The Effectiveness of Deduplication on Virtual Machine Disk Images

Keren Jin
Storage Systems Research Center
University of California, Santa Cruz
kjin@cs.ucsc.edu

Ethan L. Miller
Storage Systems Research Center
University of California, Santa Cruz
elm@cs.ucsc.edu

ABSTRACT

Virtualization is becoming widely deployed in servers to efficiently provide many logically separate execution environments while reducing the need for physical servers. While this approach saves physical CPU resources, it still consumes large amounts of storage because each virtual machine (VM) instance requires its own multi-gigabyte disk image. Moreover, existing systems do not support *ad hoc* block sharing between disk images, instead relying on techniques such as overlays to build multiple VMs from a single "base" image.

Instead, we propose the use of deduplication to both reduce the total storage required for VM disk images and increase the ability of VMs to share disk blocks. To test the effectiveness of deduplication, we conducted extensive evaluations on different sets of virtual machine disk images with different chunking strategies. Our experiments found that the amount of stored data grows very slowly after the first few virtual disk images if only the locale or software configuration is changed, with the rate of compression suffering when different versions of an operating system or different operating systems are included. We also show that fixed-length chunks work well, achieving nearly the same compression rate as variable-length chunks. Finally, we show that simply identifying zero-filled blocks, even in ready-to-use virtual machine disk images available online, can provide significant savings in storage.

Categories and Subject Descriptors

D.4.3 [**Operating Systems**]: File Systems Management; D.4.8 [**Operating Systems**]: Performance

General Terms

measurement, performance

1. INTRODUCTION

In modern server farms, virtualization is being used to provide ever-increasing number of servers on virtual machines (VMs), reducing the number of physical machines required while preserving isolation between machine instances. This approach better utilizes server resources, allowing many different operating system instances to run on a small number of servers, saving both hardware acquisition costs and operational costs such as energy, management, and cooling. Individual VM instances can be separately managed, allowing them to serve a wide variety of purposes and preserving the level of control that many users want. However, this flexibility comes at a price: the storage required to hold hundreds or thousands of multi-gigabyte VM disk images, and the inability to share identical data pages between VM instances.

One approach saving disk space when running multiple instances of operating systems on multiple servers, whether physical or virtual, is to share files between them; *i. e.*, sharing a single instance of the `/usr/local` file via network mount. This approach is incompatible with VM disk images, however, since the internal file structure of a VM disk image is invisible to the underlying file system. Standard compression such as that provided by the Lempel-Ziv compression used in `gzip` [30], is ineffective because, while it can reduce the storage space used by a single disk image, it cannot eliminate commonalities between files.

Instead, others have proposed the use of deduplication to reduce the storage space required by the many different VM disk images that must be stored in a medium to large scale VM hosting facility [18]. While it seems clear that deduplication is a good approach to this problem, our research quantifies the benefits of using deduplication to reduce the storage space needed for multiple VM disk images. Our experiments also investigate *which* factors impact the level of deduplication available in different sets of VM disk images, some of which are under system control (*e. g.*, fixed versus variable-sized chunking and average chunk size) and some of which are dependent on the usage environment (*e. g.*, operating system version and VM target use). By quantifying the effects of these factors, our results provide guidelines for both system implementers and sites that host large numbers of virtual machines, showing which factors are important to consider and the costs of making design choices at both the system and usage level.

The paper is organized as follows. Section 2 reviews related works and background material on deduplication and virtual machines. Section 3 introduces our chunking and deduplication approaches to analyze VM disk images. Section 4 evaluates the effect of deduplication on sets of VM disk images for different purposes. Section 5 discusses directions for future work, and Section 6 summarizes our conclusions.

SYSTOR 2009 May 2009, Haifa, Israel

2. BACKGROUND

Our research studies the effectiveness of applying deduplication to virtual machine environments. In this section, we provide some background on both technologies.

2.1 Virtual Machines

Virtual machine monitors provide a mechanism to run multiple operating system instances on a single computer system. Systems such as Xen [4], VMware, and QEMU [5] provide a full execution environment to "guest" operating systems, using many techniques to convince each guest operating system that it has control of a full computer system.

One technique used by virtual machine monitors is the use of a file in the underlying "host" operating system to hold the contents of the guest's disk. These virtual disk images, whose size is specified when the virtual machine is created, can be either *flat* or *sparse*. Flat images are fixed-size files, with one block for each block in the guest's disk. Initially, unused blocks are zero-filled; thus, flat disk images tend to have a lot of zero-filled blocks, particularly if they are relatively large to allow the guest operating system space into which to store additional data. Sparse images, unlike flat images, only contain virtual disk blocks that have been written at least once. Thus, sparse images can occupy relatively little space when created, and only grow as the guest operating system uses more and more of its virtual disk. Note, however, that sparse images *can* contain zero-filled blocks, since the guest operating system may write a block containing all zeros; such a block would be stored by the virtual machine in the disk image file. While there is no global standard for virtual machine disk images, specifications for one file format is available from VMware [25].

2.2 Deduplication

Deduplication is an technology that can be used to reduce the amount of storage required for a set of files by identifying duplicate "chunks" of data in a set of files and storing only one copy of each chunk [17, 19]. Subsequent requests to store a chunk that already exists in the chunk store are done by simply recording the *identity* of the chunk in the file's inode or block list; by not storing the chunk a second time, the system stores less data, thus reducing cost.

Different implementations of deduplication use different techniques to break files into chunks. Fixed-size chunking, such as that used in Venti [20] simply divides files at block boundaries. Variable-size chunking, used in systems such as LBFS [17] and Deep Store [28], computes a Rabin fingerprint [21] or similar function across a sliding window to place chunk boundaries, resulting in blocks that may have different lengths. This approach typically provides better deduplication, since it is more resistant to insertion or deletion of a few bytes in the middle of a file; however, it may negatively impact performance by requiring non-block aligned I/O requests.

While their approaches to identifying chunks differ, both fixed-size and variable-size chunking use cryptographically-secure content hashes such as MD5 or SHA1 [2] to identify chunks, thus allowing the system to quickly discover that newly-generated chunks already have stored instances. Even for a 128 bit MD5 hash, the chance of a single collision among 10^{15} chunks—10^{18} bytes, assuming an average chunk size of 1 KB—is about 10^{-9} [14]. By using a 160 bit hash such as SHA1, we can reduce the probability of a *single* collision in an exabyte-scale chunk store to about 7×10^{-18}. Collisions in SHA1 have been reported [26], and intentionally forging different but still equivalently meaningful independent chunks is a potential security breach, though it is not yet practical [9]. While these two issues might prevent a system implementer from using SHA1 [13], we chose SHA1 to name chunks in our experiments because these issues clearly do not impact the statistical results gathered from our experiments—one or two "false collisions" would not significantly alter our findings.

Deduplication and similar technologies have already been used to reduce bandwidth and storage demands for network file systems [17, 1, 11], reduce the storage demands created by VM checkpoints [18], store less data in backup environments [8, 29, 20], and reduce storage demands in archival storage [28, 27]. Using deduplication in an online system requires fast identification of duplicate chunks; techniques such as those developed by Bhagwat, *et al.* [6] and Zhu, *et al.* [29] can help alleviate this problem. Moreover, Nath, *et al.* found that deduplication was sufficiently fast for storing checkpoints of VM images [18] and the Difference Engine [12] used deduplication to share in-memory pages between different virtual machine instances. While these uses are not identical to read-write sharing of VM disk image chunks, the relatively low performance overhead for other uses of deduplication in VM images suggests that a file system for VM disk images should be sufficiently fast to use in a production environment.

Several projects have investigated the use of deduplication in virtualization environments. Nath, *et al.* used content-addressable storage to store multiple checkpoints from a hosting center running multiple virtual machines [18]. Their experiments covered 817 checkins across 23 users and 36 virtual machines. However, this study created "gold images" for each operating system (Windows XP and Linux) upon which each VM was based; while the gold images received security patches, users did not have a choice of which Linux distribution to run. Our research explores a much wider range of Linux configurations, exposing the factors that affect deduplication. The Difference Engine project [12] used deduplication to allow multiple virtual machines to share in-memory pages. This approach proved effective, though the level of deduplication was lower for memory page deduplication than for storage deduplication. Moreover, the VMs were limited in sample size;in contrast, the virtual disks we studied showed a large amount of variation. Liguori and Van Hensbergen explored the use of content addressable storage (CAS) with virtual machines [16]. They examined overlap between pairs of VM disk images for both Linux and Windows XP, measuring the amount of deduplication possible between them. They then experimented with an implementation of CAS in Qemu, showing that Venti, a content-addressable store, performs worse than "vanilla" systems, though this may be due to Venti's inefficiency. Their deduplication results confirm some of our findings, but they do not exhaustively study different operating system characteristics and their impact on deduplication effectiveness.

3. METHODOLOGY

Since our experiments were focused on the amount of deduplication possible from sets of VM disk images, we first broke VM disk images into chunks, and then analyzed different sets of chunks to determine both the amount of dedu-

Name	Value
divisor	512, 1024, 2048, 4096 (bytes)
maximum chunk size	divisor × 2
minimum chunk size	divisor / 16
irreducible polynomial	0x91407E3C7A67DF6D
residual	0
sliding window size	minimum chunk size / 2

Table 1: Parameters for variable-size chunking.

plication possible and the *source* of chunk similarity. Section 3.1 discusses the techniques we used to generate chunks from VM disk images, and Section 3.2 discusses the approach we used to measure deduplication effectiveness.

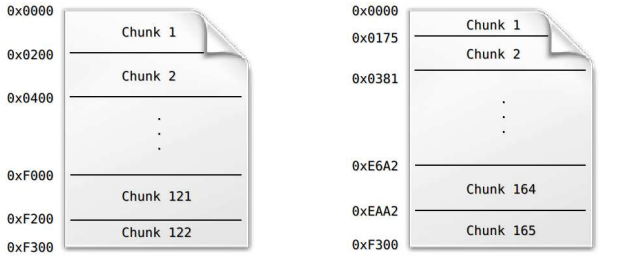
We use the term *disk image* to denote the logical abstraction containing all of the data in a VM, while *image files* refers to the actual files that make up a disk image. A disk image is always associated with a single VM; a *monolithic* disk image consists of a single image file, and a *spanning* disk image has one or more image files, each limited to a particular size, typically 2 GB. When we refer to "disk images," we are referring to multiple VM disk image files that belong to multiple distinct VMs. Finally, we use the term *chunk store* to refer to the system that stores the chunks that make up one or more disk images.

3.1 Chunking

In order to locate identical parts of disk images, we divide the image files into chunks to reduce their granularities. This is done by treating each image file as a byte stream and identifying boundaries using either a "constant" function (for fixed-size chunking) or a Rabin fingerprint (for variable-sized chunking). A chunk is simply the data between two boundaries; there are implicit boundaries at the start and end of each image file. Chunks are identified by their SHA1 hash, which is calculated by running SHA1 over the contents of the chunk. We assume that chunks with the same chunk ID are identical; we do not do a byte-by-byte comparison to ensure that the chunks are identical.

We implemented both fixed-size and variable-size chunking to test the efficiency of each approach in deduplicating disk images. Fixed-size chunking was done by reading an image file from its start and setting chunk boundaries every N bytes, where N is the chunk size. For variable-size chunking, we calculated 64-bit Rabin fingerprint using the irreducible polynomial from Table 1 for a fixed-size sliding window, and slid the window one byte at a time until the fingerprint modulo the divisor equals to the residual; at this point, a boundary was created at the start of the window, as shown in Figure 1. Of course, this approach results in chunks that may have greatly varying sizes, so we imposed minimum and maximum chunk sizes on the function to reduce the variability, as is usually done in real-world systems [28]. The specific parameters we used in variable-size chunking are shown in Table 1; these parameters were chosen to ensure that the chunking algorithm generates chunks with the desired average chunk size. We conducted experiments for average chunk sizes of 512, 1024, 2048, and 4096 bytes for both fixed-size and variable-size chunking.

We chunked each image file separately because fixed-size chunking exhibits the "avalanche effect": although altering bytes in the file only changes the corresponding chunk IDs, inserting or removing bytes before the end of an image file changes *all* of the remaining chunk IDs, unless the length of



(a) Fixed-size chunking with chunk size 512 B.

(b) Variable-size chunking with expected average chunk size 512 B.

Figure 1: Chunking a file with size 0xF300 bytes.

insertion or deletion is multiple of the chunk size for fixed-size chunking. Thus, if image file sizes are not multiples of the chunk size, the result of chunking across files is different than that of chunking each separately. Also, because both monolithic and spanning image files have a header specific to the VM instance, chunking sequentially across the spanning files does not restore the original guest file system because the last chunk of each file could be shorter than the specified chunk size.

Zero-filled chunks are common in VM disk images, and come from three sources. One source is VM-specific: disk images can contain zero blocks corresponding to space not yet used by the virtual machine. Another source is runs of zeroes in the file system, caused by space that has been zeroed by the operating system running in the VM. The third source is application-generated zero-filled blocks, as are sometimes generated by databases and other applications. The relative frequency of the three sources of zeroed blocks varies in different VMs. While the first source is VM-generated, different types of disk images (flat versus. sparse) can have different numbers of zero blocks in them. Decisions such as the maximum disk image size can influence this number as well. The other two sources of zero blocks are due to the guest operating system and applications; thus, they are less affected by the choice of virtual disk size. In fixed-size chunking, all zero-filled chunks are identical—they all contain identical content. In the variable-size chunking experiments, runs of zeros do not generate boundaries, and thus result in chunks of the maximum chunk size. Since all zero-filled chunks are the same (maximal) size (except perhaps for a run at the end of an image file), they are all identical to one another.

To further reduce space, we compressed each chunk using `zip` after hashing it to generate the chunk ID. This approach is often used with deduplication, and results in additional space saving with no loss of fidelity.

3.2 Deduplication

The deduplication process is simple: for each chunk being stored, attempt to locate an existing instance in the chunk store. If none is found, the new chunk is added to the chunk store; otherwise, the new chunk is a shared chunk. As described in Section 3.1, nearly all zero chunks are identical, except for a non-maximal length zero chunk at the end of an image file. Since even large spanning disk images have relatively few files, most of which end with non-zero chunks, an optimization that recognizes such non-maximal length zero chunks would provide little benefit.

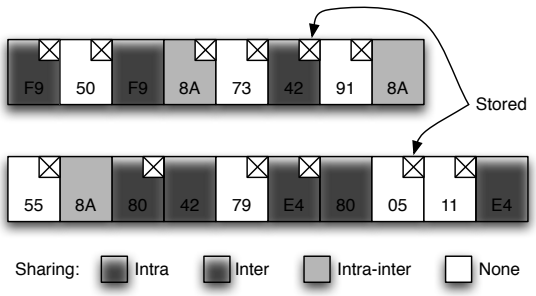


Figure 2: Share categories of chunks. Chunks seen for the first time must be stored; subsequent occurrences need not be stored.

The chunk ID, which is generated from the SHA1 hash of the chunk's content, is the only value used to look up existing chunks. Even for an exabyte-scale chunk store, collisions would be highly unlikely; for the multi-terabyte chunk store in our experiments, the chances of collision are even smaller. We calculate the deduplication ratio for a given chunk store and chunking method by:

$$1 - \frac{\textit{Stored bytes of all disk images}}{\textit{Original bytes of all disk images}}$$

The deduplication ratio is a fraction in $[0, 1)$, since there is at least one stored chunk in a chunk store, and the worst possible case is that there are no duplicate chunks. We exclude per-chunk overhead in our studies; this overhead is less than 4% for 512 B chunks, and smaller for larger chunks.

We classify each occurrence of a chunk into one of four categories, shown in Figure 2. When a chunk appears exactly once in the entire set of VM disk images, it is called an *unshared chunk*, labeled "none" in Figure 2. Chunks that appear in more than one disk image are termed *inter-image shared chunks*, and chunks that appear multiple times, within a single disk image but not elsewhere are called *intra-image shared chunks*. Chunks that appear in multiple disk images and appear more than once in at least one of those images are *inter-intra-image shared chunks*. Zero-filled chunks are tracked separately; however, they are typically inter-intra-image shared chunks in sets with more than one disk image because zero-filled chunks appear in every disk image that we examined.

As Figure 2 shows, all chunks must be stored the first time they are seen. Subsequent occurrences of each chunk are not stored, reducing the total storage space required to store the set of disk images. All stored chunks are grouped together for our experiments, while non-stored chunks are classified by the disk images in which other occurrences of the chunks are found. Thus, a chunk c that occurs one time in disk image A and then two times in disk image B would result in one *stored* chunk and two *inter-intra shared* chunks because there are occurrences of chunk c in two *separate* disk images, and multiple occurrences of chunk c in at least one disk image. The total size of a chunk store before deduplication is thus the sum of the sizes of the stored chunks and three shared categories. This notation differs from the metrics used by another study [16], which only concentrates on "duplicate" chunks. In their study, two identical disk images would be 100% similar, and 40% chunks would typically need to be stored.

Changing the processing order for a set of disk images can produce different intermediate results for the number and type of shared chunks and the deduplication ration, but the final result will always be the same for a given set of disk images. For example, processing disk images in the order $\langle A1, A2, B\rangle$ would result in a high level of inter-image sharing after the second disk image was added, assuming that images $A1$ and $A2$ are very similar and both dissimilar to image B. However, processing the files in the order $\langle A1, B, A2\rangle$ would result in a much lower deduplication ratio after the second disk image was added, but the final result would be the same as for the first processing order.

4. EXPERIMENTS

To determine which factors affect deduplication ratios for sets of disk images, we first downloaded the pre-made disk images listed in Table 2 from Internet sites including VMware's Virtual Appliance Marketplace [24], Thoughtpolice [23], and bagvapp's Virtual Appliances [3] site. Most of the VMs were created in VMware's format, and were compatible with VMware Server 2.0. The VirtualBox VMs were compatible with VirtualBox 2.0.

For the figures in this section, *stored* chunks are counted the first time seen during the deduplication process, each of which must be stored. Each of the other chunk categories is reduced in size by its remaining instances. Zero chunks are isolated as a separate chunk class, not part of inter-intra shared chunk.

4.1 Overall Impacts

Before going into detail on how much specific factors impact deduplication, we evaluated deduplication for closely related disk images—a set of images all based on Ubuntu 8.04 LTS—and compared it to a set of disk images from widely divergent installations, including BSD, Linux, and OpenSolaris VMs. The results are shown in Figures 3 and 4. As these figures show, deduplication of operating systems with similar kernel versions and packaging systems deduplicate extremely well, with the system able to reduce storage for the 14 disk images by over 78% and 71% for 1 KB and 4 KB chunk size, respectively. Of the 22% of the chunks that are stored, 12% are chunks that occur exactly once and 10% are chunks that occur more than once, with only a single instance stored. Given this trend, it is likely that additional Ubuntu 8.04 disk images would add little additional operating system data. It is important to note that these images were pre-compiled and gathered from various sources; they were not created by a single coordinated entity, as was done by Nath, *et al.* [18], suggesting that hosting centers can allow users to install their own VMs and still gain significant savings by using deduplication. Since it takes more time and storage overhead to generate smaller chunks, 4 KB chunk size might be a good idea if space allows.

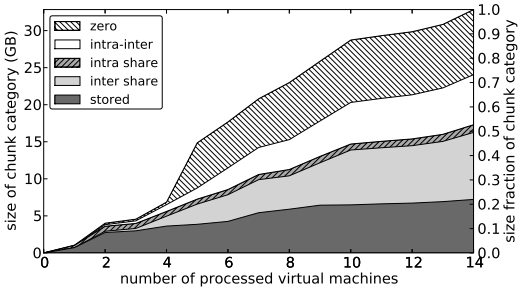
On the other hand, a set of disk images consisting of 13 operating system images including various versions of BSD, OpenSolaris, and Linux did not fare as well, as Figure 4 shows. In such a case, unique data is the largest category, and deduplication saves less than half of the total space, with zero-filled blocks second. This is close to the worst-case scenario for deduplication, since the disk images differed in every possible way, including operating system and binary format. That this approach was able to achieve a space savings of close to 50% is encouraging, suggesting that adding

Index	Name and version	Kernel	File system	Desktop	Image size	Disk type
1	Arch Linux 2008.06	Linux 2.6.25-ARCH	ext3	GNOME	3.5G	XS
2	CentOS 5.0	Linux 2.6.18-8.el5	ext3	None	1.2G	XS
3	CentOS 5.2	Linux 2.6.18-92.1.10.el5	ext3	GNOME	3.3G	MS
4	DAMP	Dragonfly 1.6.2-RELEASE	ufs	None	1.1G	MF
5	Darwin 8.0.1	Darwin 8.0.1	HFS+	None	1.5G	MF
6	Debian 4.0.r4	Linux 2.6.18-6-486	ext3	None	817M	XS
7	Debian 4.0	Linux 2.6.18-6-686	ext3	GNOME	2.5G	MS
8	DesktopBSD 1.6	FreeBSD 6.3-RC2	ufs	KDE	8.1G	XF
9	Fedora 7	Linux 2.6.21-1.3194.fc7	ext3	GNOME	2.9G	XS
10	Fedora 8	Linux 2.6.23.1-42.fc8	ext3	GNOME	3.4G	XS
11	Fedora 9 en-US	Linux 2.6.25-14.fc9.i686	ext3	GNOME	3.4G	XS
12	Fedora 9 fr	Linux 2.6.25-14.fc9.i686	ext3	GNOME	3.6G	XS
13	FreeBSD 7.0	FreeBSD 7.0-RELEASE	ufs	None	1.2G	XS
14	Gentoo 2008.0	Linux 2.6.24-gentoo-r8	ext3	Xfce	5.5G	XS
15	Gentoo 2008.0 with LAMP	Linux 2.6.25-gentoo-r7	ext3	None	8.1G	XF
16	Knoppix 5.3.1	Linux 2.6.24.4	ext3	KDE	13G	XS
17	Kubuntu 8.04.1	Linux 2.6.24-19-generic	ext3	KDE	2.6G	MS
18	Mandriva 2009.0	Linux 2.6.26.2-desktop-2mnb	ext3	GNOME	3.3G	XS
19	NAMP	NetBSD 3.1 (GENERIC)	ffs	None	1.1G	MF
20	OAMP	OpenBSD 4.0	ffs	None	804M	MS
21	OpenBSD 4.3	OpenBSD 4.3	ffs	None	558M	MS
22	OpenSolaris 2008.5	SunOS 5.11	ZFS	GNOME	3.8G	XS
23	openSUSE 11.0	Linux 2.6.25-1.1-pae	ext3	KDE	8.1G	MF
24	PC-BSD 1.5	FreeBSD 6.3-RELEASE-p1	ufs	KDE	2.2G	MS
25	Slackware 12.1	Linux 2.6.24.5-smp	ext3	KDE	3.5G	MS
26	Ubuntu 8.04 en-US	Linux 2.6.24-19-generic	ext3	GNOME	3.5G	MS
27	Ubuntu 8.04 fr	Linux 2.6.24-19-generic	ext3	GNOME	2.5G	XS
28	Ubuntu 8.04 JeOS	Linux 2.6.24-16-virtual	ext3	None	293M	MS
29	Ubuntu 6.10 Server	Linux 2.6.17-10-server	ext3	None	520M	XS
30	Ubuntu 7.04 Server	Linux 2.6.20-15-server	ext3	None	557M	XS
31	Ubuntu 7.10 Server	Linux 2.6.22-14-server	ext3	None	543M	XS
32	Ubuntu 8.04 Server	Linux 2.6.24-16-server	ext3	None	547M	XS
33	Ubuntu 8.04 LTS (1)	Linux 2.6.24-16-server	ext3	GNOME	3.0G	XS
34	Ubuntu 8.04 LTS (2)	Linux 2.6.24-16-server	ext3	GNOME	2.9G	MS
35	Ubuntu 8.04 LTS (3)	Linux 2.6.24-16-server	ext3	GNOME	2.2G	XS
36	Ubuntu 8.04 LTS (4)	Linux 2.6.24-16-server	ext3	GNOME	2.9G	MS
37	Ubuntu 8.04 LTS (5)	Linux 2.6.24-16-server	ext3	GNOME	2.9G	MS
38	Ubuntu 8.04 LTS (6)	Linux 2.6.24-16-server	ext3	None	547M	XS
39	Ubuntu 8.04 LTS (7)	Linux 2.6.24-16-server	ext3	GNOME	2.1G	XS
40	Ubuntu 8.04 LTS (8)	Linux 2.6.24-16-server	ext3	None	1.1G	MS
41	Ubuntu 8.04 LTS (9)	Linux 2.6.24-16-server	ext3	None	559G	MS
42	Ubuntu 8.04 LTS (10)	Linux 2.6.24-16-server	ext3	GNOME	3.2G	MS
43	Ubuntu 8.04 LTS (11)	Linux 2.6.24-16-server	ext3	None	604M	MS
44	Ubuntu 8.04.1 LTS (12)	Linux 2.6.24-16-server	ext3	GNOME	2.4G	MS
45	Ubuntu 8.04.1 LTS (13)	Linux 2.6.24-16-server	ext3	GNOME	8.1G	XF
46	Ubuntu 8.04.1 LTS (14)	Linux 2.6.24-16-server	ext3	None	1011M	XS
47	Xubuntu 8.04	Linux 2.6.24-16-generic	ext3	Xfce	2.3G	XS
48	Zenwalk 5.2b	Linux 2.6.25.4	ext3	Xfce	2.5G	XS
49	Ubuntu 8.04 Server on VMware	Linux 2.6.24-16-server	ext3	None	1011M	MS
50	Ubuntu 8.04 Server on VirtualBox	Linux 2.6.24-16-server	ext3	None	969M	MS
51	Ubuntu 8.04 Server on VMware	Linux 2.6.24-16-server	ext3	None	4.1G	XF
52	Ubuntu 8.04 Server on VirtualBox	Linux 2.6.24-16-server	ext3	None	4.1G	MF

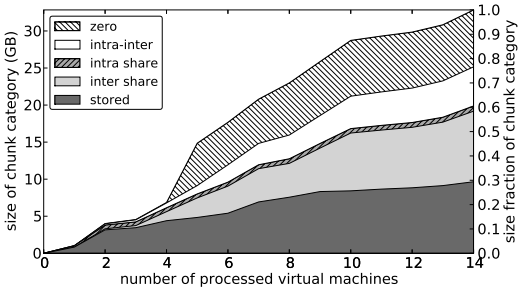
Table 2: Virtual machine disk images used in the study. Under disk type, *M* is a monolithic disk image file, *X* is a disk image file split into 2 GB chunks, *S* is a sparse image file, and *F* is a flat image file. For example, *MS* would correspond to a single monolithic disk image formatted as a sparse image.

further VMs will result in more space savings, though not as good as for the case in which all VMs are highly similar.

In our next experiment, we compared deduplication levels for chunk stores consisting only of VMs with a single operating system—BSD or Linux—to that of two more heterogeneous chunk stores: one with BSD, OpenSolaris, and Darwin (labeled "Unix"), and another with all types of operating systems (labeled "All"). The *All* chunk store is a super set of the other three chunk stores, and contains more flat disk images. As Figure 5 shows, the *All* chunk store achieved the best deduplication ratio, indicating that, even for operating systems of different lineage, there is redundancy available to be removed by deduplication. This seemingly contradictory high deduplication level comes from two sources. The first is zero chunks, as the *All* chunk store includes more flat disk images than any other chunk stores. The second is additional inter chunks, as they appeared only once in *BSD* or *Linux* chunk stores and were considered as stored chunks. Nevertheless, the higher levels of sharing for non-zero chunks in the *Linux* chunk store indicates that the Linux disk images were more homogeneous than the BSD images and *All* images.



(a) Average chunk size = 1 KB.



(b) Average chunk size = 4 KB.

Figure 3: Growth of data in different categories for 14 different Ubuntu 8.04 LTS instances. Stored data grows very slowly after the second VM is integrated. In these figures, stored data includes the first instance of each chunk, regardless of how it is shared. The sharp increase in size at the 5th VM is due to the use of a flat disk image, in which there are a large number of zero-filled (empty) sectors.

Another notable feature in Figure 5 is that the fraction of zero chunks decreases markedly from the 512 B Linux case to the 4 KB Linux case. While the overall number of chunks decreases by a factor of 8 (4096/512), the fraction of chunks that contain all zeros shrinks, with a corresponding increase in unique chunks, indicating that most of the zero chunks are 2 KB or less in length. The figure also shows that, while the relative fractions of zero chunks and unique chunks changes from 512 B chunks to 4 KB chunks, the amount of sharing changes relatively little, indicating that chunk size may have relatively little impact on relative frequency of shared chunks.

Figure 6 shows the cumulative distribution of chunks by both count and total size in the *Linux* chunk store. Chunks that appear only once are *unique* chunks and must be stored; for other chunks, the chunk store need only contain one copy of the chunk. As the figure shows, over 70% of chunks occur exactly once, and these chunks make up about 35% of the undeduplicated storage. The zero chunks take about 20% storage, as shown in Figure 5; they are particularly common in flat disk images.

4.2 Impact of Specific Factors

Our next set of experiments examine many factors that might affect deduplication, identifying those that are most critical and quantifying their effects on the deduplication ratio.

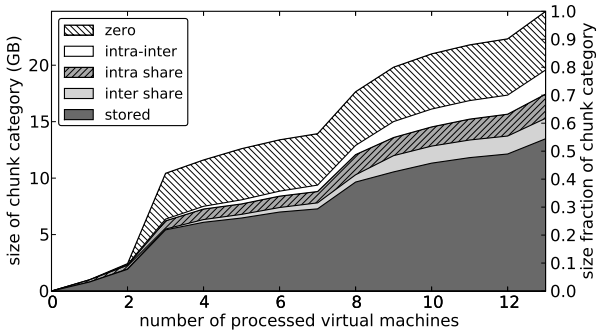


Figure 4: Growth of category data for 13 Unix and Linux virtual machines, using variable size chunking, with an average chunk size of 1 KB. Unique data grows significantly as each disk image is added. As in Figure 3, the sharp increase in size at the 3rd VM is due to a large number of empty, zero-filled sectors in a flat disk image.

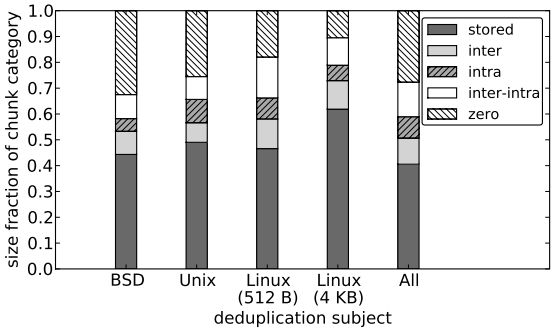


Figure 5: Effects of varying operating system type on deduplication. All experiments used 512 B variable-size chunks, except for the Linux experiment that uses 4 KB variable-size chunks. There is more intra-inter sharing in Linux than in BSD, indicating that the former is more homogeneous.

We first examined the effect of chunk size and boundary creation technique on deduplication ratio. Figure 7 shows the effect of deduplicating a set of disk images for different versions of Ubuntu Server using fixed and variable-size chunking for different average chunk sizes. As expected, smaller chunk sizes result in better deduplication ratios; this is done by converting unique chunks into shared chunks and zero chunks. Interestingly, the number of zero chunks grows significantly as chunk size decreases, indicating that zero chunks are more likely to be generated by the operating system or applications than by the VM software writing out the virtual disk, since zeros in the virtual disk would likely be 4 KB or larger. It is also important to note that fixed-size chunking is even more effective than variable-size chunking in this experiment, suggesting that fixed-size chunking may be appropriate for VM disk images.

We next examined the effects of different releases on deduplication effectiveness. Figure 8 shows the deduplication ratios for consecutive and non-consecutive releases of Ubuntu and Fedora. *Ubuntu78* is deduplicated against Ubuntu 7.10 and Ubuntu 8.04, and *Ubuntu68* is deduplicated against

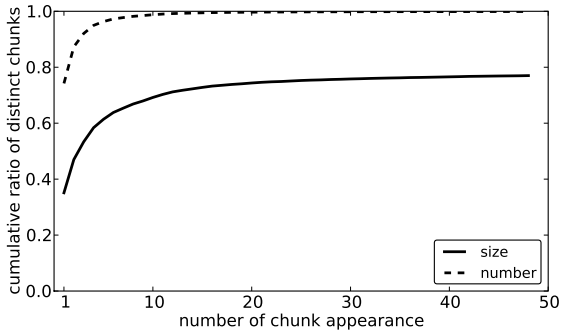


Figure 6: Cumulative distribution of chunks by count and total size. The upper line shows the cumulative number of chunks with total count of n or less, and the lower line shows the total space that would be consumed by chunks with count n or less. The data is from the *Linux* chunk store with chunk size 512 B. As the graph shows, most chunks occur fewer than 14 times, and few non-zero chunks appear more than 50 times in the set of disk images.

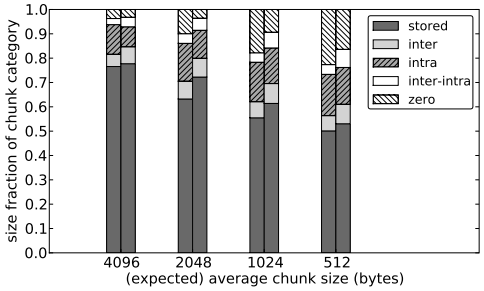


Figure 7: Effects of chunk size and fixed versus. variable-size chunking on deduplication for a set of disk images including Ubuntu Server 6.10, 7.04, 7.10 and 8.04. In each group of two bars, the left bar measures fixed-size chunking and the right bar indicates variable-size chunking. The graph shows that smaller chunk sizes result in more effective deduplication.

Ubuntu 6.10 and Ubuntu 8.04. *Fedora89* is deduplicated against Fedora 8 and Fedora 9, while *Fedora79* is deduplicated against Fedora 7 and Fedora 9. Deduplication patterns between consecutive and non-consecutive releases of a single distribution appear similar, and deduplication is only slightly less effective when skipping a release. This experiment shows that, when a mature operating system such as Ubuntu updates its major version, most of the data, *e. g.*, base system or software architecture, remains unchanged. Another interesting point from Figure 8 is that variable-size chunking does much better on Fedora than does fixed-size chunking, in large part because it is able to convert unique blocks into inter-intra shared blocks. We do not know why this is; however, it is one of the only cases in which variable-size chunking significantly outperforms fixed size chunking in our experiments, confirming that deduplication for VMs should use fixed-size chunking rather than adding the complexity of variable-size chunking.

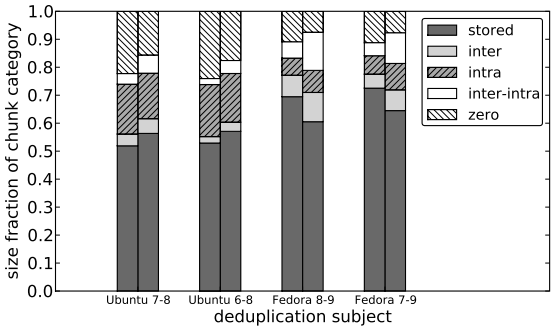


Figure 8: Deduplication on different releases of a single Linux distribution. The left bar in each pair is fixed-size chunking and right bar is variable-size chunking; chunk size is 512 B. Consecutive releases only have slightly lower deduplication ratio than non-consecutive releases.

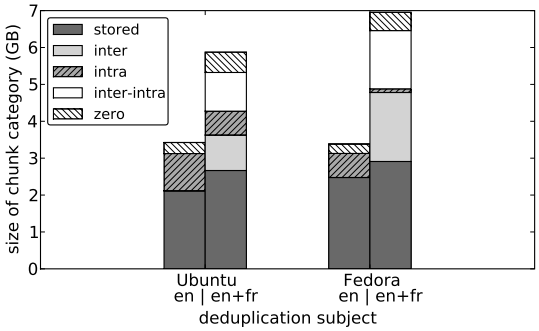


Figure 9: Locale versions (variable-size chunking, chunk size 512 B). The left bar in each pair only measures the English version and the right bar measures both English and French versions. Storing different locale versions of same distribution produces high deduplication.

Figure 9 shows the impact of OS locale upon deduplication. We deduplicated English and French versions of Ubuntu Server 8.04 and Fedora 9 separately; the only differences between the two versions were a few files relating to software interfaces and keyboard layouts. As the figure shows, deduplicating the French version of each OS against its English version adds few stored chunks, indicating that changing the localization of an operating system introduces very few unique blocks.

We next evaluated the effect of deduplicating Linux versions that derive from a common root. Ubuntu and Knoppix are both based on Debian, and Fedora, CentOS, and Mandriva descend from Red Hat Linux. The result, shown in Figure 10, is surprising: despite their common lineage, Linux versions derived from a single root do not deduplicate well against each other. While mature releases do not change much, as Figure 8 showed, there are significant changes when a new Linux distribution "forks off." Thus, it is necessary to consider these distributions as distinct ones when considering deduplication effectiveness.

We next evaluated the effects of varying the operating system while keeping the purpose of the distribution—in this

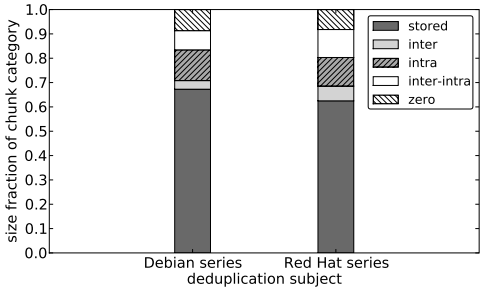


Figure 10: Distribution lineage. Debian series comprises Debian, Ubuntu and Knoppix. Red Hat series comprises Fedora, CentOS and Mandriva. Variable-size chunking, chunk size 512 B. Despite the common lineages, there is a relatively low level of deduplication.

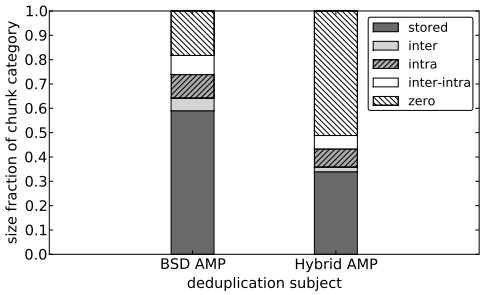
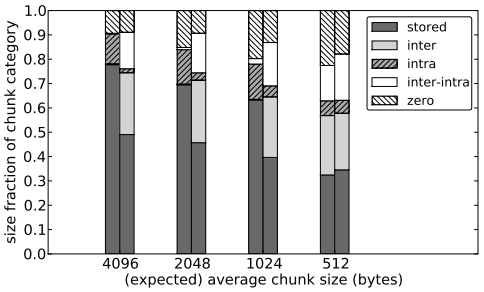


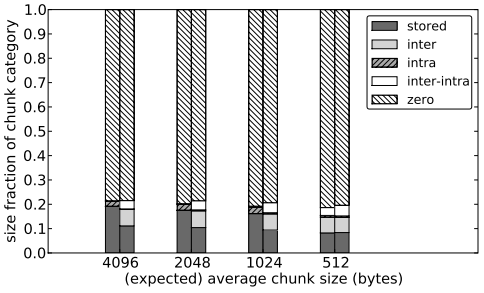
Figure 11: Deduplication of Web appliance VMs (variable-size chunking, chunk size 512 B). *AMP* stands for Apache, MySQL and PHP. Since the deduplication ratios for both cases are low when zero chunks are excluded, we can conclude that diverse operating systems with similar goals do not have many identical chunks of code.

case, serving Web pages—constant. We built a chunk store from disk images of DragonflyBSD, NetBSD and OpenBSD with Apache, MySQL, PHP and all dependent packages, as described in Section 4.3. We then added similar configurations of CentOS, Gentoo and Ubuntu into the chunk store; the results are shown in Figure 11. Excluding the large number of zero chunks, the deduplication ratios are not high, showing that the common purpose of the systems does not improve the deduplication ratio. While it may appear that diversifying the chunk store with Linux distributions helps, this is an illusion caused by the large number of zero chunks; the levels of sharing remain low.

The choice of virtual machine monitor (VMM) does not significantly change the virtual disk image, as Figure 12 shows. The same virtual machine, Ubuntu Server 8.04.1, on VirtualBox 2.0 and VMware 2.0 deduplicates extremely well, saving about half of the space, as long as variable-size chunking is used. The large variation in deduplication effectiveness for fixed-size chunking is due to different offsets in the disk image for the actual disk blocks (the first part of the file is occupied by the VMM-specific header). Since the offset of the actual virtual disk data in the two disk images files have the same value modulo 512 but not modulo 1024, fixed-size chunking is only effective for 512 B chunks. Figure 12b



(a) Ubuntu Server 8.04.1 on VMware and VirtualBox (sparse disk image, 4 GB maximum)



(b) Ubuntu Server 8.04.1 on VMware and VirtualBox (flat disk image, preallocate 4 GB space)

Figure 12: Virtual machines created by different VMMs deduplicate well. The bar on the left measures fixed-size chunking, and the right-hand bar measures variable-size chunking.

shows that, not surprisingly, preallocated disk images contain a lot of zeros. While most VMMs provide support for sparse images, it may be useful to push this functionality from the VMM into the file system if other deduplication is also implemented there.

4.3 Impact of Package Management

Users of modern Unix-like operating systems can modify their functionality by installing and removing software packages as well as by storing user-related data. In this section, we experimented with the impact on deduplication effectiveness of adding and removing packages. The packages used in the experiments are described in Table 4; all experiments were conducted on a fresh Ubuntu Server 8.04 and CentOS 5.2 installation, which use `aptitude` and `yum`, respectively, for package management. Table 3 lists the sizes of the VMs and the packages.

Figure 13 shows the deduplication effectiveness for different sets of disk images with differing installation orders. Deduplication tests on these images show two important facts. First, installation order is relatively unimportant for deduplication; all cases have very high deduplication ratios, indicating that each disk image in a pair is nearly identical to the other. Second, increasing the number of packages installed reduces the level of deduplication but, again, installation order is relatively unimportant. This is an important finding for hosting centers; it shows that hosting centers can allow users to install their own packages without fear that it will negatively impact the effectiveness of deduplication. Again, this shows that it is not necessary to base disk images

(a) Package set 1. Some packages only have a `deb` version or an `rpm` version.

Index	Package name		Software detail	Application
	Ubuntu	CentOS		
1	apache2 apache2-doc	httpd httpd-manual	Apache 2.2	web server
2	php5 libapache2-mod-php5	php	PHP 5.2.4	language
3	mysql-server mysql-doc-5.0	mysql-server	MySQL Server 5.0	database
4	exim4	exim	Exim MTA 4.69	email service
5	mediawiki php5-gd	mediawiki	MediaWiki 1.11.2	wiki
6	vsftpd	vsftpd	Very Secure FTP 2.0.6	FTP server
7	subversion libapache2-svn	subversion	Subversion 1.4.6	version control system
8	bacula	bacula-client bacula-console bacula-director bacula-storage	Bacula 2.2.8	backup
9	rpm	-	RPM 4.4.2.1	package manager

(b) Package set 2

Index	Package name	Software detail	Application area
1	lighttpd lighttpd-doc	lighttpd 1.4.19	Web Server
2	rails	Ruby on Rails 2.0.2	Language
3	postgresql postgresql-contrib postgresql-doc	PostgreSQL 8.3.1	database
4	postfix	Postfix 2.5.1	Email service
5	python-moinmoin	Moin Moin 1.5.8	Wiki Application
6	cupsys	CUPS 1.3.7	Print Server
7	cvs xinetd	CVS 1.12.13	Version Control System
8	backuppc	BackupPC 3.0.0	Backup
9	yum	Yum 2.4.0	Package Management

Table 4: Two package sets.

Name	Image size	Packages		
		Size	# Before	# After
Ubuntu, package set 1, asc, instance1	879	332	255	391
Ubuntu, package set 1, asc instance 2	883	336	255	391
Ubuntu, package set 1, desc	885	338	255	392
Ubuntu, package set 2, asc	873	326	255	397
Ubuntu, package set 2, desc	867	320	255	402
Ubuntu, remove no dependency	885	-	391	377
Ubuntu, remove all dependencies	881	-	391	256
CentOS, package set 1, asc	1435	387	359	443
Fedora, package set 1, asc	1307	405	194	306

Table 3: Package installation and removal sets. "asc" and "desc" refer to the order in which packages are installed. Image size is post-installation / removal. All sizes are in megabytes; the "before" and "after" columns reflect the number of installed packages before and after the installation / removal action.

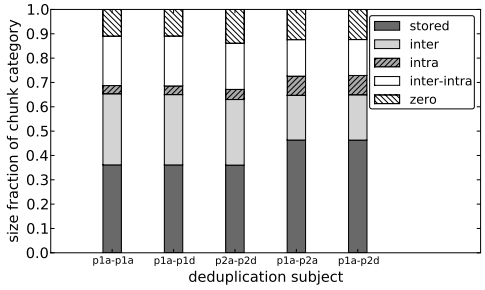


Figure 13: Differing package installation orders. *p1a* denotes package set 1, ascending order, and others follow the same naming rule. Variable-size chunking, chunk size 512 B. Different package installation orders generate nearly identical disk images.

on a "gold" image; it suffices to use deduplication to detect identical disk blocks in the images.

When we calculated the deduplication ratio for the second case, we found that the amount of newly introduced unique data is far less than the absolute size of the package sets. Although the two sets have no package in common, they are designed to be in same application areas, making corresponding packages likely to share dependencies. The dependent packages outnumber the listed packages, allowing the potential of high deduplication; moreover, the final package counts of the two package sets are approximately equal, implying that the dependency package numbers of them are approximately equal.

We also experimented with installing the same packages in the same order on different operating systems. The Debian distribution uses `deb` packages and manages them using `aptitude`. Red Hat distributions use the RPM package system, which uses `yum` to install and update packages. Some packages are system related, either with different binary or different library and dependencies. Figure 14 shows the result of installing package set 1 on Ubuntu, CentOS and Fe-

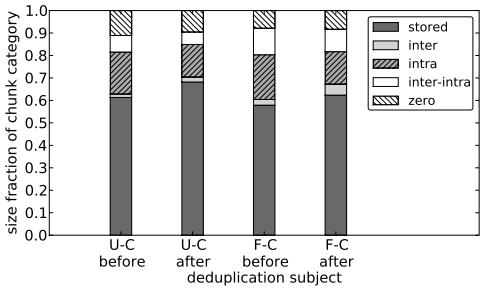


Figure 14: Package systems comparison on Ubuntu Server 8.04 (U), CentOS 5.2, no desktop (C) and Fedora 9, no desktop (F). Both install package set 1 in ascending order. Variable-size chunking, chunk size 512 B. Package system's contribution to deduplication is outweighed by OS heterogeneity.

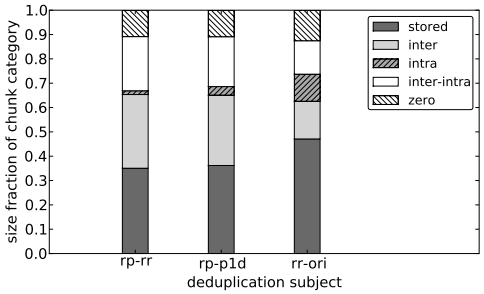


Figure 15: Package removal orders. Variable-size chunking, chunk size 512 B. *rp* denotes removing only target packages. *rr* denotes removing target packages and all dependencies (revert to original). *ori* denotes the original disk image before package installation. Data of the removed packages are not erased from disk image, which still resemble the image with the installed packages.

dora, revealing two points. First, by installing the same set of packages, the deduplication ratio for different package systems drops. This is not surprising—the binary data for exactly the same software packages in `deb` and `rpm` format are different. Second, the deduplication ratio for the *same* package system also drops. This indicates software packages are primarily OS dependent, not package system dependent. The conclusion is that packaging system has little effect on deduplication ratio, particularly when compared to the diversity of the underlying operating systems.

The final factor we examined was the effect of removing packages; we compared the resulting disk images with the original and dirty disk images from earlier experiments. "Removing" was done by executing the `apt-get purge` command; as a result, the package name is removed from the list generated by `dpkg --get-selections`. We can see from Figure 15 that, as expected, removed packages are not actually wiped off the disk; rather, the files are marked as deleted. This is evident from the high deduplication ratio between the installed and removed images and lower ratio for the other two pairs. We can also see that the removal order is relatively unimportant as well. In order to preserve a high deduplication ratio, there should be as little difference in package installation as possible because it is only package installation, not removal, that alters the disk image significantly. It is a good idea to leave all dependencies (orphan or not) installed rather than removed because the file system will allocate different regions for these dependencies when reinstalling them. Moreover, on a deduplicated system, unused packages require no additional space, so there is no reason to remove them.

Method	Size
tar+gzip, level 9	21152781050 (19.7 GB)
7-zip, level 9	15008385189 (14.0 GB)
var, chunk size 512 B, level 9	29664832343 (27.6 GB)
var, chunk size 4 KB, level 9	23994815029 (22.3 GB)

Table 5: Compression parameters.

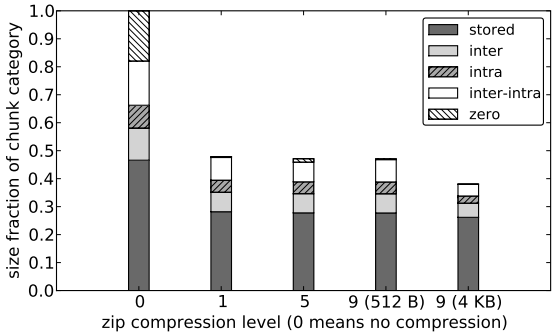


Figure 16: Chunk-wise compression. Variable-size chunking, chunk size 512 B except the rightmost bar for 4 KB. Chunk-wise compression is effective but limited by small window size.

4.4 Chunk Compression

In this section, we discuss the impact of compressing chunks on space reduction. We employed the `zlib` compression method, using the DEFLATE algorithm [10]), which is essentially the same as that used in Linux's `zip` and `gzip` code. As a good comparison group, we used `7-zip` (LZMA algorithm) to archive and compress the original disk images, because various benchmarks [7, 15] have shown that `7-zip` is capable of higher compression than `zip`. We used the *Linux* chunk store from Section 4.1 for our compression experiments. The final sizes of the chunk store are shown in Table 5.

Figure 16 tells us three things. First, chunk-wise compression can reduce the size of the stored chunks by 40%. Most of the saving is contributed by chunks that occur exactly once; shared chunks are unlikely to be further compressed. Since we only store a single instance of zero chunks, the saving from such chunks is no more than 1 KB. Second, for 512 B chunks, increasing compression level yields negligible benefit because the 512 B chunks do not fully utilize `zip`'s 32 KB sliding window. Third, while larger chunk size yields better compression, this effect does not counteract the lower deduplication ratio generated by using larger chunks, and may deteriorates real-time performance as spending more time on decompression.

5. FUTURE WORK

Our experiments have shown which factors influence the *amount* of deduplication available when deduplicating VM disk images, but they do not address the issue of locality in deduplication. We believe that deduplication in VM disk images will exhibit both temporal and spatial locality because similar files in different disk images will contain many similar chunks in the same order. We are currently working to evaluate the amount of locality available in disk images. If there is significant locality, we can improve deduplication performance by co-locating nearby chunks from one disk image, since those chunks will likely be near one another in other disk images as well.

Another important factor for deduplication is user related data. While all of the VMs in our experiments were not used by real users, involving user data in VMs may impact deduplication ratios [18, 19]. Different results may be discovered when identical, homogeneous or heterogeneous VMs are used and deduplicated.

Issues such as privacy and security can be important for virtual machines, particularly when a hosting center contains thousands of disk images. Thus, we are investigating the use of secure deduplication [22] to combine encryption with deduplication for virtual machines. Doing so promises to provide not only privacy, but also better security since it will be more difficult for an intruder from system A to compromise encrypted chunks from system B.

We have not integrated deduplication into a VMM; however we plan to modify QEMU [5] or Xen [4] to support such functionality. In addition to performance measurements, we hope to test whether flat disk images and sparse disk images perform similarly under deduplication; if, as we expect, they do exhibit similar performance, it will make the use of flat disk images more appealing because deduplication eliminates the need to optimize for unused, zero-filled sectors. Performance measurements similar to those on a more efficient file system than the Venti system used by Liguori and Van Hensbergen [16] will show whether deduplication can be effectively integrated into a virtual machine monitor.

6. CONCLUSIONS

Deduplication is an efficient approach to reduce storage demands in environments with large numbers of VM disk images. As we have shown, deduplication of VM disk images can save 80% or more of the space required to store the operating system and application environment; it is particularly effective when disk images correspond to different versions of a single operating system "lineage," such as Ubuntu or Fedora.

We explored the impact of many factors on the effectiveness of deduplication. We showed that package installation and language localization have little impact on deduplication ratio. However, factors such as the base operating system (BSD versus. Linux) or even the Linux distribution can have a major impact on deduplication effectiveness. Thus, we recommend that hosting centers suggest "preferred" operating system distributions for their users to ensure maximal space savings. If this preference is followed subsequent user activity will have little impact on deduplication effectiveness.

We found that, in general, 40% is approximately the highest deduplication ratio if no obviously similar VMs are involved. However, while smaller chunk sizes provide better deduplication, the relative importance of different categories of sharing is largely unaffected by chunk size. As expected, chunk-wise compression dramatically reduces chunk store size, but compression level has little effect on the amount space saved by chunk compression.

We also noted that fixed-size chunking works very well for VM disk images, outperforming variable-sized chunking in some cases, thus confirming earlier findings [19] stated. In particular, in small chunk stores such as those in the Ubuntu Server series experiment in Section 4.1, fixed-size chunking results in better deduplication than variable-size chunking This is good news for implementers of deduplication systems, since fixed-size chunking is typically easier to implement, and performs considerably better.

Surprisingly, the deduplication ratio of different releases within a given lineage does not depend heavily on whether the releases are consecutive. We expected to find that, the further away two releases are, the less effective deduplication becomes. We suspect that this effect is not seen because the same parts of the operating system are changed at each release, while large portions of the operating system remain unchanged for many releases. We also noted that, while different releases of a given lineage are similar, large changes are made in operating systems when they are "forked off" from existing distributions.

Throughout all of our experiments, we found that the exact effectiveness of deduplication is data-dependent—hardly surprising, given the techniques that deduplication uses to reduce the amount of storage consumed by the VM disk images. However, our conclusions are based on real-world disk images, not images created for deduplication testing; thus, we believe that these findings will generalize well to a wide array of VM disk images. We believe that the results of our experiments will help guide system implementers and VM users in their quest to significantly reduce the storage requirements for virtual machine disk images for hosting centers with large numbers of individually managed virtual machines.

Acknowledgments

We would like to thank our colleagues in the Storage Systems Research Center for their input and guidance. This work was supported in part by funding from Symantec and the UC MICRO program. Ethan Miller was also supported by the Department of Energy's Petascale Data Storage Institute under award DE-FC02-06ER25768. We are also grateful for the financial support of the industrial partners of the SSRC.

7. REFERENCES

[1] Annapureddy, S., Freedman, M. J., and Mazières, D. Shark: Scaling file servers via cooperative caching. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI)* (2005), pp. 129–142.

[2] Anonymous. Secure hash standard. FIPS 180-1, National Institute of Standards and Technology, Apr. 1995.

[3] `http://bagside.com/bagvapp/`.

[4] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the art of virtualization.

In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)* (2003).

[5] Bellard, F. Qemu, a fast and portable dynamic translator. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2005), USENIX Association, pp. 41–41.

[6] Bhagwat, D., Eshghi, K., and Mehra, P. Content-based document routing and index partitioning for scalable similarity-based searches in a large corpus. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '07)* (Aug. 2007), pp. 105–112.

[7] Collin, L. A quick benchmark: Gzip vs. Bzip2 vs. LZMA, 2005.

[8] Cox, L. P., Murray, C. D., and Noble, B. D. Pastiche: Making backup cheap and easy. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)* (Boston, MA, Dec. 2002), pp. 285–298.

[9] Daum, M., and Lucks, S. Hash Collisions (The Poisoned Message Attack) "The Story of Alice and her Boss". *Presentation at Rump Sessions of Eurocrypt 2005 5* (2005).

[10] Deutsch, P. Deflate compressed data format specification version 1.3.

[11] Douceur, J. R., Adya, A., Bolosky, W. J., Simon, D., and Theimer, M. Reclaiming space from duplicate files in a serverless distributed file system. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS '02)* (Vienna, Austria, July 2002), pp. 617–624.

[12] Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A. C., Varghese, G., Voelker, G. M., and Vahdat, A. Difference Engine: Harnessing memory redundancy in virtual machines. In *Proceedings of the 8th Symposium on Operating Systems Design and Implementation (OSDI)* (Dec. 2008), pp. 309–322.

[13] Henson, V. An analysis of compare-by-hash. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS-IX)* (May 2003).

[14] Hollingsworth, J., and Miller, E. Using content-derived names for configuration management. In *Proceedings of the 1997 Symposium on Software Reusability (SSR '97)* (Boston, MA, May 1997), IEEE, pp. 104–109.

[15] Kingsley G. Morse, J. Compression tools compared. *Linux J. 2005*, 137 (2005), 3.

[16] Liguori, A., and Van Hensbergen, E. Experiences with content addressable storage and virtual disks. In *Proceedings of the First Workshop on I/O Virtualization* (Dec. 2008).

[17] Muthitacharoen, A., Chen, B., and Mazières, D. A low-bandwidth network file system. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)* (Oct. 2001), pp. 174–187.

[18] Nath, P., Kozuch, M. A., O'Hallaron, D. R., Harkes, J., Satyanarayanan, M., Tolia, N., and Toups, M. Design tradeoffs in applying content addressable storage to enterprise-scale systems based on virtual machines. In *Proceedings of the 2006 USENIX Annual Technical Conference* (2006).

[19] Policroniades, C., and Pratt, I. Alternatives for detecting redundancy in storage systems data. In *Proceedings of the 2004 USENIX Annual Technical Conference* (Boston, MA, June 2004), USENIX, pp. 73–86.

[20] Quinlan, S., and Dorward, S. Venti: A new approach to archival storage. In *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)* (Monterey, California, USA, 2002), USENIX, pp. 89–101.

[21] Rabin, M. O. Fingerprinting by random polynomials. Tech. Rep. TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[22] Storer, M. W., Greenan, K. M., Long, D. D. E., and Miller, E. L. Secure data deduplication. In *Proceedings of the 2008 ACM Workshop on Storage Security and Survivability* (Oct. 2008).

[23] `http://www.thoughtpolice.co.uk/vmware/`.

[24] `http://www.vmware.com/appliances/`.

[25] VMware Inc. Virtual disk format. VMware web site, http://www.vmware.com/interfaces/vmdk.html, 11 2007.

[26] Wang, X., Yin, Y. L., and Yu, H. Finding collisions in the full SHA-1. *Lecture Notes in Computer Science 3621* (2005), 17–36.

[27] You, L. L. *Efficient Archival Data Storage*. PhD thesis, University of California, Santa Cruz, June 2006. Available as Techncial Report UCSC-SSRC-06-04.

[28] You, L. L., Pollack, K. T., and Long, D. D. E. Deep Store: An archival storage system architecture. In *Proceedings of the 21st International Conference on Data Engineering (ICDE '05)* (Tokyo, Japan, Apr. 2005).

[29] Zhu, B., Li, K., and Patterson, H. Avoiding the disk bottleneck in the Data Domain deduplication file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST)* (Feb. 2008).

[30] Ziv, J., and Lempel, A. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory 23*, 3 (May 1977), 337–343.