that serves virtualized clients. While it may appear that removing all meta-data operations implies that application benchmarks can generally be replaced with random I/O benchmarks, such as IOzone [11], this is insufficient. As shown in Section 5, the VM-NAS I/O stack generates a range of I/O sizes, jump distances, and request offsets that cannot be modeled with a simple distribution (uniform or otherwise).

Third, all write requests that come to the NAS server are synchronous. For NFS, this means that the *stable* attribute is set on each and every write, which is typically not true for physical clients. The block layers of many OSes expect that when the hardware reports a write completion, the data has been saved to persistent storage. Similarly, the NFS protocol's stable attribute specifies that the NFS server cannot reply to a `WRITE` until the data is persistent. So the hypervisor satisfies the guest OS's expectation by always setting this attribute on `WRITE` requests. Since many modern NAS servers try to improve performance by gathering write requests into larger chunks in RAM, setting the stable attribute invalidates this important optimization for virtualized clients.

Fourth, in-file randomness increases significantly with virtualized clients. On a physical client, access patterns (whether sequential or random) are distinct on a per-file basis. However, in virtualized clients, both sequential and random operations are blended into a single disk image file. This causes the NAS server to receive what appears to be many more random reads and writes to that file. Furthermore, guest file system fragmentation increases image file randomness. On the other hand, cross-file randomness decreases, as each disk image file is typically accessed by only a single VM; i.e., it can be easier to predict which files will be accessed next based on their status, and to differentiate them by how actively they are used (running VMs, stopped ones, etc.).

Fifth, the I/O sizes of original requests can both decrease and increase while passing through the virtualization layers. Guest file systems perform reads and writes in units of their block size, often 4KB. So, when reading a file of, say, 6KB size, the NAS server observes two 4KB reads for a total of 8KB, while a physical client would request only 6KB (25% less). Since many modern systems operate with a lot of small files [31], this difference can have a significant impact on bandwidth. Similarly, when reading 2KB of data from two consecutive data blocks in a file (1KB in each block), the NAS server may observe two 4KB reads for a total of 8KB (one for each block), while a physical NAS client may send only a single 2KB request. A NAS server designed for a virtualized environment could optimize its block-allocation and fragmentation-prevention strategies to take advantage of this observation.

Interestingly, I/O sizes can also *decrease* because guest file systems sometimes split large files into blocks that might not be adjacent. This is especially true for aged file systems with higher fragmentation [37]. Consequently, whereas a physical client might pass an application's 1MB read directly to the NAS, a virtualized client can sometimes submit several smaller reads scattered across the (aged) disk image. An emulated disk controller driver can also reduce the size of an I/O request. For example, we observed that the Linux IDE driver has a maximum I/O size of 128KB, which means that any application requests larger than this value will be split into smaller chunks. Note that such workload changes happen even in a physical machine as requests flow from a file system to a physical disk. However, in a VM-NAS setup, the transformed requests hit not a real disk, but a file on NAS, and as a result the NAS experiences a different workload.

The sixth change is that when an application writes to part of a block, the guest file system must perform a read-modify-write (RMW) to first read in valid data prior to updating and writing it back to the NAS server. Consequently, virtualized clients often cause RMWs to appear on the wire [19], requiring two block-sized round trips for every update. With physical clients, the RMW is generally performed at the NAS server, avoiding the need to first send valid data back to the NAS client.

Parameter	**RHEL 6.2**	**Win 2008 R2 SP1**
No. of CPUs	1	
Memory	1GB	2GB
Host Controller	Paravirtual	LSI Logic Parallel
Disk Drive Size	16GB	50GB
Disk Image Format	Thick flat VMDK	
Guest File System	Ext3	NTFS
Guest I/O Scheduler	CFQ	n/a

Table 3: Virtual Machine configuration parameters.

Seventh, the think time between I/O requests can increase due to varying virtualization overhead. It has been shown that for a single VM and modern hardware, the overhead of virtualization is small [4]. However, as the number of VMs increases, the contention for computational resources grows, which can cause a significant increase in the request inter-arrival times. Longer think times can prevent a NAS device from filling the underlying hardware I/O queues and achieving peak throughput.

In summary, both static and dynamic properties of NAS workloads change when virtualized clients are introduced into the infrastructure. The changes are sufficiently significant that direct comparison of certain workload properties between virtual and physical clients becomes problematic. For example, cross-file randomness has a rather different meaning in the virtual client, where the number of files is usually one per VM. Therefore, in the rest of the paper we focus solely on characterizing workloads from virtualized clients, without trying to compare them directly against the physical client workload. However, where possible, we refer to the original workload properties.

4 VM-NAS Workload Characterization

In this section we describe our experimental setup and then present and characterize a set of four different application-level benchmarks.

4.1 Experimental Configuration

Every layer in the VM-NAS I/O stack can be configured in several ways: different guest OSes can be installed, various virtualization solutions can be used, etc. The way in which the I/O stack is assembled and configured can significantly change the resulting workload. In the current work we did not try to evaluate every possible configuration, but rather selected several representative setups to demonstrate the utility of our techniques. The methodology we have developed is simple and accessible enough to evaluate many other configurations. Table 3 presents the key configuration options and parameters we used in our experiments. Since our final goal is to create NAS benchmarks, we only care about the settings of the layers above the NAS server; we treat the NAS itself as a black box.

We used two physical machines in our experimental setup. The first acted as a *NAS server*, while the second represented a typical *virtualized client* (see Figure 3). The hypervisor was installed on a Dell PowerEdge R710 node with an Intel Xeon E5530 2.4GHz 4-core CPU and 24GB of RAM. We used local disk drives in this machine for the hypervisor installation—VMware ESXi 5.0.0 build 62386. We used two guest OSes in the virtual setup: Red Hat Enterprise Linux 6.2 (RHEL 6.2) and Windows 2008 R2 SP1. We stored the OS's VM disk images on the local, directly attached disk drives. We conducted our experiments with a separate virtual disk in every VM, with the corresponding disk images being stored on the NAS. We pre-allocated all of the disk images (thick provisioning) to avoid performance anomalies across runs related to thin provisioning (e.g., delayed block allocations). The RHEL 6.2 distribution comes with a paravirtualized driver for VMware's emulated controller, so we used this controller for the Linux VM. We left the default format and mount options for guest file systems unchanged.

The machine designated as the NAS server was a Dell PowerEdge 1800 with six 250GB Maxtor 7L250S0 disk drives connected through a Dell CERC SATA 1.5/6ch controller, intended to be used as a storage server in enterprise environments. It is equipped with an Intel Xeon 2.80GHz Irwindale single-core CPU and 512MB of memory. The NAS server consisted of both the Linux NFS server and IBM's General Parallel File System (GPFS) version 3.5 [35]. GPFS is a scalable clustered file system that enables a scale-out, highly-available NAS solution and is used in both virtual and non-virtual environments. Our workload characterization and benchmark synthesis techniques treat NAS servers as a black box and are valid regardless of its underlying hardware and software. Since our ultimate goal is to create benchmarks capable of stressing any NAS, we did not characterize NAS-specific characteristics such as request latencies. Our benchmarks, however, let us manually configure the think time. By decreasing think time (along with increasing the number of VMs), a user can scale the load to the processing power of a NAS to accurately measure its peak performance.

4.2 Application-Level Benchmarks

In the Linux VM we used Filebench [15] to generate file system workloads. Filebench can emulate the I/O patterns of several enterprise applications; we used the File-, Web-, and Database-server workloads. We scaled up the datasets of these workloads so that they were larger than the amount of RAM in the VM (see Table 4).

Because Filebench does not support Windows, in our Windows VM we used JetStress 2010 [23], a disk-subsystem benchmark that generates a Microsoft Exchange Mail-server workload. It emulates accesses to the Exchange database by a specific number of users, with a corresponding number of log file updates. Complete workload configurations (physical and virtualized), along with all the software we developed as part of this project are available from https://avatar.fsl.cs.sunysb.edu/groups/t2mpublic/.

Workload	Dataset size	Files	R/W/M ratio	I/O Size
File-server	2.0GB	20,000	1/2/3	WF
Web-server	1.6GB	100,000	10/1/0	WF
DB-server	2.0GB	10	10/1/0	2KB
Mail-server	24.0GB	120	1/2/0	32KB

Table 4: High-level workload characterization for our benchmarks. R/W/M is the Read/Write/Modify ratio. WF (Whole-File) means the workload only reads or writes complete files. The mail-server workload is based on JetStress, for which R/W/M ratios and I/O sizes were estimated based on [24].

Although SPECsfs is a widely used NAS benchmark [38], we could not use it in our evaluation because it incorporates its own NFS client, which makes it impossible to run against a regular POSIX interface. We hope that the workload analysis and proposed benchmarks presented in this paper can be used by SPEC for designing future SPECsfs synthetic workloads.

VMware's VMmark is a benchmark often associated with testing VMs [45]. However, this benchmark is designed to evaluate the performance of a hypervisor machine, not the underlying storage system. For example, VMmark is sensitive to how fast a hypervisor's CPU is and how well it supports virtualization features (such as AMD-V and Intel VT [1, 22]). However, these details of hypervisor configuration should not have a large effect on NAS benchmark results. Although VMmark also indirectly benchmarks the I/O subsystem, it is hard to distinguish how much the I/O component contributes to the overall system performance. Moreover, VMmark requires the installation of several hypervisors and additional software (e.g., Microsoft Exchange) to generate the load. Our goal is complementary: to design a realistic benchmark for the NAS that serves as the backend storage for a hypervisor like VMware.

Our goal in this project was to transform some of the already existing benchmarks to their virtualized counterparts. As such, we did not replay any real-world traces in the VMs. Both Filebench and JetStress generate workloads whose statistical characteristics remain the same over time (i.e., stationary workloads). Consequently, new virtualized benchmarks also exhibit this property.

4.3 Characterization

We executed all benchmarks for 10 minutes (excluding the preparation phase) and collected NFS traces at the NAS server. We repeated every run 3 times and verified the consistency of the results. The traces were collected

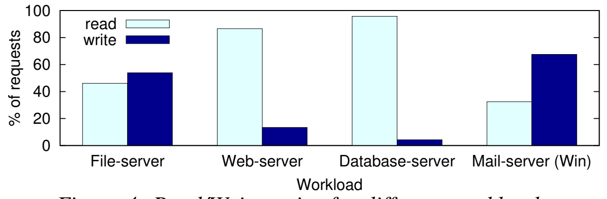


Figure 4: Read/Write ratios for different workloads

using the GPFS *mmtrace* facility [21] and then converted to the DataSeries format [5] for efficient analysis.

We developed a set of tools for extracting various workload characteristics. There is always a nearly infinite number of characteristics that can be extracted from a trace, but a NAS benchmark needs to reproduce only those that significantly impact the performance of NAS servers. Since there is no complete list of workload characteristics that impact NAS, in the future we plan to conduct a systematic study of NASes to create such a list. For this paper, we selected characteristics that clearly affect most NASes: (1) read/write ratio; (2) I/O size; (3) jump (seek) distance; and (4) offset popularity.

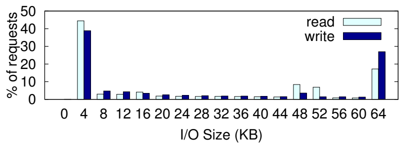
As we mentioned earlier, the workloads produced by VMs contain no meta-data operations. Thus, we only characterize the ratio of data operations—`READ`s to `WRITE`s. The jump distance of a request is defined as the difference in offsets (block addresses) between it and the immediately preceding request (accounting for I/O size as well). We do not take the operation type into account when calculating the jump distance. The offset popularity is a histogram of the number of accesses to each block within the disk image file; we report this as the number of blocks that were accessed once, twice, etc. We present the offset popularity and I/O size distributions on a per-operation basis. Figure 4 depicts the read/write ratios and Figures 5–8 present I/O size, jump distance, and offset popularity distributions for all workloads. For jump distance we show a CDF because it is the clearest way to present this parameter.

Read/Write ratio. Read/write ratios vary significantly across the analyzed workloads. The File-server workload generates approximately the same number of reads and writes, although the original workload had twice as many writes (Table 4). We attribute this difference to the high number of meta-data operations (e.g., `LOOKUP`s and `STAT`s) that were translated to reads by the I/O stack. The Web-server and the Database-server are read-intensive workloads, which is true for both original and virtualized workloads. The corresponding original workloads do not contain many meta-data operations, and therefore the read/write ratio remained unchanged (unlike the File-server workload). The Mail-server workload, on the other hand, is write-intensive: about 70% of all operations are writes, which is close to the original benchmark where two thirds of all operations are writes. As with the Web-server and Database-server workloads, the lack of meta-data operations kept the read/write ratio unchanged,

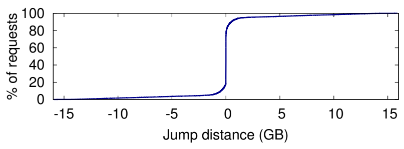
I/O size distribution. The I/O sizes for all workloads vary from 512B to 64KB; the latter limit is imposed by the RHEL 6.2 NFS server, which sets 64KB as the default maximum NFS read and write size. All requests smaller than 4KB correspond to 0 on the bar graphs. There are few writes smaller than 4KB for the File-server and Web-server workloads, but for the Database- and Mail-server (JetStress) workloads the corresponding percentages are 80% and 40%, respectively. Such small writes are typical for databases (Microsoft Exchange emulated by JetStress also uses a database) for two reasons. First, the Database-server workload writes 2KB at a time using direct I/O. In this case, the OS page cache is bypassed during write handling, and consequently the I/O size is not increased to 4KB (the page size) when it reaches the block layer. The block layer cannot then merge requests, due to their randomness. Second, databases often perform operations synchronously by using the `fsync` and `sync` calls. This causes the guest file system to atomically update its meta-data, which can only be achieved by writing a single sector (512B) to the virtual disk drive (and hence over NFS).

For the File-server and Web-server workloads, most of the writes happen in 4KB and 64KB I/O sizes. The 4KB read size is dominant in all workloads because this is the guest file system block size. However, many of the File-server's reads were merged into larger requests by the I/O scheduler and then later split into 64KB sizes by the NFS client. This happens because the average file size for the File-server is 128KB, so whole-file reads can be merged. For the Web-server workload, the average file size is only 16KB, so there are no 64KB reads at all. For the same reason, the Web-server workload exhibits many reads around 16KB (some files are slightly smaller, others are slightly larger, in accordance with Filebench's gamma distribution [47]). Interestingly, for the Mail-server workload, many requests have non-common I/O sizes. (We define an I/O size as non-common if fewer than 1% of such requests have such I/O size.) We grouped all non-common I/O sizes in the bucket called "Rest" in the histogram. This illustrates that approximately 15% of all requests have non-common I/O sizes for the Mail-server workload.

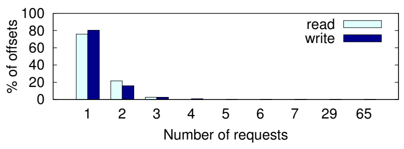
Jump distance. The CDF jump distance distribution graphs show that many workloads demonstrate a significant level of sequentiality, which is especially true for the File-server workload: more than 60% of requests are sequential. Another 30% of the requests in the File-server workload represent comparatively short jumps: less than 2GB, the size of the dataset for this workload; these are jumps between different files in the active dataset. The remaining 10% of the jumps come from



(a) I/O Size

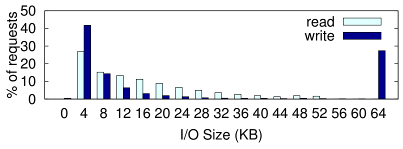


(b) Jump distance

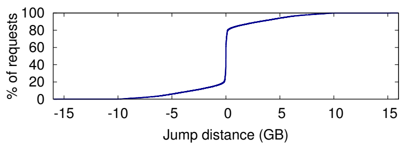


(c) Offset popularity

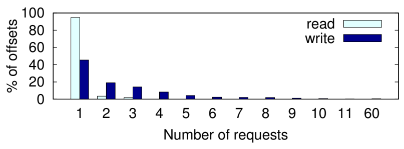
Figure 5: Characteristics of a virtualized File-server workload.



(a) I/O Size

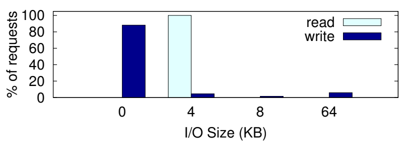


(b) Jump distance

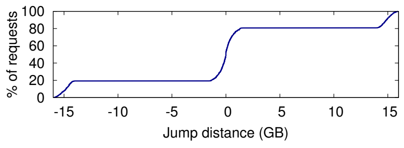


(c) Offset popularity

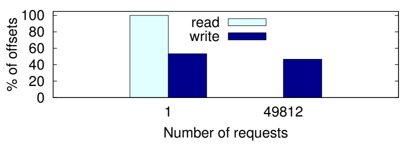
Figure 6: Characteristics of a virtualized Web-server workload.



(a) I/O Size

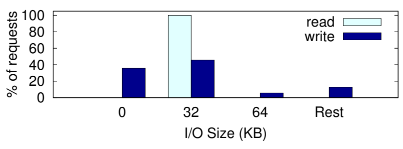


(b) Jump distance

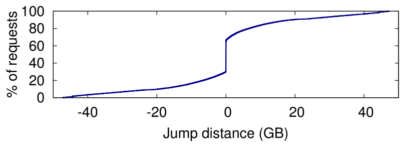


(c) Offset popularity

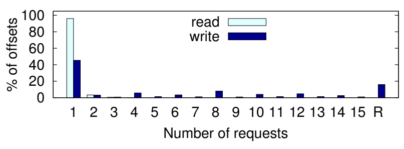
Figure 7: Characteristics of a virtualized Database-server workload.



(a) I/O Size



(b) Jump distance



(c) Offset popularity

Figure 8: Characteristics of a virtualized Mail-server workload.

meta-data updates and queries, and are spread across the entire disk. The Web-server workload exhibits similar behavior except that the active dataset is larger—about 5–10GB. The cause of this is a larger number of files in the workload (compared to File-server) and the allocation policy of Ext3 that tries to spread many files across different block groups.

For the Database-server workload there are almost no sequential accesses. Over 60% of the jumps are within 2GB because that is the dataset size. Interestingly, about 40% of the requests have fairly long jumps that are caused by frequent file system synchronization, which leads to meta-data updates at the beginning of the disk.

In the Mail-server workload approximately 40% of the requests are sequential, and the rest are spread across the 50GB disk image file. A slight bend around 24GB corresponds to the active dataset size. Also, note that the Mail-server workload uses the NTFS file system, which uses a different allocation policy than Ext3; this explains the difference in the shape of the Mail-server curve from other workloads.

Offset popularity. In all workloads, most of the offsets were accessed only once. The absolute numbers on these graphs depend on the run time, e.g., when one runs a benchmark longer, then the chance of accessing the same offset increases. However, the shape of the curve remains the same as time progresses (although it shifts to the right). For the Database workload, 40% of all blocks were updated several thousand times. We attribute this to the repeated updates of the same file system meta-data structures due to frequent file system synchronization. The Mail-server workload demonstrates a high number of overwrites (about 50%). These overwrites are caused by Microsoft Exchange overwriting the log file multiple times. With Mail-server, "R" on the X axes designates the "Rest" of the values, because there were too many to list. We therefore grouped all of the values that contributed less than 1% into the R bucket.

5 New NAS Benchmarks

This section describes our methodology for the creation of new NAS benchmarks for virtualized environments and then evaluates their accuracy.

5.1 Trace-to-Model Conversion

Our NAS benchmarks generate workloads with characteristics that closely follow the statistical distributions presented in Section 4.3. We decided not to write a new benchmarking tool, but rather exploit Filebench's ability to express I/O workloads with its Workload Modeling Language (WML) [48], which allows one to flexibly define processes and the I/O operations they perform. Filebench interprets WML and translates its instructions to corresponding POSIX system calls. Our use of Filebench will facilitate the adoption of our new virtualized benchmarks: existing Filebench users can easily run new WML configurations.

We extended the WML language to support two virtualization terms: *hypervisor* and *vm* (virtual machine). We call the extended version WML-V (by analogy with AMD-V). WML-V is backwards compatible with the original WML, so users can merge virtualized and non-virtualized configurations to simultaneously emulate the workloads generated by both physical and virtual clients.

For each analyzed workload—File-server, Web-server, Database-server and Mail-server—we created a corresponding WML-V configuration file. By modifying these files, a user can adjust the workloads to reflect a desired benchmarking scenario, e.g., defining the number of VMs and the workloads they run.

Listing 1 presents an abridged example of a WML-V configuration file that defines a single hypervisor, which runs 5 Database VMs and 2 Web-server VMs. *Flowops* are Filebench's defined I/O operations, which are mapped to POSIX calls, such as `open`, `create`, `read`, `write`, and `delete`. In the VM case, we only use read and write flowops, since meta-data operations do not appear in the virtualized workloads. For every defined VM, Filebench will pre-allocate a disk image file of a user-defined size—16GB in the example listing.

```
HYPERVISOR name="physical-host1" {
 VM name="dbserver-vm",dsize=16gb,instances=5 {
  flowop1, ...
 }
 VM name="websever-vm",dsize=16gb,instances=2 {
  flowop1, ...
 }
}
```

Listing 1: An abridged WML-V workload description that defines 7 VMs: 5 run database workloads and 2 generate Web-server workloads.

Filebench allows one to define random variables with desired empirical distributions; various flowop attributes can then be assigned to these random variables. We used this ability to define read and write I/O-size distributions and jump distances. We achieved the required read/write ratios by putting an appropriate number of read and write flowops within the VM definition. The generation of a workload with user-defined jump distances and offset popularity distributions is a complex problem [28] that Filebench does not solve; in this work, we do not attempt to emulate this parameter. However, as we show in the following section, this does not significantly affect the accuracy of our benchmarks.

Ideally, we would like Filebench to translate flowops directly to NFS procedures. However, this would require us to implement an NFS client within Filebench (which is an ongoing effort within the Filebench community). To work around this limitation, we mount NFS with the `sync` flag and `open` the disk image files with the O_DIRECT flag, ensuring that I/O requests bypass the Linux page cache. These settings also ensure that (1) no additional read requests are performed to the NFS server (readahead); (2) that all write requests are immediately sent to the NFS server without modification; and (3) that replies are returned only after the data is on disk. This behavior was validated with extensive testing. This approach works well in this scenario because we do not need to generate meta-data procedures on the wire; that would be difficult to achieve using this method because a 1:1 mapping of meta-data operations does not exist between system calls and NFS procedures.

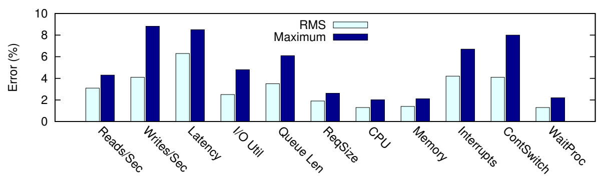
Our enhanced Filebench reports aggregate operations per second for all VMs and individually for each VM. Operations in the case of virtualized benchmarks are different from the original non-virtualized equivalent: our benchmarks report the number of reads and writes per second; application-level benchmarks, however, report application-level operations (e.g., the number of HTTP requests serviced by a Web-server). Nevertheless, the numbers reported by our benchmarks can be directly used to compare the performance of different NAS servers under a configured workload.

None of our original benchmarks, except the database workload, emulated think time, because our test was designed as an I/O benchmark. For the database benchmark we defined think time as originally defined in Filebench—200,000 loop iterations. Think time in all workloads can be adjusted by trivial changes to the workload description.

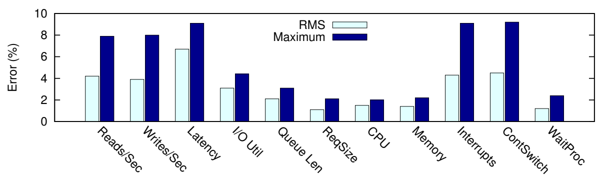
5.2 Evaluation

To evaluate the accuracy of our benchmarks we observed how the NAS server responds to the virtualized benchmarks as compared to the original benchmarks when executed in a VM. We monitored 11 parameters that represent the response of a NAS and are easy to extract through the Linux */proc* interface: (1) Reads/second from the underlying block device; (2) Writes/second; (3) Request latency; (4) I/O utilization; (5) I/O queue length; (6) Request size; (7) CPU utilization; (8) Memory usage; (9) Interrupt count; (10) Context-switch count; and (11) Number of processes in the wait state. We call these *NAS response parameters*.

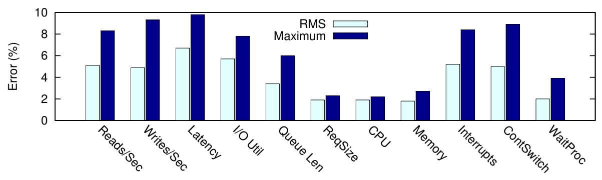
We sampled the response parameters every 30 seconds during a 10-minute run and calculated the relative difference between each pair of parameters. Figure 9 presents maximum and Root Mean Square (RMS) difference we observed for four workloads. In these experiments a single VM with an appropriate workload was



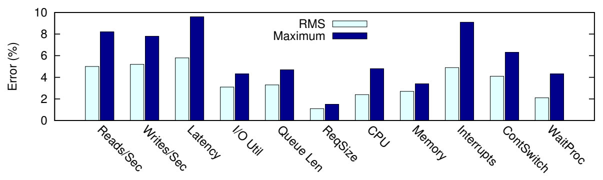
(a) File-server



(b) Web-server

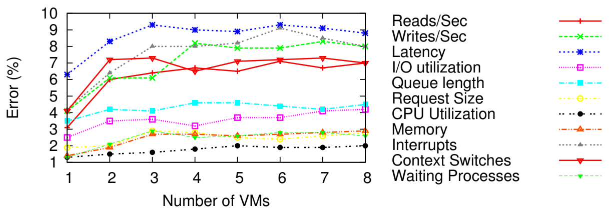


(c) Database-server

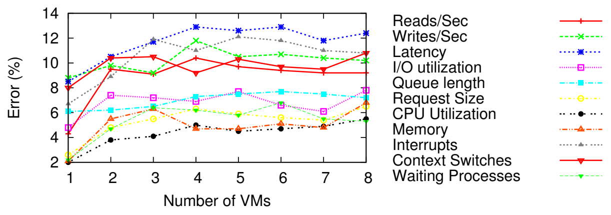


(d) Mail-server (Windows, JetStress)

Figure 9: Root Mean Square (RMS) and maximum relative distances of response parameters for all workloads.



(a) RMS distance



(b) Maximum

Figure 10: Response parameter errors depending on the number of VMs deployed. The first four VMs (1–4) execute four different workloads we analyzed. The next four VMs (5–8) are repeated in the same order.

used. The maximum relative error of our benchmarks is always less than 10%, and the RMS distance is within 7% across all parameters. Certain response parameters show especially high accuracy; for example, the RMS distance for request size is within 4%. Here, the accuracy is high because our benchmarks directly emulate I/O size distribution. Errors in CPU and memory utilization were less than 5%, because the NAS in our experiments did not perform many CPU-intensive tasks.

Scalability with Multiple Virtual Machines. The benefit of our benchmarks is that a user can define many VMs with different workloads and measure NAS performance against this specific workload configuration. To verify that the accuracy of our benchmarks does not decrease as we emulate more VMs, we conducted a multi-VM experiment. We first ran one VM with a File-server in it, then added a second VM with a Web-server workload, then a third VM executing the Database-server workload, and finally a fourth VM running JetStress. After that we added another four VMs with the same four workloads in the same order. In total we had 8 different configurations ranging from 1 to 8 VMs; this setup was designed to heavily stress the NAS under several, different, concurrently running workloads. We then emulated the same 8 configurations using our benchmarks and again monitored the response parameters. Figures 10(a) and 10(b) depict RMS and maximum relative errors, respectively, depending on the number of VMs.

When a single VM is emulated, our benchmarks show the best accuracy. Beyond one VM, the RMS error increased by about 3–5%, but still remained within 10%. For four parameters—latency, writes/sec, interrupts and context switches count—the maximum error observed during the whole run was the highest among other parameters—in the 10–13% range.

In summary, our benchmarks show a high accuracy for both single- and multi-VM experiments, even under heavy stress.

6 Related Work

Storage performance in virtualized environments is an active research area. Le et al. studied the storage performance implications of combining different guest and host file systems [29]. Boutcher et al. examined how the selection of guest OS and host I/O schedulers impacts the performance of a virtual machine [10]. Both of these works focused on the performance aspects of the problem, not workload characterization or generation; also, the authors used direct-attached storage, which is simpler but less common in modern enterprise data centers.

Hildebrand et al. discussed the implications of using the VM-NAS architecture with enterprise storage servers [19]. That work focused on the performance implications of the VM-NAS I/O stack without thoroughly investigating the changes to the I/O workload. Gulati et al. characterized the SAN workloads produced by VMs for several enterprise applications [17]. Our techniques can also be used to generate new benchmarks for SAN-based deployments, but we selected to investigate VM-NAS setups first, for two reasons. First, NAS servers are becoming a more popular solution for hosting VM disk images. Second, the degree of workload change in such deployments is higher: NAS servers use more complex network file-system protocols whereas SANs and DAS use a simpler block-based protocol.

Ahmad et al. studied performance overheads caused by I/O stack virtualization in ESX with a SAN [4]. That study did not focus on workload characterization but rather tried to validate that modern VMs introduce low overhead compared to physical nodes. Later, the same authors proposed a low-overhead method for on-line workload characterization in ESX [3]. However, their tool characterizes traces collected at the virtual SCSI layer and consequently does not account for any transformations that may occur in ESX and its NFS client. In contrast, we collect the trace at the NAS layer after all request transformations, allowing us to create more accurate benchmarks.

Casale et al. proposed a model for predicting storage performance when multiple VMs use shared storage [12, 26]. Practical benchmarks like ours are complementary to that work and allow one to verify such predictions in real life. Ben-Yehuda et al. analyzed performance bottlenecks when several VMs are used to provide different functionalities on a storage controller [7]. The authors focused on lowering network overhead via intelligent polling and other techniques.

Trace-driven performance evaluation and workload characterization have been the basis of many studies [14, 25, 27, 33]. Our trace-characterizing techniques and benchmark-synthesis techniques are based on multi-dimensional workload analysis. Chen et al. used multi-dimensional trace analysis to infer behavior of enterprise storage systems [13]. Tarasov et al. proposed a technique for automated translation of block-I/O traces to workload models [40]. Yadawakar et al. proposed to discover applications based on multi-dimensional characteristics of NFS traces [49].

In summary, to the best of our knowledge, there have been no earlier studies that systematically analyzed virtualized NAS workloads. Moreover, we are the first to present new NAS benchmarks that accurately generate virtualized I/O workloads.

7 Conclusions and Future Work

We have studied the transformation of existing NAS I/O workloads due to server virtualization. Whereas such transformations were known to occur due to virtualization, they have not been studied in depth to date. Our analysis revealed several significant I/O workload changes due to the use of disk images and the placement of the guest block layer above the NAS file client. We observed and quantified significant changes such as the disappearance of file system meta-data operations at the NAS layer, changes in I/O sizes, changes in file counts and directory depths, asynchrony changes, increased randomness within files, and more.

Based on these observations from real-world workloads, we developed new benchmarks that accurately represent NAS workloads in virtualized data centers—and yet these benchmarks can be run directly against the NAS without requiring a complex virtualization environment configured with VMs and applications. Our new virtualized benchmarks represent four workloads, two guest operating systems, and up to eight virtual machines. Our evaluation reveals that the relative error of these new benchmarks across more than 11 parameters is less than 10% on average. In addition to providing a directly usable measurement tool, we hope that our work will provide guidance to future NAS standards, such as SPEC, in devising benchmarks that are better suited to virtualized environments.

Future work. We plan to extend the number of generated benchmarks by analyzing actual applications and application traces, including typical VM operations such as booting, updating, and snapshotting—and examine root and I/O swap partition access patterns. We also expect to explore more VM configuration options such as additional guest file systems (and their age), hypervisors, and NAS protocols. Once a larger body of virtual NAS benchmarks exists, we will be able to study the I/O workload's sensitivity to each configuration parameter as well as investigate the impact of extracting and reproducing additional trace characteristics in the generated benchmarks. To avoid manual analysis and transformation of large numbers of applications, we plan to investigate the feasibility of automatically transforming physical workloads to virtual workloads via a multi-level trace analysis of the VM-NAS I/O stack.

Acknowledgements. We thank the anonymous USENIX reviewers for their valuable feedback. This work was supported in part by NSF awards CCF-0937833 and CCF-0937854, and an IBM PhD Fellowship.

References

[1] Advanced Micro Devices, Inc. Industry leading virtualization platform efficiency, 2008. *www.amd.com/virtualization*.

[2] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch. A five-year study of file-system metadata. In *Proceedings of the Fifth USENIX Conference on File and Storage Technologies (FAST '07)*, 2007.

[3] I. Ahmad. Easy and efficient disk I/O workload characterization in VMware ESX server. In *Proceedings of IEEE International Symposium on Workload Characterization (IISWC)*, 2007.

[4] 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 IEEE International Symposium on Workload Characterization (IISWC)*, 2003.

[5] E. Anderson, M. Arlitt, C. Morrey, and A. Veitch. DataSeries: an efficient, flexible, data format for structured serial data. *ACM SIGOPS Operating Systems Review*, 43(1), January 2009.

[6] Atlantis Computing. *www.atlantiscomputing.com/*.

[7] M. Ben-Yehuda, M. Factor, E. Rom, A. Traeger, E. Borovik, and B. Yassour. Adding advanced storage controller functionality via low-overhead virtualization. In *Proceedings of the Tenth USENIX Conference on File and Storage Technologies (FAST '12)*, 2012.

[8] Thomas Bittman. *Virtual machines and market share through 2012*. Gartner, October 2009. ID Number: G00170437.

[9] Thomas Bittman. *Q&A: six misconceptions about server virtualization*. Gartner, July 2010. ID Number: G00201551.

[10] D. Boutcher and A. Chandra. Does virtualization make disk scheduling passé? In *Proceedings of the 1st Workshop on Hot Topics in Storage (HotStorage '09)*, 2009.

[11] D. Capps. IOzone file system benchmark. *www.iozone.org*.

[12] G. Casale, S. Kraft, and D. Krishnamurthy. A model of storage I/O performance interference in virtualized systems. In *Proceedings of the International Workshop on Data Center Performance (DCPerf)*, 2011.

[13] Y. Chen, K. Srinivasan, G. Goodson, and R. Katz. Design implications for enterprise storage systems via multi-dimensional trace analysis. In *Proceedings of the 23rd ACM Symposium on Operating System Principles (SOSP '11)*, 2011.

[14] D. Ellard, J. Ledlie, P. Malkani, and M. Seltzer. Everything you always wanted to know about NFS trace analysis, but were afraid to ask. Technical Report TR-06-02, Harvard University, Cambridge, MA, June 2002.

[15] Filebench. *http://filebench.sourceforge.net*.

[16] G. R. Ganger and M. F. Kaashoek. Embedded inodes and explicit grouping: exploiting disk bandwidth for small files. In *Proceedings of the Annual USENIX Technical Conference*, 1997.

[17] A. Gulati, C. Kumar, and I. Ahmad. Storage workload characterization and consolidation in virtualized environments. In *Proceedings of 2nd International Workshop on Virtualization Performance: Analysis, Characterization, and Tools (VPACT)*, 2009.

[18] A. Gulati, G. Shanmuganathan, X. Zhang, and P. Varman. Demand based hierarchical QoS using storage resource pools. In *Proceedings of the Annual USENIX Technical Conference*, 2012.

[19] D. Hildebrand, A. Povzner, R. Tewari, and V. Tarasov. Revisiting the storage stack in virtualized NAS environments. In *Proceedings of the Workshop on I/O Virtualization (WIOV '11)*, 2011.

[20] D. Hitz, J. Lau, and M. Malcolm. File system design for an NFS file server appliance. In *Proceedings of the USENIX Winter Technical Conference*, 1994.

[21] IBM. General Parallel File System problem determination guide. Technical Report GA22-7969-02, IBM, December 2004. *http://pic.dhe.ibm.com/infocenter/db2luw/v9r8/index.jsp?topic=\%2Fcom.ibm.db2.luw.sd.doc\%2Fdoc\%2Ft0056934.html*.

[22] Intel Corporation. Intel virtualization technology (Intel VT), 2008. *www.intel.com/technology/virtualization/*.

[23] Microsoft Exchange Server JetStress 2010. *www.microsoft.com/en-us/download/details.aspx?id=4167*.

[24] Understanding database and log performance factors. *http://technet.microsoft.com/en-us/library/ee832791.aspx*.

[25] 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 (IISWC)*, 2008.

[26] S. Kraft, G. Casale, D. Krishnamurthy, D. Greer, and P. Kilpatrick. Performance models of storage contention in cloud environments. *Software and Systems Modeling*, 12(2), March 2012.

[27] G. H. Kuenning, G. J. Popek, and P. Reiher. An analysis of trace data for predictive file caching in mobile computing. In *Proceedings of the Summer 1994 USENIX Conference*, 1994.

[28] Z. Kurmas, J. Zito, L. Trevino, and R. Lush. Generating a jump distance based synthetic disk access

pattern. In *Proceedings of the International IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, 2006.

[29] D. Le, H. Huang, and H. Wang. Understanding performance implications of nested file systems in a virtualized environment. In *Proceedings of the Tenth USENIX Conference on File and Storage Technologies (FAST '12)*, 2012.

[30] A. W. Leung, S. Pasupathy, G. Goodson, and E. L. Miller. Measurement and analysis of large-scale network file system workloads. In *Proceedings of the USENIX Annual Technical Conference (ATC '08)*, 2008.

[31] D. Meyer and W. Bolosky. A study of practical deduplication. In *Proceedings of the Nineth USENIX Conference on File and Storage Technologies (FAST '11)*, 2011.

[32] OpenStack Foundation. *www.openstack.org/*.

[33] J. Ousterhout, H. Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson. A trace-driven analysis of the UNIX 4.2 BSD file system. In *Proceedings of the Tenth ACM Symposium on Operating System Principles (SOSP)*, 1985.

[34] D. Roselli, J. R. Lorch, and T. E. Anderson. A comparison of file system workloads. In *Proceedings of the Annual USENIX Technical Conference*, 2000.

[35] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of the First USENIX Conference on File and Storage Technologies (FAST '02)*, 2002.

[36] Simplivity. *www.simplivity.com/*.

[37] K. A. Smith and M. I. Seltzer. File system aging—increasing the relevance of file system benchmarks. In *Proceedings of the 1997 International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 1997)*, 1997.

[38] SPEC. SPECsfs2008. *www.spec.org/sfs2008*, July 2008.

[39] J. Sugerman, G. Venkitachalam, and B. Lim. Virtualizing I/O devices on VMware workstations hosted virtual machine monitor. In *Proceedings of the Annual USENIX Technical Conference*, 2001.

[40] V. Tarasov, K. S. Kumar, J. Ma, D. Hildebrand, A. Povzner, G. Kuenning, and E. Zadok. Extracting flexible, replayable models from large block traces. In *Proceedings of the Tenth USENIX Conference on File and Storage Technologies (FAST '12)*, 2012.

[41] Tintri. *www.tintri.com/*.

[42] VMware vCloud. *http://vcloud.vmware.com/*.

[43] Richard Villars and Noemi Greyzdorf. *Worldwide file-based storage 2010–2014 forecast update*. IDC, December 2010. IDC #226267.

[44] VirtualBox. *https://www.virtualbox.org/*.

[45] VMMark. *www.vmware.com/go/vmmark*.

[46] VMware, Inc. *VMware Virtual Machine File System: Technical Overview and Best Practices*, 2007. *www.vmware.com/pdf/vmfs-best-practices-wp.pdf*.

[47] A. W. Wilson. Operation and implementation of random variables in Filebench.

[48] Filebench Workload Model Language (WML). *http://sourceforge.net/apps/mediawiki/filebench/index.php?title=Filebench_Workload_Language*.

[49] N. Yadwadkar, C. Bhattacharyya, and K. Gopinath. Discovery of application workloads from network file traces. In *Proceedings of the Eighth USENIX Conference on File and Storage Technologies (FAST '10)*, 2010.

[50] Natalya Yezhkova, Liz Conner, Richard L. Villars, and Benjamin Woo. *Worldwide enterprise storage systems 2010–2014 forecast: recovery, efficiency, and digitization shaping customer requirements for storage systems*. IDC, May 2010. IDC #223234.

[51] Y. Yu, D. Shin, H. Eom, and H. Yeom. NCQ vs I/O scheduler: preventing unexpected misbehaviors. *ACM Transaction on Storage*, 6(1), March 2010.