Proceedings of the 2006 USENIX Annual Technical Conference (USENIX '06), Boston, Massachusetts, May-June 2006.

Design Tradeoffs in Applying Content Addressable Storage to Enterprise-scale Systems Based on Virtual Machines

Partho Nath[†], Michael A. Kozuch[⋆], David R. O'Hallaron[‡], Jan Harkes[‡],
M. Satyanarayanan[‡], Niraj Tolia[‡], and Matt Toups[‡]
[†]*Penn State University,* [⋆]*Intel Research Pittsburgh, and* [‡]*Carnegie Mellon University*

Abstract

This paper analyzes the usage data from a live deployment of an enterprise client management system based on virtual machine (VM) technology. Over a period of seven months, twenty-three volunteers used VM-based computing environments hosted by the system and created over 800 checkpoints of VM state, where each checkpoint included the virtual memory and disk states. Using this data, we study the design tradeoffs in applying content addressable storage (CAS) to such VM-based systems. In particular, we explore the impact on storage requirements and network load of different privacy properties and data granularities in the design of the underlying CAS system. The study clearly demonstrates that relaxing privacy can reduce the resource requirements of the system, and identifies designs that provide reasonable compromises between privacy and resource demands.

1 Introduction

The systems literature of recent years bears witness to a significantly increased interest in virtual machine (VM) technology. Two aspects of this technology, namely platform independence and natural state encapsulation, have enabled the application of this technology in systems designed to improve scalability [6, 14, 16, 32, 40, 49], security [15, 21, 47], reliability [1, 4, 8, 25, 44], and client management [7, 5, 20].

The benefits derived from platform independence and state encapsulation, however, often come with an associated cost, namely the management of significant data volume. For example, enterprise client management systems [7, 20] may require the storage of tens of gigabytes of data *per user*. For each user, these systems store an image of the user's entire VM state, which includes not only the state of the virtual processor and platform devices, but the memory and disk states as well.

While this cost is initially daunting, we would expect a collection of VM state images to have significant data redundancy because many of the users will employ the same operating systems and applications. Content addressable storage (CAS) [3, 27, 30, 36, 44, 48] is an emerging mechanism that can reduce the costs associated with this volume of data by eliminating such redundancy. Essentially, CAS uses cryptographic hashing techniques to identify data by its *content* rather than by name. Consequently, a CAS-based system will identify sets of identical objects and only store or transmit a single copy even if higher-level logic maintains multiple copies with different names.

To date, however, the benefit of CAS in the context of enterprise-scale systems based on VMs has not been quantified. In this paper, we analyze data obtained from a seven-month, multi-user pilot deployment of a VM-based enterprise client management system called Internet Suspend/Resume (ISR) [19, 37]. Our analysis aims to answer two basic questions:

Q1: By how much can the application of CAS reduce the system's storage requirements?

Q2: By how much can the application of CAS reduce the system's network traffic?

The performance of CAS depends upon several system parameters. The answers to Q1 and Q2, therefore, are analyzed in the context of the two most important of these design criteria:

C1: The *privacy policy*, and

C2: the *object granularity*.

The storage efficiency of a CAS system, or the extent to which redundant data is eliminated, depends upon the degree to which that system is able to *identify* redundant data. Hence, the highest storage efficiency requires users to expose cryptographic digests to the system and potentially to other users. As we shall see, the effects of this exposure can be reduced but not eliminated. Consequently, criterion C1 represents a trade-off between storage efficiency and privacy.

Object granularity, in contrast, is a parameter that dictates how finely the managed data is subdivided. Because CAS systems exploit redundancy at the object level, large objects (like disk images) are often represented as a sequence of smaller objects. For example, a multi-gigabyte disk image may be represented as a sequence of 128 KB objects (or *chunks*). A finer granularity (smaller chunksize) will often expose more redundancy than a coarser granularity. However, finer granularities will also require more meta-data to track the correspondingly larger number of objects. Hence, criterion C2 represents the trade-off between efficiency and meta-data overhead.

The results obtained from the ISR pilot deployment indicate that the application of CAS to VM-based management systems is more effective in reducing storage and network resource demands than applying traditional compression technology such as the Lempel-Ziv compression [50] used in *gzip*. This result is especially significant given the non-zero runtime costs of compressing and uncompressing data. In addition, combining CAS and traditional compression reduces the storage and network resource demands by a factor of two beyond the reductions obtained by using traditional compression technology alone.

Further, using this real-world data, we are able to determine that enforcing a strict privacy policy requires approximately 1.5 times the storage resources required by a system with a less strict privacy policy. Finally, we have determined that the efficiency improvements derived from finer object granularity typically outweighs the meta-data overhead. Consequently, the disk image chunksize should be between 4 and 16 KB.

Sections 4 and 5 will elaborate on these results from the pilot deployment. But first, we provide some background on ISR, content addressable storage, and the methodology used in the study.

2 Background

2.1 Internet Suspend/Resume

Internet Suspend/Resume (ISR) is an enterprise client management system that allows users to access their personal computing environments from different physical machines. The system is based on a combination of VM technology and distributed storage. User computing environments are encapsulated by VM instances, and the state of such a VM instance, when idle, is captured by system software and stored on a carefully-managed server. There are a couple of motivations for this idea. First, decoupling the computing environment from the hardware allows clients to migrate across different hosts. Second, storing VM state on a remote storage repository simplifies the management of large client installations. The physical laptops and desktops in the installation no longer contain any hard user-specific state, and thus client host backups are no longer necessary; the only system that needs to be backed up is the storage repository.

Figure 1 shows the setup of a typical ISR system. The captured states of user environments are known as known as *parcels* and are stored on a collection of (possibly) distributed *content servers*. For example, in the figure, Bob owns two parcels. One environment includes Linux as the operating system, and the other includes Windows XP.

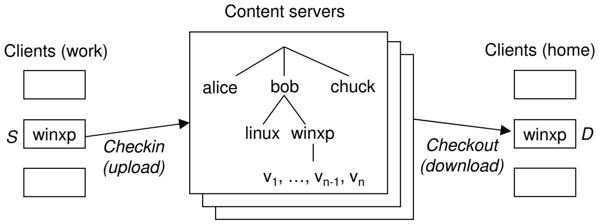


Figure 1: An ISR system.

Each parcel captures the complete state of some VM instance. The two most significant pieces of state are the *memory image* and the *disk image*. In the current ISR deployment, memory images are 256 MB and disk images are 8 GB. Each memory image is represented as a single file. Each disk image is partitioned into a set of 128 KB *chunks* and stored on disk, one file per chunk.

For each parcel, the system maintains a sequence of checkpointed diff-based *versions*, $v_1, \ldots, v_{n-1}, v_n$. Version v_n is a complete copy of the memory and disk image. Each version v_k, $1 \leq v_k \leq v_{n-1}$, has a complete copy of the memory image, along with the chunks from the v_k version of the disk image that changed between version v_k and v_{k+1}.

Each client host in the ISR system runs a *VM monitor* that can load and execute any parcel. ISR provides a mechanism for suspending and transferring the execution of these parcels from one client host to another. For example, Figure 1 shows a scenario where a user transfers the execution of a VM instance from a source host S at the office to a destination host D at home.

The transfer occurs in two phases: a *checkin* step followed by a *checkout* step. After the user suspends execution of the VM monitor on S, the checkin step uploads the memory image and any dirty disk chunks from S to one of the content servers, creating a new parcel version on the server. The checkout step downloads the memory image of the most recent parcel version from the content server to *D*. The user is then able to resume execution of the parcel on *D* (even before the entire disk image is present). During execution, ISR fetches any missing disk chunks from the content server *on demand* and caches those chunks at the client for possible later use.

2.2 Content Addressable Storage

Content addressable storage (CAS) is a data management approach that shows promise for impoving the efficiency of ISR systems. CAS uses cryptographic hashing to reduce storage requirements by exploiting commonality across multiple data objects [13, 23, 29, 42, 43, 48]. For example, to apply CAS to an ISR system, we would represent each memory and disk image as a sequence of fixed-sized chunk files, where the filename of each chunk is computed using a collision-resistant cryptographic hash function. Since chunks with identical names are assumed to have identical contents, a single chunk on disk can be included in the representations of multiple memory and disk images. The simplest example of this phenomenon is that many memory and disk images contain long strings of zeros, most of which can be represented by a single disk chunk consisting of all zeros. A major goal of this paper is to determine to what extent such redundancy exists in realistic VM instances.

3 Methodology

Sections 4 and 5 present our analysis of CAS technology in the context of ISR based on data collected during the first 7 months of a pilot ISR deployment at Carnegie Mellon University. This section describes the deployment, and how the data was collected and analyzed.

3.1 Pilot Deployment

The pilot deployment (pilot) began in January, 2005, starting with about 5 users and eventually growing to 23 active users. Figure 2 gives the highlights. Users were recruited from the ranks of Carnegie Mellon students and staff and given a choice of a Windows XP parcel, a Linux parcel, or both. Each parcel was configured with an 8 GB virtual disk and 256 MB of memory. The *gold images* used to create new parcels for users were updated at various times over the course of the pilot with security patches.

Number of users	23
Number of parcels	36
User environment	Windows XP or Linux
Memory image size	256 MB
Disk image size	8 GB
Client software	ISR+Linux+VMware
Content server	IBM BladeCenter
Checkins captured	817
Uncompressed size	6.5 TB
Compressed size	0.5 TB

Figure 2: Summary of ISR pilot deployment.

The content server is an IBM BladeCenter with 9 servers and a 1.5 TB disk array for storing user parcels. Users downloaded and ran their parcels on Linux-based clients running VMware Workstation 4.5.

3.2 Data Collection

During the course of the pilot, users performed numerous checkin operations, eventually creating 817 distinct parcel versions on the content server. In August, 2005, after 7 months of continuous deployment, a snapshot of the memory and disk images of these parcel versions was taken on the content server. In uncompressed form, the snapshot state would have consumed about 6.5 TB. However, due to ISR's diff-based representation and gzip compression, it only required about 0.5 TB of disk space. This snapshot state was copied to another server, where it was post-processed and stored in a database for later analysis.

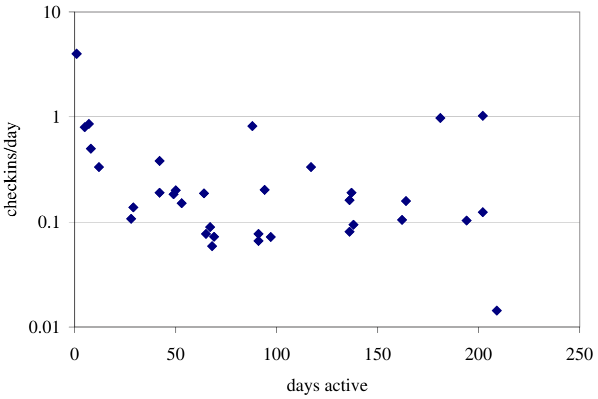


Figure 3: Observed parcel checkin frequency

Figure 3 summarizes parcel usage statistics for the deployment data. Each point in the figure represents a single parcel and indicates the number of days that parcel was active as well as its checkin frequency (average number of checkins per day). Parcels could be active for less than the entire duration of the deployment either because the parcel was created after the initial deployment launch or because a user left the study early (e.g. due to student graduation or end-of-semester constraints). Since new users were added throughout the course of the pilot, during post-processing we normalized the start time of each user to day zero. No extrapolation of data was performed, thus the usage data for a user who has used the system for n days appears in the first n days worth of data in the corresponding analysis. We also removed several parcels that were used by developers for testing, and thus were not representative of typical use.

3.3 Analysis

The August 2005 snapshot provided a complete history of the memory and disk images produced by users over time. This history allowed us to ask a number of interesting "what if" questions about the impact of different design choices, or policies, on the performance of the ISR system. In particular, we explored three different storage policies: a baseline non-CAS *Delta* policy and two different CAS policies called *IP* and *ALL*. These are summarized in Figure 4. In each approach, a parcel's memory and disk images are partitioned into fixed-sized chunks, which are then encrypted, and optionally compressed using conventional tools like gzip.

Policy	Encryption	Meta-data
Delta	private per-parcel key	none
IP	private per-parcel key	(tag) array
ALL	convergent encryption	(tag, key) array

Figure 4: Storage policy encryption technique summary.

As will be shown in sections 4 and 5, differences in the storage and encryption of data chunks affect not only the privacy afforded to users but also dramatically alter the resources required for storage and network transmission. For our evaluations, we chose chunksizes of 4KB (a typical disk-allocation unit for most operating systems) and larger.

Delta policy. In this non-CAS approach, the most recent disk image v_n contains a complete set of chunks. For each version $k < n$, disk image v_k contains only those chunks that differ in disk image v_{k+1}. Thus, we say that Delta exploits *temporal redundancy* across the versions.

Chunks in all of the versions in a parcel are encrypted using the same per-parcel private key. Individual chunks are addressed by their position in the image (logical block addressing), hence no additional meta-data is needed. Memory images are represented in the same way. Delta is similar to the approach used by the current ISR prototype (the current prototype only chunks the disk image and not the memory image). We chose it as the baseline because it is an effective state-of-the-art non-CAS approach for representing versions of VM images.

IP (intra-parcel) policy. In this CAS approach, each parcel is represented by a separate pool of unique chunks shared by all versions, $v_1, \ldots, v_n$, of that parcel. Similar to Delta, IP identifies temporal redundancy between contiguous parcel versions. However, IP can also identify temporal redundancy in non-contiguous versions (e.g., disk chunk i is identical in versions 4 and 6, but different in version 5), and it can also identify any *spatial redundancy* within each version.

As with Delta, each chunk is encrypted using a single per-parcel private key. However, each version of each disk image (and each memory image) requires additional meta-data to record the sequence of chunks that comprise the image. In particular, the meta-data for each image is an array of *tags*, where tag i is the SHA-1 hash of chunk i. This array of tags is called a *keyring*.

ALL policy. In this CAS approach, all parcels for all users are represented by a single pool of unique chunks. Each chunk is encrypted using *convergent encryption* [11], where the encryption key is simply the SHA-1 hash of the chunk's original plaintext contents. This allows chunks to be shared across different parcels and users, since if the original plaintext chunks are identical, then the encrypted chunks will also be identical.

As with IP, each version of each disk image (and each memory image) requires additional keyring meta-data to record the sequence of chunks that compose the image, in this case an array of (tag, key) tuples, where key i is the encryption key for chunk i, and tag i is the SHA-1 hash of the encrypted chunk. Each keyring is then encrypted with a per-parcel private key.

The IP and ALL policies provide an interesting tradeoff between privacy and space efficiency. Intuitively, we would expect the ALL policy to be the most space-efficient because it identifies redundancy across the maximum number of chunks. However, this benefit comes at the cost of decreased privacy, both for individual users and the owners/operators of the storage repository. The reason is that ALL requires a consistent encryption scheme such as convergent encryption for all blocks. Thus, individual users are vulnerable to dictionary-based traffic analysis of their requests, either by outside attackers or the administrators of the systems. Owner/operators are vulnerable to similar analysis, if, say, the contents of their repository are subpoenaed by some outside agency.

Choosing appropriate chunk sizes is another interesting policy decision. For a fixed amount of data, there is a tension between chunk size and the amount of storage required. Intuitively, we would expect that smaller chunk sizes would result in more redundancy across chunks, and thus use less space. However, as the chunk size decreases, there are more chunks, and thus there is more keyring meta-data. Other chunking techniques such as Rabin Fingerprinting [26, 31, 38] generate chunks of varying sizes in an attempt to discover redundant data that does not conform to a fixed chunk size. The evaluation of non-fixed-size chunk schemes is beyond the scope of this paper but is on our agenda for future work.

The remainder of the paper uses the data from the ISR deployment to quantify the impact of CAS privacy and chunksize policies on the amount of storage required for the content servers, and the volume of data that must be transferred between clients and content servers.

4 Results: CAS & Storage

Because server storage represents a significant cost in VM-based client management systems, we begin our discussion by investigating the extent to which a CAS-based storage system could reduce the volume of data managed by the server.

4.1 Effect of Privacy Policy on Storage

As expected, storage policy plays a significant role in the efficiency of the data management system. Figure 5 presents the growth in storage requirements over the lifetime of the study for the three different policies using a fixed chunksize (128 KB). As mentioned in Section 3.2, the graph normalizes the starting date of all users to day zero. The growth in the storage from thereon is due to normal usage of disks and storage of memory checkpoints belonging to the users. The storage requirement shown includes both the disk and memory images.

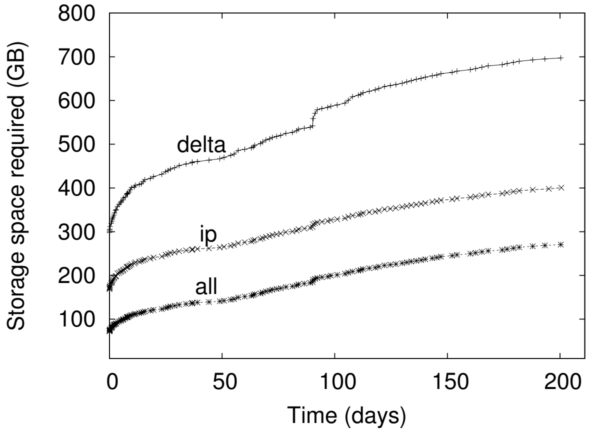


Figure 5: Growth of storage needs for Delta, IP, and ALL.

CAS provides significant savings. As shown in Figure 5, adopting CAS with the IP policy reduces the required server resources at day 201 under the Delta policy by 306 GB, from 717 GB to 411 GB. This reduction represents a savings of 42%.

Recall that adopting CAS is a lossless operation; CAS simply stores the same data more efficiently than the Delta policy. The improved efficiency is due to the fact that the Delta policy only exploits temporal redundancy between versions. That is, the Delta policy only identifies identical objects when they occur in the same location in subsequent versions of a VM image. The IP policy, in contrast, identifies redundancy anywhere within the parcel – within a version as well as between versions (including between non-subsequent versions).

Note that the 42% space savings was realized without compromising privacy. Users in a CAS-IP-backed system do not expose the contents of their data to any greater degree than users of a Delta-backed system.

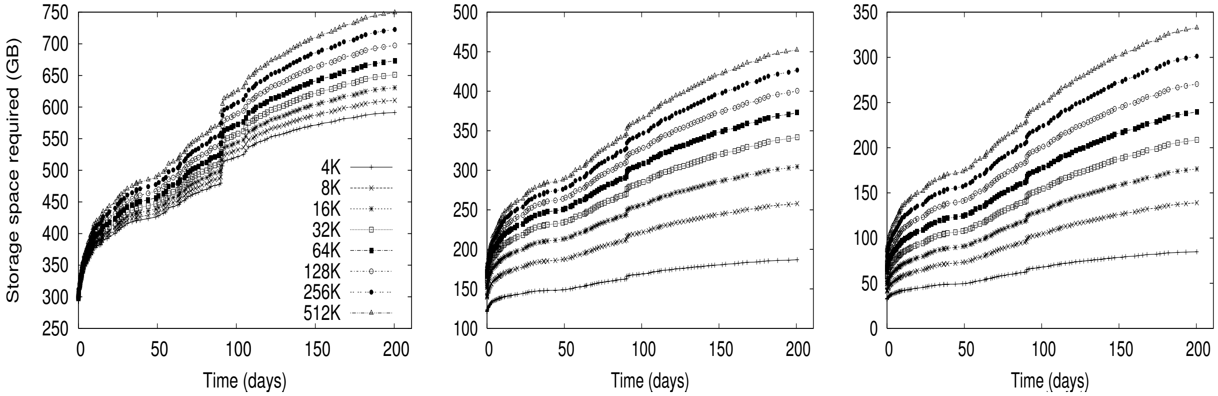
Relaxing privacy introduces additional gains. In systems where a small relaxation of privacy guarantees is acceptable, additional savings are possible. When the privacy policy is relaxed from IP to ALL, the system is able to identify additional redundancy that may exist between different users' data. From Figure 5, we see that such a relaxation will reduce the storage resources required by another 133 GB, to 278 GB. The total space savings realized by altering the policy from Delta to ALL is 61%.

On comparing ALL with IP in Figure 5, we see that the curves are approximately parallel to each other. However, under certain situations, a system employing the ALL policy could dramatically outperform a similar system that employs the IP policy. Imagine for example a scenario where a security patch is applied by each of a large number, *N*, of users in an enterprise. Assuming that the patch affected each user's environment in the same way, by introducing *X* MB of new data, an IP server would register a total addition of *NX* MB. In contrast, an ALL server would identify the *N* copies of the patched data as identical and would consequently register a total addition of *X* MB.

The starting points of the curves in Figure 5 are also of interest. Because the X-axis has been normalized, this point corresponds to the creation date of all parcels. To create a new parcel account, the system administrator copies a gold image as version 1 of the parcel. Hence, we would assume that the system would exhibit very predictable behavior at time zero.

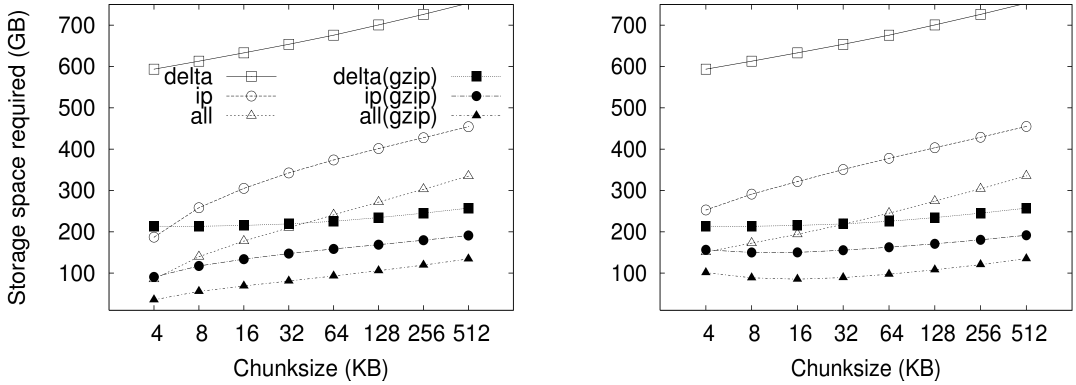
For example, under the Delta policy which only reduces redundancy *between versions*, the system data should occupy storage equal to the number of users times the space allocated to each user. In the deployment, users were allocated 8 GB for disk space and 256 MB for memory images. Thirty-six parcels should then require approximately 300 GB of storage space which is exactly the figure reported in the figure.

For the IP policy, one would also expect the server to support a separate image for each user. However, CAS had eliminated the redundant data within each of these images yielding an average image size of approximately 4 GB. The observed 171 GB storage space is consistent with this expectation.



(a) Delta Policy (b) IP Policy (c) ALL Policy

Figure 6: Storage space growth for various chunksizes without meta-data overhead (y-axis scale varies).



(a) Without meta-data overhead (b) With meta-data overhead

Figure 7: Server space required, after 201 deployment days.

Under the ALL policy in contrast, one would expect the system to store a single copy of the gold image shared by all users, yielding a total storage requirement of 8 GB plus 256 MB (closer to 4 GB, actually, due to the intra-image redundancy elimination). We were quite surprised, consequently, to observe the 72 GB value reported in the figure. After reviewing the deployment logs, we determined that this value is due to the introduction of multiple gold images into the system. To satisfy different users, the system administrators supported images of several different Linux releases as well as several instances of Windows images. In all, the administrators had introduced 13 different gold images, a number that is consistent with the observed 72 GB of occupied space.

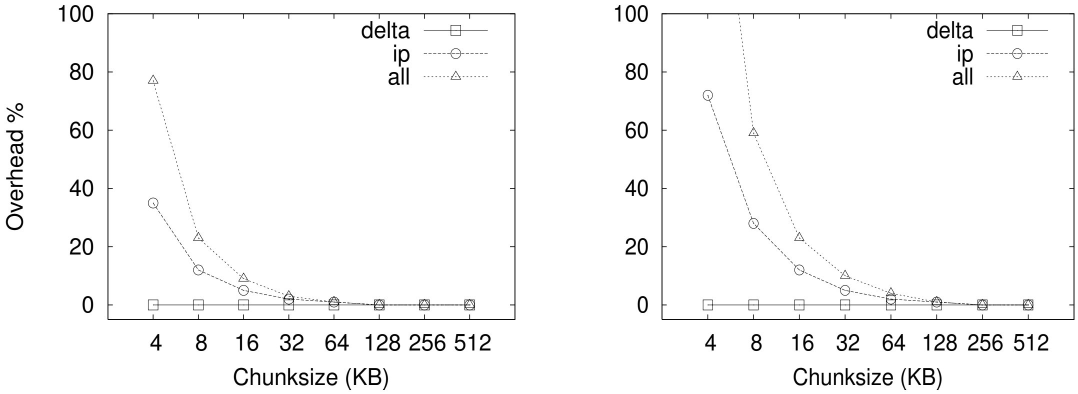
Another point of interest is a disturbance in the curve that occurs at the period around 100 days. We note that the disturbance is significant in the Delta curve, smaller in the IP curve, and almost negligible in the ALL curve. We've isolated the disturbance to a single user and observe that this anomaly is due to the user reorganizing his disk image without creating new data that did not already exist somewhere in the system. Hence, we conclude that this must have been an activity similar to defragmentation or re-installation of an operating system.

4.2 Effect of Chunksize on Storage

In addition to privacy considerations, the administrator of a VM-based client management system may choose to optimize the system efficiency by tuning the chunksize. The impact of this parameter on storage space requirements is depicted in Figure 6; in this figure, we present what the growth curves of Figure 5 would have been had we chosen different chunksizes.

Note that the effect of this parameter is not straightforward. Varying the chunksize has three different effects on efficiency.

First, smaller chunksizes tend to expose more redundancy in the system. As a trivial exercise, consider two objects each of which, in turn, comprises two blocks ($Object_1 = AB$ and $Object_2 = CA$). If the chunksize is chosen to be a whole object, the content addresses of



(a) Uncompressed user data

(b) Compressed user data

Figure 8: Meta-data overhead expressed as a percentage of user data.

$Object_1$ and $Object_2$ will differ and no redundancy will be exposed. If the chunksize is chosen to be a block, in contrast, the identical *A* blocks will be identified and a space savings of 25% will result.

Second, smaller chunksizes require the maintenance of more meta-data. With the whole-object chunksize from the example above, the system would maintain two content addresses, for $Object_1$ and $Object_2$. With the block chunksize, however, the system must maintain two sets of two content addresses so that $Object_1$ and $Object_2$ may each be properly reconstructed. Note further that this additional meta-data maintenance is required whether or not any redundancy was actually identified in the system.

Third, smaller chunksizes tend to provide a reduced opportunity for post-chunking compression. In addition to chunk-level redundancy elimination through CAS, intra-chunk redundancy may be reduced through traditional compression techniques (such as *gzip*). However, as the chunksize is reduced, these techniques have access to a smaller intra-chunk data pool on which to operate, limiting their efficiency.

To better understand the effect of chunksize, we analyzed the deployment data for all three storage policies with and without compression under several different chunksizes. The results are shown in Figure 7.

All three effects of chunksize can be observed in this figure. For example, Figure 7(a), which ignores the increased meta-data required for smaller chunksizes, clearly indicates that smaller chunksizes expose more redundancy. These gains for small chunk sizes, however, are erased when the meta-data cost is introduced to the storage requirements in Figure 7(b). Finally, the reduced opportunities for compression due to smaller chunksize can be observed in Figure 7(b) by comparing the IP and IP(gzip) or ALL and ALL(gzip) curves.

CAS is more important than compression. In Figure 7(a), the Delta curve *with compression* intersects the IP and ALL curves *without compression*. The same is true in Figure 7(b) with respect to the ALL curve. This indicates, that given appropriate chunksizes, a CAS-based policy can outperform compression applied to a non-CAS-based policy.

Considering meta-data overheads, the ALL policy outperforms Delta with compression for all the chunksizes less than 64KB. This is a very remarkable result. Compression in the storage layer may be a high latency operation, and it may considerably affect virtual disk operation latencies. By use of CAS, one can achieve savings that exceed traditional compression techniques! If additional space savings are required, compression can be applied after the application of content addressing.

Figure 7(a) shows that compression provides an additional savings of a factor of two to three. For example, the space demands for the ALL policy, drops from 87GB to 36GB when using 4KB chunks, and from 342GB to 137GB when using 512KB chunks.

Exposing redundancy outweighs meta-data overhead. Figure 8 shows the ratio of meta-data (keyring size) to the size of the data. We observe that this ratio is as high as 80% for ALL, and 35% for IP at 4KB chunksize without compression and even higher after compression is applied to the basic data. Yet, from Figure 7(b), we observe from the IP and ALL curves that reducing chunksize always yields a reduction in storage requirements. This indicates that the gains through CAS-based redundancy elimination far exceed the additional meta-data overhead incurred from smaller chunksize.

The picture changes slightly with the introduction of traditional compression. The IP(gzip) and ALL(gzip) curves of Figure 7(b) indicate that the smallest chunksize is not optimal. In fact, we see from Figure 8 that the meta-data volume becomes comparable to the data volume at small chunksizes.

Small chunk sizes improve efficiency. With Figure 7(b), we are in a position to recommend optimal chunk sizes. Without compression, the optimal chunksize is 4 KB for the Delta, IP and ALL policies. With compression, the optimal chunksize is 8 KB for the Delta(gzip) policy and 16 KB for the IP(gzip) and ALL(gzip) policies.

5 Results: CAS & Networking

In a VM-based client management system, the required storage resources, as discussed in the previous section, represent a cost to the system administrator in terms of physical devices, space, cooling, and management. However, certain user operations, such as check-in and checkout, require the transmission of data over the network. While the system administrator must provision the networking infrastructure to handle these transmissions, perhaps the more significant cost is the user time spent waiting for the transmissions to complete.

For example, a common telecommuting scenario may be that a user works at the office for some time, checks-in their new VM state, travels home, and attempts to checkout their VM state to continue working. In the absence of CAS or traditional compression, downloading just the 256 MB memory, which is required before work can resume, over a 1 Mbps DSL line requires more than 30 minutes of wait time. After working at home for some time, the user will also want to checkin their new changes. Because the checkin image is typically larger than the checkout image, and because the upload speed of ADSL is often much slower than the download speed, the checkin operation can often require two hours or more.

Consequently, we devote this section to characterizing the benefits that CAS provides in terms of reducing the volume of data to be transmitted during typical upload (checkin) or download (checkout) operations.

5.1 Effect of Privacy Policy on Networking

As with storage, we begin the discussion by considering the effect of privacy policy on networking. We note that our definition of privacy policy affects the representation of data chunks in storage, not the mechanics of chunk transmission. However, the chosen storage policy can affect the capability of the system to identify redundant data blocks that need not be sent because they already exist at the destination.

As an example, suppose that a user copies a file within their virtual environment. This operation may result in a virtual disk that contains duplicate chunks. Under the IP and ALL policies, at the time of upload, the client will send a digest of modified chunks to the server, and the server may respond that the duplicate chunks need not be sent because the chunks (identified by the chunks' tags) already exist on the server. Such redundant data can occur for a variety of reasons (particularly under the ALL policy) including the push of software patches, user download of popular Internet content, and the installation and compilation of common software packages.

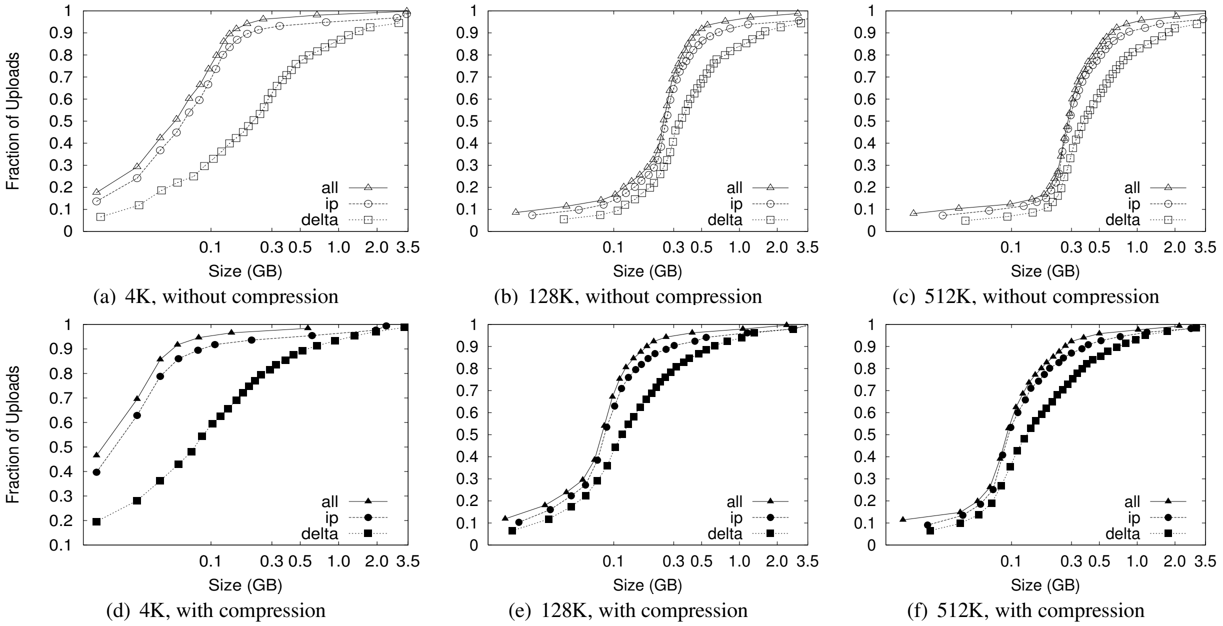
During download (checkout) operations, the client code will search through the existing version(s) of the user's data on that client to identify chunks that need not be retrieved from the server. As the system is only comparing the latest version on the server with the existing version on the client, the volume of data to be transmitted does not depend on the privacy policy. In contrast, the volume of data transmitted during upload (checkout) operations does depend on the privacy policy employed because, at the server, redundant chunks are only identified within that user's version history under the IP policy, but can be identified acrosss *all* users' version histories under the ALL policy. These differences based on storage policy are summarized in Figure 9 and affect our discussion in two ways: (1) this section (Section 5.1), which investigates the effects of privacy policy, only considers the upload operation, and (2) Figures 12 and 13 in Section 5.2 contain curves simply labeled CAS that represent the identical download behaviors of the IP and ALL policies.

	Redundancy Comparison	
	Upload (between client copy and...)	Download (between server version N and ...)
Delta	server version N-1	current client version
IP	server versions [1, N-1]	current client version
ALL	all versions/all parcels	current client version

Figure 9: Search space for identifying redundant blocks during data synchronization operations. Note that for download, the system inspects the most recent version available at the client (which may be older than $N-1$).

CAS is essential. The upload volume for each of the storage policies with and without compression is presented in Figure 10. Because the upload size for any user session includes the 256MB memory image and any hard disk chunks modified during that session, the upload data volumes vary significantly due to user activity across the 800+ checkin operations collected. Consequently, we present the data as a cumulative distribution function (CDF) plots. In the ideal case, most upload sizes would be small; therefore, curves that tend to occupy the upper left corner are better. Note that the ALL policy strictly outperforms the IP policy, which in turn, strictly outperforms the Delta policy.

The median (50^{th} percentile) and 95^{th} percentile sizes



(a) 4K, without compression

(b) 128K, without compression

(c) 512K, without compression

(d) 4K, with compression

(e) 128K, with compression

(f) 512K, with compression

Figure 10: CDF of upload sizes for different policies, without and with the use of compression.

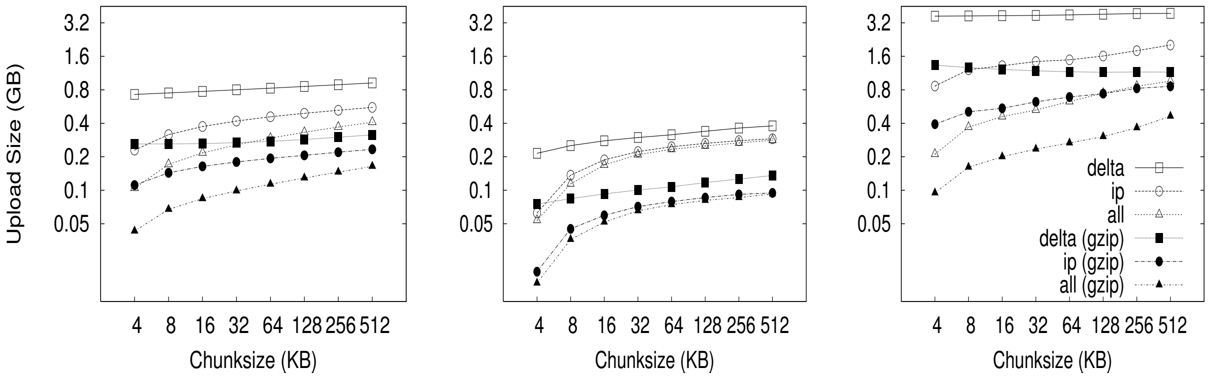
from Figure 10 are presented along with average upload sizes in Figure 11. Note that the median upload sizes tend to be substantially better than the mean sizes, indicating that the tail of the distribution is somewhat skewed in that the user will see a smaller than average upload sizes for 50% of the upload attempts. Even so, we see from Figure 11(c) that the tail is not so unwieldy as to present sizes more than a factor of 2 to 4 over the average upload size 95% of the time.

Figure 11(a) shows that, for the 128 KB chunksize used in the deployment, the use of CAS reduces the average upload size from 880 MB (Delta policy) to 340 MB (ALL policy). The use of compression reduces the upload size to 293 MB for Delta and 132 MB for ALL. Further, CAS policies provide the most significant benefits where they are needed most, for large upload sizes. From Figure 11(b) we see that CAS improves small upload operations by a modest 20 to 25 percent, while from Figure 11(c), we see that CAS improves the performance of large uploads by a factor of 2 to 5 without compression, and by a factor of 1.5 to 3 with compression. Thus, we observe that CAS significantly reduces the volume of data transmitted during upload operations, and hence the wait time experienced at the end of a user session.

CAS outperforms compression. Figure 11(a) indicates that the ALL policy *without* compression outperforms the Delta policy *with* compression for chunk sizes less than 64 KB (as does the IP policy at a 4 KB chunk size). This shows that for our application, inter-chunk CAS techniques may identify and eliminate more redundancy than traditional intra-chunk compression techniques. The difference may be substantial, particularly when the upload size is large. As Figure 11(c) shows, the ALL policy *without* compression (chunksize=4 KB) outperforms the Delta policy *with* compression (chunksize=512 KB) by a factor of 4.

IP identifies both temporal and spatial redundancy. For each of the components of Figure 10, we see that the IP policy consistently outperforms the Delta policy. Both of these policies restrict the search space for redundancy identification to a single parcel. However, the Delta policy only detects temporal redundancy between the current and last versions of the parcel, while the IP policy detects temporal and spatial redundancy across all versions of the parcel. The savings of IP over Delta indicate that users often create modified chunks in their environment that either existed at some point in the past, or in another location within the parcel.

ALL identifies *inter*-parcel savings. In all of Figure 10, the common observation between an IP and ALL comparison is that the ALL policy consistently outperforms the IP policy. This observation is consistent with our intuition that for upload operations, the ALL policy must perform *at least* as well as the IP policy because the ALL policy identifies redundancy within the set of blocks visible to the IP policy as well as blocks in other parcels. In fact, Figure 11(a) indicates that the ALL policy performs about twice as well as the IP policy for small chunk sizes and approximately 25 percent better at larger chunk sizes.



(a) Average (b) 50^{th} Percentile (c) 95^{th} Percentile

Figure 11: Upload sizes for different chunksizes.

This difference shows the benefit of having a larger pool of candidate chunks when searching for redundant data. As mentioned, one source of this gain can be the "broadcast" of objects to many users (e.g. from software installation, patches, popular documents, big email attachments, etc.). In systems leveraging the ALL policy, therefore, operations that might be expected to impose a significant burden such as the distribution of security patches may result in very little realized cost because the new data need only be stored once and transmitted once (across all users in the system).

5.2 Effect of Chunksize on Networking

The choice of chunksize will affect both the download size and upload size to a server. We continue our discussion of upload operations first, and then discuss the appropriate chunksize for download operations.

5.2.1 Effect on Upload Size

Smaller chunksize is better for CAS. Figure 11(a) shows very clearly that smaller chunksizes result in more efficient upload transmission for CAS policies. In fact, under the ALL policy, users with 4 KB chunk sizes will experience average upload sizes that are approximately one-half the average size experienced by users with a 128 KB chunk size (whether compression is employed or not).

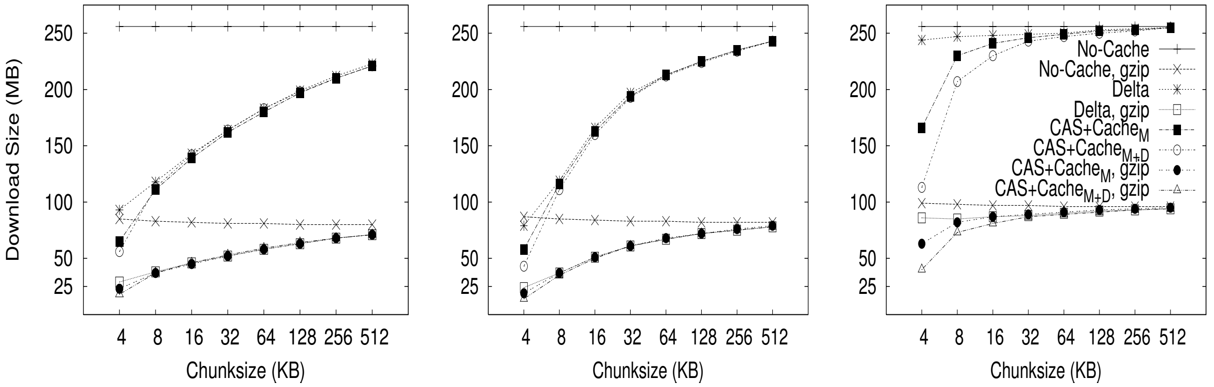
Chunk sizes of 4 KB turned out to be optimal for all policies when considering the average upload size. However, chunksize plays a very limited role for the non-CAS (Delta) policy, and Figure 11(c) indicates that smaller chunk sizes may even be a liability for transfer size outliers under the Delta policy with compression.

5.2.2 Effect on Download Size

Employing CAS techniques also potentially affects the volume of data transmitted during download operations in two ways. First, CAS can identify intra-version redundancy and reduce the total volume of data transmission. Second, when a user requests a download of their environment to a particular client, CAS has the potential to expose any chunks selected for download that are identical to chunks that happen to have been cached on that client from previous sessions.

To simplify our discussion we assume that the client has cached at most one previous version of the parcel in question, and if a cached version is present, it is the version prior to the one requested for download. This assumption corresponds to an expected common user telecommuting behavior. Namely, the user creates version $N-1$ of a parcel at home and uploads it to the server. The user then retrieves version $N-1$ at work, creates version N, and uploads that to the server. Our operation of interest is the user's next download operation at home; upon returning home, the user desires to download version N and modify it. Fortunately, the user may still have version $N-1$ cached locally, and thus, only the modified data that does not exist in the cache need be retrieved. Note that this CAS technique can be likened to a sub-set of the IP policy which inspects chunks of a single user, but only for a single previous version.

Our client management system, ISR, supports two basic modes for download: *demand-fetch* and *complete-fetch*. Demand-fetch mode instantiates the user's environment after downloading the minimum data needed to reconstruct the user's environment, essentially the physical memory image corresponding to the user's VM (256 MB in our test deployment). In particular, the largest portion of the VM image, the virtual disk drive, is *not* retrieved before instantiating the user's environment.



(a) Average (b) 50^{th} Percentile (c) 95^{th} Percentile

Figure 12: Download size when fetching memory image of latest version.

During operation, missing data blocks (chunks) must be fetched on demand in a manner analogous to demand-paging in a virtual memory system. The complete-fetch mode, in contrast, requires that the entire VM image including the virtual disk image (8.25 GB in our test deployment) be present at the client before the environment is instantiated.

Caching improves demand-fetch. To evaluate the effect of client-side caching on demand-fetch download volume, we calculated how much data would need to be transferred from the server to a client under various conditions and collected those results in Figure 12. The curve labeled “No-cache” depicts the volume of data that would be transmitted if no data from the previous version of the parcel were present in the client cache. Under the “Delta” policy, the chunks in the memory image are compared with the same chunks (those at the same offset within the image) in the previous version of the memory image to determine whether they match. The “CAS+Cache$_M$” policy compares the keyring for the new memory image with the keyring for the previous memory image to determine which chunks need to be transferred. The “CAS+Cache$_{M+D}$” policy is similar except that it searches all the data cached on the client (memory *and* disk) to identify chunks that are already present on the client. Each basic curve in Figure 12 also has a companion curve depicting the download volumes observed when compression is employed during the transfer.

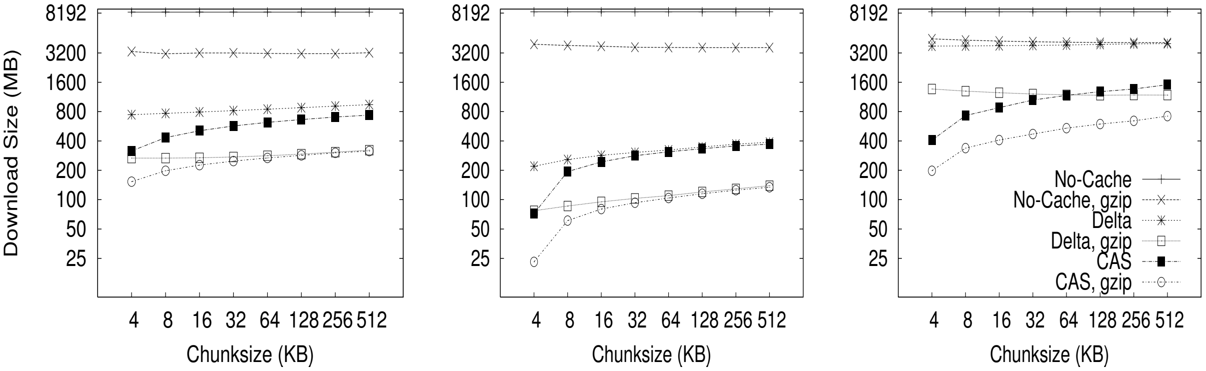
As shown in Figure 12(a), introducing a differencing mechanism (either Delta or CAS) yields a reduction of approximately 20% (for the 128 KB chunk size) in the download size relative to the size when no cached copy is present. Using compression alone, however, is very effective– reducing the transfer size from 256 MB to approximately 75 MB in the absence of caching. Leveraging cached data in addition to compression yields a further 20% reduction.

Chunk size dramatically affects demand-fetch. Moving to a smaller chunk size can have a significant effect on the volume of data transmitted during a download operation, particularly if compression is not used, as shown in Figure 12. The average download size, in particular, is reduced by a factor of two (for Delta) to four (for “CAS+Cache$_{M+D}$”) when the chunk size is reduced from 128 KB to 4 KB when comparing the policies either with or without compression. Further, we see again that, with a 4 KB chunk size, the CAS policies *without* compression outperform the no-cache policy *with* compression.

The difference between the “CAS+Cache$_M$” and “CAS+Cache$_{M+D}$” policies is also most apparent with a 4 KB chunk size. At this size, in the absence of compression, leveraging the cached disk image in addition to the memory image reduces the average transfer size to 56 MB from the 65 MB required when leveraging just the memory image. A similar gain is observed when compression is employed; the transfer size is reduced from 23 MB (for “M+D”) to 18 MB (for “M”)– a savings of more than 20%.

However, the added benefit of inspecting additional cached data diminishes quickly as the chunk size increases beyond 4 KB. We believe this phenomenon is due, at least in part, to the fact that the 4 KB size corresponds to the size of both memory pages disk blocks in these VMs. Consequently, potentially redundant data is most likely to be exposed when chunks are aligned to 4 KB boundaries.

Caching significantly improves complete-fetch. The need for efficient download mechanisms is perhaps greatest in the complete-fetch mode due to the volume of data in question. In this mode, the user is requesting the download of the entire VM image, the most signifi-



(a) Average

(b) 50^{th} Percentile

(c) 95^{th} Percentile

Figure 13: Download size when fetching memory *and disk* of latest version.

cant component of which is the virtual disk drive image. In our test deployment, the virtual disk drive was a very modest 8 GB in size. One can readily imagine that users might desire virtual disk drive spaces an order of magnitude larger. However, even with a modest size (8 GB) and a fast network (100 Mbps), a complete-fetch download will require at least 10 minutes. Consequently, reducing the volume of data to be transferred by at least an order of magnitude is essential to the operation of these client management systems.

The basic tools are the same as those mentioned for demand-fetch mode. That is, a cache of at least one previous version of the parcel is maintained at the client, if possible. Redundancy between the cached version and the current version on the server is identified and only non-redundant chunks are transferred during the download. Further, the transferred chunks are (optionally) compressed prior to transmission. One difference between our treatment of demand-fetch and complete-fetch is that the CAS policy for complete-fetch mode always compares the entire current server version with the entire cached client version. Consequently, Figure 13 includes a single "CAS" curve rather than the separate "M" and "M+D" curves of Figure 12.

Figure 13(a) indicates that intelligent transfer mechanisms can, in fact, significantly reduce the volume of data transmitted during a complete-fetch operation. Compression reduces the average data volume from 8394 MB to 3310 MB, a factor of 2.7. In contrast, the Delta policy *without* compression yields a factor of 9.5 and a factor of 28.6 *with* compression, assuming a 128 KB chunk size. At the same chunk size, CAS provides even more impressive savings: factors of 12.6 and 29.5, without and with compression, respectively.

Small chunk sizes yield additional savings. While the slopes of the "CAS" and "CAS,gzip" curves are not as dramatic as in previous figures, reducing the chunk size from 128 KB to 4 KB still yields significant savings. At this chunk size, the average download size shrinks from the nominal 8+ GB size by a factor of 31.4 without compression and a factor of 55 (*fifty-five!*) by employing both CAS and compression.

CAS has a big impact where it's needed most. Figure 13(c) indicates that the 4 KB "CAS,gzip" combination may be particularly effective for download operations that may otherwise have resulted in large data transfers. The performance gap between "CAS,gzip" and "Delta,gzip" is particularly large in this graph. In fact, for small chunk sizes "CAS" *without* compression significantly outperforms the Delta policy *with* compression. Note in particular that when employing the "CAS,gzip" policy with the 4 KB chunk size, the 95^{th} percentile upload sizes are not significantly larger than the average size, thus providing the user with better expected bounds on the time required for a complete-fetch download.

6 Related Work

Our results are most directly applicable to VM-based client management systems such as the Collective [7, 35], Soulpad [5], and ISR [19, 37], as well as systems that use VMs for Grid applications [9, 14, 22, 24, 39]. Further, our results also provides guidelines for the storage design of applications that need to version VM history. Examples include intrusion detection [12], operating systems development [18], and debugging system configurations [46]. Related applications include storage cluster and web services where VMs are being used for balancing load, increasing availability, and simplifying administration [28, 45].

The study could also help a large number of systems that use use CAS to improve storage and network utilization. Examples of CAS-based storage systems include EMC's Centera [13], Deep Store [48], the Venti [30], the

Pastiche [10] backup system, the TAPER [17] scheme for replica synchronization and Farsite [2]. Other systems use similar CAS-based techniques to eliminate duplicate data at various levels in the network stack. Systems such as the CASPER [42] and LBFS [27] file systems, Rhea et al.'s CAS-enabled WWW [33], etc. apply these optimizations at the application layer. Other solutions such as the DOT transfer service [41] and Riverbed's WAN accelerator [34] use techniques such as Rabin Fingerprinting [26, 31, 38] to detect data duplication at the transfer layer. However, most of these systems have only concentrated on the mechanism behind using CAS. Apart from Bolosky et al. [3] and Policroniades and Pratt [29], there have been few studies that measure data commonality in real workloads. The study in this paper helps by providing a point of reference for commonality seen in VM migration workloads.

7 Conclusions

Managing large volumes of data is one of the major challenges inherent in developing and maintaining enterprise client management systems based on virtual machines. Using empirical data collected during seven-months of a live-deployment of one such system, we conclude that leveraging content addressable storage (CAS) technology can significantly reduce the storage and networking resources required by such a system (questions Q1 and Q2 from Section 1).

Our analysis indicates that CAS-based management policies typically benefit from dividing the data into very small chunk sizes despite the associated meta-data overhead. In the absence of compression, 4 KB chunks yielded the most efficient use of both storage and network resources. At this chunk size, a privacy-preserving CAS policy can reduce the system storage requirements by approximately 60% when compared to a block-based differencing policy (*Delta*), and a savings of approximately 80% is possible by relaxing privacy.

Similarly, CAS policies that leverage data cached on client machines reduce the average quantity of data that must be transmitted during both upload and download operations. For upload, this technique again results in a savings (compared to Delta) of approximately 70% when preserving privacy and 80% when not. This technique also reduces the cost of *complete-fetch* download operations by more than 50% relative to the Delta policy (irrespective of CAS privacy policy) and by more than an order of magnitude relative to the cost when caching is not employed.

Leveraging compression in addition to CAS techniques provides additional resource savings, and the combination yields the highest efficiency in all cases. However, a surprising finding from this work is that CAS alone yields higher efficiency for this data set than compression alone, which is significant because the use of compression incurs a non-zero runtime cost for these systems.

Acknowledgments

This work is sponsored in part by NSF under grants CNS-0509004 and IIS-0429334, in part by a subcontract from SCEC as part of NSF ITR EAR-01-22464, and in part by support from Intel and CMU CyLab. Our heartfelt thanks to Intel's Casey Helfrich for his assistance developing the software and deployment, Dave Bantz and Ramón Cáceres at IBM for the donation of the BladeCenter, Mark Poepping and his team at CMU computing services for housing the server, and Bob Cosgrove, Tod Pike, and Mark Puskar in the CMU School of Computer Science facilities group for helping integrate our experiment into the campus environment. All unidentified trademarks are the property of their respective owners.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, 2003.

[2] W. J. Bolosky, S. Corbin, D. Goebel, , and J. R. Douceur. Single Instance Storage in Windows 2000. In *Proceedings of the 4th USENIX Windows Systems Symposium*, 2000.

[3] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. *ACM SIGMETRICS Performance Evaluation Review*, 28(1):34–43, 2000.

[4] T. C. Bressoud and F. B. Schneider. Hypervisor-based fault tolerance. In *SOSP '95: Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, 1995.

[5] R. Caceres, C. Carter, C. Narayanaswami, and M. Raghunath. Reincarnating PCs with portable SoulPads. In *MobiSys '05: Proceedings of the 3rd International conference on Mobile Systems, Applications, and Services*, 2005.

[6] B. Calder, A. A. Chien, J. Wang, and D. Yang. The entropia virtual machine for desktop grids. In *VEE '05: Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution environments*, 2005.

[7] R. Chandra, N. Zeldovich, C. Sapuntzakis, and M. S. Lam. The Collective: A Cache-Based System Management Architecture. In *Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI '05)*, Boston, MA, USA, May 2005.

[8] P. Chen and B. Noble. When Virtual is Better Than Real. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems*, 2001.

[9] S. Childs, B. A. Coghlan, D. O'Callaghan, G. Quigley, and J. Walsh. A single-computer grid gateway using virtual machines. In *Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA 2005)*, pages 310–315, Taipei, Taiwan, Mar. 2005.

[10] L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: Making Backup Cheap and Easy. In *OSDI: Symposium on Operating Systems Design and Implementation*, 2002.

[11] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 617, Washington, DC, USA, 2002. IEEE Computer Society.

[12] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen. ReVirt: Enabling Intrusion Analysis Through Virtual-Machine Logging and Replay. In *OSDI*, 2002.

[13] EMC Corporation. *EMC Centera Content Addressed Storage System*, 2003. http://www.emc.com/.

[14] R. J. Figueiredo, P. A. Dinda, and J. A. B. Fortes. A case for grid computing on virtual machines. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS '03)*, page 550, Washington, DC, USA, 2003. IEEE Computer Society.

[15] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: a virtual machine-based platform for trusted computing. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003.

[16] K. Govil, D. Teodosiu, Y. Huang, and M. Rosenblum. Cellular Disco: Resource Management using Virtual Clusters on Shared-Memory Multiprocessors. In *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*, 1999.

[17] N. Jain, M. Dahlin, and R. Tewari. Taper: Tiered approach for eliminating redundancy in replica synchronization. In *USENIX Conference on File and Storage Technologies*, Dec 2005.

[18] S. T. King, G. W. Dunlap, and P. M. Chen. Debugging Operating Systems with Time-Traveling Virtual Machines. In *Proceedings of the USENIX 2005 Annual Technical Conference*, pages 1–15, Anaheim, CA, Apr. 2005.

[19] M. Kozuch and M. Satyanarayanan. Internet Suspend/Resume. In *Fourth IEEE Workshop on Mobile Computing Systems and Applications*, Callicoon, New York, June 2002.

[20] M. A. Kozuch, C. J. Helfrich, D. O'Hallaron, and M. Satyanarayanan. Enterprise Client Management with Internet Suspend/Resume. *Intel Technology Journal*, November 2004.

[21] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo. VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, 2004.

[22] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo. Vmplants: Providing and managing virtual machine execution environments for grid computing. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 7, Washington, DC, USA, 2004. IEEE Computer Society.

[23] P. Kulkarni, F. Douglis, J. D. LaVoie, and J. M. Tracey. Redundancy Elimination Within Large Collections of Files. In *USENIX Annual Technical Conference, General Track*, 2004.

[24] B. Lin and P. Dinda. Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling. In *Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing (SC 2005)*, Seattle, WA, Nov. 2005.

[25] D. E. Lowell, Y. Saito, and E. J. Samberg. Devirtualizable virtual machines enabling general, single-node, online maintenance. In *ASPLOS-XI: Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, 2004.

[26] U. Manber. Finding Similar Files in a Large File System. In *Proceedings of the USENIX Winter 1994 Technical Conference*, 1994.

[27] A. Muthitacharoen, B. Chen, and D. Mazieres. A Low-Bandwidth Network File System. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, 2001.

[28] M. Nelson, B.-H. Lim, and G. Hutchins. Fast Transparent Migration for Virtual Machines. In *Proceedings of the USENIX 2005 Annual Technical Conference*, Anaheim, CA, Apr. 2005.

[29] C. Policroniades and I. Pratt. Alternatives for Detecting Redundancy in Storage Systems Data. In *USENIX Annual Technical Conference, General Track*, 2004.

[30] S. Quinlan and S. Dorward. Venti: A New Approach to Archival Storage. In *Proceedings of the FAST 2002 Conference on File and Storage Technologies*, 2002.

[31] M. Rabin. Fingerprinting by Random Polynomials. In *Harvard University Center for Research in Computing Technology Technical Report TR-15-81*, 1981.

[32] J. Reumann, A. Mehra, K. G. Shin, and D. D. Kandlur. Virtual Services: A New Abstraction for Server Consolidation. In *USENIX Annual Technical Conference, General Track*, 2000.

[33] S. Rhea, K. Liang, and E. Brewer. Value-Based Web Caching. In *Proceedings of the Twelfth International World Wide Web Conference*, 2003.

[34] Riverbed Technology, Inc. http://www.riverbed.com.

[35] C. Sapuntzakis and M. S. Lam. Virtual Appliances in the Collective: A Road to Hassle-Free Computing. In *Proceedings of the Ninth Workshop on Hot Topics in Operating System*, May 2003.

[36] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum. Optimizing the Migration of Virtual Computers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, December 2002.

[37] M. Satyanaranyanan, M. A. Kozuch, C. J. Helfrich, and D. R. O'Hallaron. Towards Seamless Mobility on Pervasive Hardware. *Pervasive and Mobile Computing*, 1(2):157–189, July 2005.

[38] N. T. Spring and D. Wetherall. A Protocol-Independent Technique for Eliminating Redundant Network Traffic. In *Proceedings of ACM SIGCOMM*, August 2000.

[39] A. I. Sundararaj and P. A. Dinda. Towards virtual networks for virtual machine grid computing. In *Proceedings of the 3rd Virtual Machine Research and Technology Symposium*, pages 177–190, San Jose, CA, May 2004.

[40] N. Taesombut and A. Chien. Distributed Virtual Computer (DVC): Simplifying the Development of High Performance Grid Applications. In *Proceedings of the Workshop on Grids and Advanced Networks (GAN'04)*, 2004.

[41] N. Tolia, M. Kaminsky, D. G. Andersen, and S. Patil. An architecture for internet data transfer. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI '06)*, San Jose, CA, May 2006.

[42] N. Tolia, M. Kozuch, M. Satyanarayanan, B. Karp, A. Perrig, and T. Bressoud. Opportunistic use of content addressable storage for distributed file systems. In *Proceedings of the 2003 USENIX Annual Technical Conference*, pages 127–140, San Antonio, TX, June 2003.

[43] M. Vilayannur, P. Nath, and A. Sivasubramaniam. Providing Tunable Consistency for a Parallel File Store. In *Proceedings of the Fourth USENIX Conference on File and Storage Technologies (FAST'05)*, 2005.

[44] C. A. Waldspurger. Memory Resource Management in VMware ESX Server. In *Proceedings of the 5th Symposium on Operating System Design and Implementation*, Boston, MA, USA, 2002.

[45] A. Warfield, R. Ross, K. Fraser, C. Limpach, and S. Hand. Parallax: Managing Storage for a Million Machines. In *Proc. 10th Workshop on Hot Topics in Operating Systems (HotOS)*, 2005.

[46] A. Whitaker, R. S. Cox, and S. D. Gribble. Configuration Debugging as Search: Finding the Needle in the Haystack. In *OSDI*, 2004.

[47] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and Performance in the Denali Isolation Kernel. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Dec 2002.

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

[49] Y. Zhang, A. Bestavros, M. Guirguis, I. Matta, and R. West. Friendly Virtual Machines: Leveraging a Feedback-Control Model for Application Adaptation. In *VEE '05: Proc. 1st ACM/USENIX Inter. Conf. on Virtual Execution Envs*, 2005.

[50] J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.