A Study of Practical Deduplication

Dutch T. Meyer[*†] and William J. Bolosky[*]
[*]Microsoft Research and [†]The University of British Columbia
{dmeyer@cs.ubc.edu, bolosky@microsoft.com}

Abstract

We collected file system content data from 857 desktop computers at Microsoft over a span of 4 weeks. We analyzed the data to determine the relative efficacy of data deduplication, particularly considering whole-file versus block-level elimination of redundancy. We found that whole-file deduplication achieves about three quarters of the space savings of the most aggressive block-level deduplication for storage of live file systems, and 87% of the savings for backup images. We also studied file fragmentation finding that it is not prevalent, and updated prior file system metadata studies, finding that the distribution of file sizes continues to skew toward very large unstructured files.

1 Introduction

File systems often contain redundant copies of information: identical files or sub-file regions, possibly stored on a single host, on a shared storage cluster, or backed-up to secondary storage. Deduplicating storage systems take advantage of this redundancy to reduce the underlying space needed to contain the file systems (or backup images thereof). Deduplication can work at either the sub-file [10, 31] or whole-file [5] level. More fine-grained deduplication creates more opportunities for space savings, but necessarily reduces the sequential layout of some files, which may have significant performance impacts when hard disks are used for storage (and in some cases [33] necessitates complicated techniques to improve performance). Alternatively, whole-file deduplication is simpler and eliminates file-fragmentation concerns, though at the cost of some otherwise reclaimable storage.

Because the disk technology trend is toward improved sequential bandwidth and reduced per-byte cost with little or no improvement in random access speed, it's not clear that trading away sequentiality for space savings makes sense, at least in primary storage.

In order to evaluate the tradeoff in space savings between whole-file and block-based deduplication, we conducted a large-scale study of file system contents on desktop Windows machines at Microsoft. Our study consists of 857 file systems spanning 162 terabytes of disk over 4 weeks. It includes results from a broad cross-section of employees, including software developers, testers, management, sales & marketing, technical support, documentation writers and legal staff. We find that while block-based deduplication of our dataset can lower storage consumption to as little as 32% of its original requirements, nearly three quarters of the improvement observed could be captured through whole-file deduplication and sparseness. For four weeks of full backups, whole file deduplication (where a new backup image contains a reference to a duplicate file in an old backup) achieves 87% of the savings of block-based. We also explore the parameter space for deduplication systems, and quantify the relative benefits of sparse file support. Our study of file content is larger and more detailed than any previously published effort, which promises to inform the design of space-efficient storage systems.

In addition, we have conducted a study of metadata and data layout, as the last similar study [1] is now 4 years old. We find that the previously observed trend toward storage being consumed by files of increasing size continues unabated; half of all bytes are in files larger than 30MB (this figure was 2MB in 2000). Complicating matters, these files are in opaque unstructured formats with complicated access patterns. At the same time there are increasingly many small files in an increasingly complex file system tree.

Contrary to previous work [28], we find that file-level fragmentation is not widespread, presumably due to regularly scheduled background defragmenting in Windows [17] and the finding that a large portion of files are rarely modified (see Section 4.4.2). For more than a decade, file system designers have been warned against measuring only fresh file system installations, since aged systems can have a significantly different performance profile [28]. Our results show that this concern may no longer be relevant, at least to the extent that the aging produces file-level fragmentation. Ninety-six percent of files observed are entirely linear in the block address space. To our knowledge, this is the first large scale study of disk fragmentation in the wild.

We describe in detail the novel analysis optimizations necessitated by the size of this data set.

2 Methodology

Potential participants were selected randomly from Microsoft employees. Each was contacted with an offer to install a file system scanner on their work computer(s) in exchange for a chance to win a prize. The scanner ran autonomously during off hours once per week from September 18 – October 16, 2009. We contacted 10,500 people in this manner to reach the target study size of about 1000 users. This represents a participation rate of roughly 10%, which is smaller than the rates of 22% in similar prior studies [1, 9]. Anecdotally, many potential participants declined explicitly because the scanning process was quite invasive.

2.1 File system Scanner

The scanner first took a consistent snapshot of fixed device (non-removable) file systems with the Volume Shadow Copy Service (VSS) [20]. VSS snapshots are both file system and application consistent[1]. It then recorded metadata about the file system itself, including age, capacity, and space utilization. The scanner next processed each file in the snapshot, writing records to a log. It recorded Windows file metadata [19], including path, file name and extension, time stamps, and the file attribute flags. It recorded any retrieval and allocation pointers, which describe fragmentation and sparseness respectively. It also recorded information about the whole system, including the computer's hardware and software configuration and the time at which the defragmentation tool was last run, which is available in the Windows registry. We took care to exclude from study the pagefile, hibernation file, the scanner itself, and the VSS snapshots it created.

During the scan, we recorded the contents of each file first by breaking the file into chunks using each of two chunking algorithms (fixed block and Rabin fingerprinting [25]) with each of 4 chunk size settings (8K-64K in powers of two) and then computed and saved hashes of each chunk. We found whole file duplicates in post-processing by identifying files in which all chunks matched. In addition to reading the ordinary contents of files we also collected a separate set of scans where the files were read using the Win32 BackupRead API [16], which includes metadata about the file and would likely be the format used to store file system backups.

We used salted MD5 [26] as our hash algorithm, but truncated the result to 48 bits in order to reduce the size of the data set. The Rabin-chunked data with an 8K target chunk size had the largest number of unique hashes, somewhat more than 768M. We expect that about two thousand of those (0.0003%) are false matches due to the truncated hash.

Another process copied the log files to our server at midnight on a random night of the week to help smooth the considerable network traffic. Nevertheless, the copying process resulted in the loss of some of the scans. Because the scanner placed the results for each of the 32 parameter settings into separate files and the copying process worked at the file level, for some file systems we have results for some, but not all of the parameter settings. In particular, larger scan files tended to be partially copied more frequently than smaller ones, which may result in a bias in our data where larger file systems are more likely to be excluded. Similarly, scans with a smaller chunk size parameter resulted in larger size scan files and so were lost at a higher rate.

2.2 Post Processing

At the completion of the study the resulting data set was 4.12 terabytes compressed, which would have required considerable machine time to import into a database. As an optimization, we observed that the actual value of any unique hash (i.e., hashes of content that was not duplicated) was not useful to our analyses.

To find these unique hashes quickly we used a novel 2-pass algorithm. During the first pass we created a 2 GB Bloom filter [4] of each hash observed. During this pass, if we tried to insert a value that was already in the Bloom filter, we inserted it into a second Bloom filter of equal size. We then made a second pass through the logs, comparing each hash to the second Bloom filter only. If it was not found in the second filter, we were certain that the hash had been seen exactly once and could be omitted from the database. If it was in the filter, we concluded that either the hash value had been seen more than once, or that its entry in the filter was a collision. We recorded all of these values to the database. Thus this algorithm was sound, in that it did not impact the results by rejecting any duplicate hashes.

[1] "Application consistent" means that VSS-aware applications have an opportunity to save their state cleanly before the snapshot is taken.

However it was not complete despite being very effective, in that some non-duplicate hashes may have been added to the database even though they were not useful in the analysis. The inclusion of these hashes did not affect our results, as the later processing ignored them.

2.3 Biases and Sources of Error

The use of Windows workstations in this study is beneficial in that the results can be compared to those of similar studies [1, 9]. However, as in all data sets, this choice may introduce biases towards certain types of activities or data. For example, corporate policies surrounding the use of external software and libraries could have impacted our results.

As discussed above, the data retrieved from machines under observation was large and expensive to generate and so resulted in network timeouts at our server or aborted scans on the client side. While we took measures to limit these effects, nevertheless some amount of data never made it to the server, and more had to be discarded as incomplete records. Our use of VSS makes it possible for a user to selectively remove some portions of their file system from our study.

We discovered a rare concurrency bug in the scanning tool affecting 0.003% of files. While this likely did not affect results, we removed all files with this artifact.

Our scanner was unable to read the contents of Windows system restore points, though it could see the file metadata. We excluded these files from the deduplication analyses, but included them in the metadata analyses.

3 Redundancy in File Contents

Despite the significant declines in storage costs per GB, many organizations have seen dramatic increases in total storage system costs [21]. There is considerable interest in reducing these costs, which has given rise to deduplication techniques, both in the academic community [6] and as commercial offerings [7, 10, 14, 33]. Initially, the interest in deduplication has centered on its use in “embarrassingly compressible” scenarios, such as regular full backups [3, 8] or virtual desktops [6, 13]. However, some have also suggested that deduplication be used more widely on general purpose data sets [31].

The rest of this section seeks to provide a well-founded measure of duplication rates and compare the efficacy of different parameters and methods of deduplication. In Section 3.1 we provide a brief summary of deduplication, and in Section 3.2 we discuss the performance challenges deduplication introduces. In Section 3.3 we share observed duplication rates across a set of workstations. Finally, Section 3.4 measures duplication in the more conventional backup scenario.

3.1 Background on Deduplication

Deduplication systems decrease storage consumption by identifying distinct chunks of data with identical content. They then store a single copy of the chunk along with metadata about how to reconstruct the original files from the chunks.

Chunks may be of a predefined size and alignment, but are more commonly of variable size determined by the content itself. The canonical algorithm for variable-sized content-defined blocks is Rabin Fingerprints [25]. By deciding chunk boundaries based on content, files that contain identical content that is shifted (say because of insertions or deletions) will still result in (some) identical chunks. Rabin-based algorithms are typically configured with a minimum and maximum chunk size, as well as an expected chunk size. In all our experiments, we set the minimum and maximum parameters to 4K and 128K, respectively while we varied the expected chunk size from 8K to 64K by powers-of-two.

3.2 The Performance Impacts of Deduplication

Managing the overheads introduced by a deduplication system is challenging. Naively, each chunk’s fingerprint needs to be compared to that of all other chunks. While techniques such as caches and Bloom filters can mitigate overheads, the performance of deduplication systems remains a topic of research interest [32]. The I/O system also poses a performance challenge. In addition to the layer of indirection required by deduplication, deduplication has the effect of de-linearizing data placement, which is at odds with many data placement optimizations, particularly on hard-disk based storage where the cost for non-sequential access can be orders of magnitude greater than sequential.

Other more established techniques to reduce storage consumption are simpler and have smaller performance impact. Sparse file support exists in many file systems including NTFS [23], XFS [29], and ext4 [15] and is relatively simple to implement. In a sparse file a chunk of zeros is stored notationally by marking its existence in the metadata, removing the need to physically store it. Whole file deduplication systems, such as the Windows SIS facility [5] operate by finding entire files that

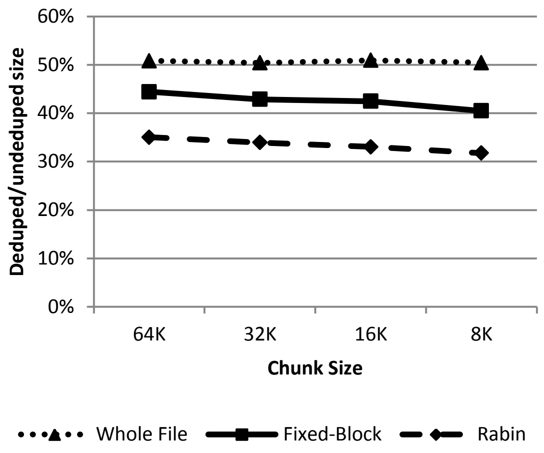

Figure 1: Deduplication vs. Chunk Size for Various Algorithms

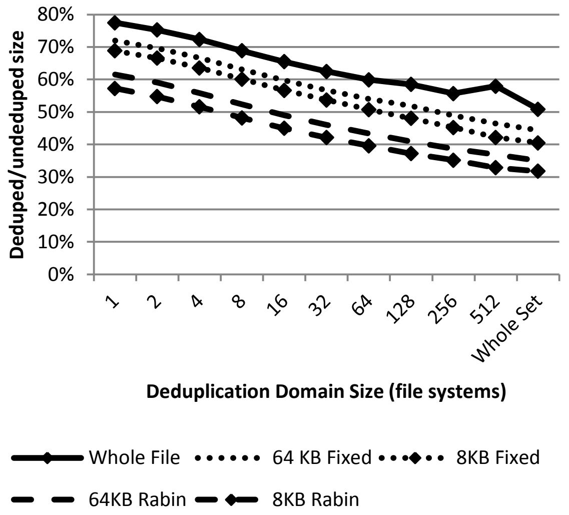

Figure 2: Deduplication vs. Deduplication Domain Size

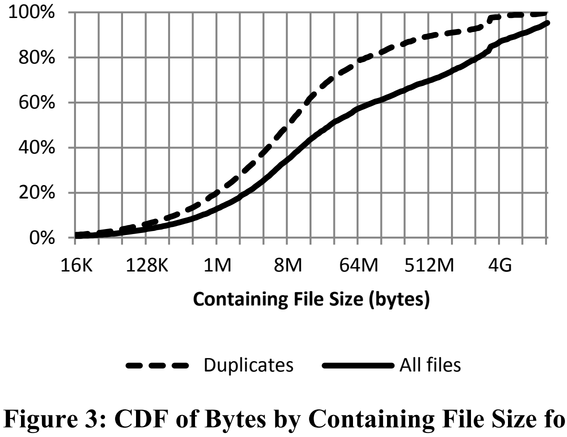

Figure 3: CDF of Bytes by Containing File Size for Whole File Duplicates and All Files

Extension	% of Duplicate Space	Mean File Size (bytes)
dll	20%	521K
lib	11%	1080K
pdb	11%	2M
<none>	7%	277K
exe	6%	572K
cab	4%	4M
msp	3%	15M
msi	3%	5M
iso	2%	436M
<a guid>	1%	604K
hxs	1%	2M
xml	1%	49K
jpg	1%	147K
wim	1%	16M
h	1%	23K

Table 1: Whole File Duplicates by Extension

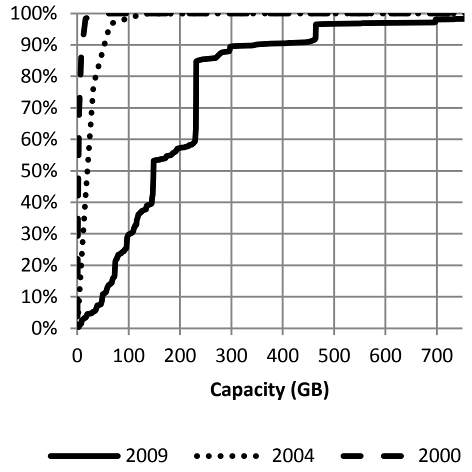

Figure 4: CDF of File System Capacity

Extension	Fixed %	Extension	Rabin %
vhd	3.6%	vhd	5.2%
pch	0.5%	lib	1.6%
dll	0.5%	obj	0.8%
pdb	0.4%	pdb	0.6%
lib	0.4%	pch	0.6%
wma	0.3%	iso	0.6%
pst	0.3%	dll	0.6%
<none>	0.3%	avhd	0.5%
avhd	0.3%	wma	0.4%
mp3	0.3%	wim	0.4%
pds	0.2%	zip	0.3%
iso	0.2%	pst	0.3%

Table 2: Non-whole File, Non-Zero Duplicate Data as a Fraction of File System Size by File Extension, 8K Fixed and Rabin Chunking

are duplicates and replacing them by copy-on-write links. Although SIS does not reduce storage consumption as much as a modern deduplication system, it avoids file allocation concerns and is far less computationally expensive than more exhaustive deduplication.

3.3 Deduplication in Primary Storage

Our data set includes hashes of data in both variable and fixed size chunks, and of varying sizes. We chose a single week (September 18, 2009) from this dataset and compared the size of all unique chunks to the total consumption observed. We had two parameters that we could vary: the deduplication algorithm/parameters and the set of file systems (called the *deduplication domain)* within which we found duplicates; duplicates in separate domains were considered to be unique contents.

The set of file systems included corresponds to the size of the file server(s) holding the machines' file systems. A value of 1 indicates deduplication running independently on each desktop machine. "Whole Set" means that all 857 file systems are stored together in a single deduplication domain. We considered all power-of-two domain sizes between 1 and 857. For domain sizes other than 1 or 857, we had to choose which file systems to include together into particular domains and which to exclude when the number of file systems didn't divide evenly by the size of the domain. We did this by using a cryptographically secure random number generator. We generated sets for each domain size ten times and report the mean of the ten runs. The standard deviation of the results was less than 2% for each of the data points, so we don't believe that we would have gained much more precision by running more trials[2].

Rather than presenting a three dimensional graph varying both parameters, we show two slices through the surface. In both cases, the y-axis shows the deduplicated file system size as a percentage of the original file system size. Figure 1 shows the effect of the chunk size parameter for the fixed and Rabin-chunked algorithms, and also for the whole file algorithm (which doesn't depend on chunk size, and so varies only slightly due to differences in the number of zeroes found and due to variations in which file systems scans copied properly; see Section 3.2). This graph assumes that all file systems are in a single deduplication domain; the shape of the curve is similar for smaller domains, through the space savings are reduced.

Figure 2 shows the effect changing the size of the deduplication domains. Space reclaimed improves roughly linearly in the log of the number of file systems in a domain. Comparing single file systems to the whole set, the effect of grouping file systems together is larger than that from the choice of chunking algorithm or chunk size, or even of switching from whole file chunking to block-based.

The most aggressive chunking algorithm (8K Rabin) reclaimed between 18% and 20% more of the total file size than did whole file deduplication. This offers weak support for block-level deduplication in primary storage. The 8K fixed block algorithm reclaimed between 10% and 11% more space than whole file. This capacity savings represents a small gain compared to the performance and complexity of introducing advanced deduplication features, especially ones with dynamically variable block sizes like Rabin fingerprinting.

Table 1 shows the top 15 file extensions contributing to duplicate content for whole file duplicates, the percentage of duplicate space attributed to files of that type, and the mean file size for each type. It was calculated using all of the file systems in a single deduplication domain. The extension marked <a guid> is a particular globally unique ID that's associated with a widely distributed software patch. This table shows that the savings due to whole file duplicates are concentrated in files containing program binaries: dll, lib, pdb, exe, cab, msp, and msi together make up 58% of the saved space.

Figure 3 shows the CDF of the bytes reclaimed by whole file deduplication and the CDF of all bytes, both by containing file size. It shows that duplicate bytes tend to be in smaller files than bytes in general. Another way of looking at this is that the very large file types (virtual hard disks, database stores, *etc.*) tend not to have whole-file copies. This is confirmed by Table 1.

Table 2 shows the amount of duplicate content not in files with whole-file duplicates by file extension as a fraction of the total file system content. It considers the whole set of file systems as a single deduplication domain, and presents results with an 8K block size using both fixed and Rabin chunking. For both algorithms, by far the largest source of duplicate data is VHD (virtual hard drive) files. Because these files are essentially disk images, it's not surprising both that they contain duplicate data and also that they rarely have whole-file duplicates. The next four file types are all compiler outputs. We speculate that they generate block-aligned duplication because they have header fields that contain, for example, timestamps but that their contents is

[2] As it was, it took about 8 machine-months to do the analyses.

otherwise deterministic in the code being compiled. Rabin chunking may find blocks of code (or symbols) that move somewhat in the file due to code changes that affect the length of previous parts of the file.

3.4 Deduplication in Backup Storage

Much of the literature on deduplication to date has relied on workloads consisting of daily full backups [32, 33]. Certainly these workloads represent the most attractive scenario for deduplication, because the content of file systems does not change rapidly. Our data set did not allow us to consider daily backups, so we considered only weekly ones.

With frequent and persistent backups, the size of historical data will quickly out-pace that of the running system. Furthermore, performance in secondary storage is less critical than in that of primary, so the reduced sequentiality of a block-level deduplicated store is of lesser concern. We considered the 483 file systems for which four continuous weeks of complete scans were available, starting with September 18, 2009, the week used for the rest of the analyses.

Our backup analysis considers each file system as a separate deduplication domain. We expect that combining multiple backups into larger domains would have a similar effect to doing the same thing for primary storage, but we did not run the analysis due to resource constraints.

In practice, some backup solutions are incremental (or differential), storing deltas between files, while others use full backups. Often, highly reliable backup policies use a mix of both, performing frequent incremental backups, with occasional full backups to limit the potential for loss due to corruption. Thus, the meaning of whole-file deduplication in a backup store is not immediately obvious. We ran the analysis as if the backups were stored as simple copies of the original file systems, except that the contents of the files was the output from the Win32 BackupRead [16] call, which includes some file metadata along with the data. For our purposes, imagine that the backup format finds whole file duplicates and stores pointers to them in the backup file. This would result in a garbage collection problem for the backup files when they're deleted, but the details of that are beyond the scope of our study and are likely to be simpler than a block-level deduplicating store.

Using the Rabin chunking algorithm with an 8K expected chunk size, block-level deduplication reclaimed 83% of the total space. Whole file deduplication, on the other hand, yielded 72%. These numbers, of course, are highly sensitive to the number of weeks of scans used in the study; it's no accident that the results were around ¾ of the space being claimed when there were four weeks of backups. However, one should not assume that because 72% of the space was reclaimed by whole file deduplication that only 3% of the bytes were in files that changed. The amount of change was larger than that, but the deduplicator found redundancy within a week as well and the two effects offset.

4 Metadata

This paper is the 3rd major metadata study of Windows desktop computers [1, 9]. This provides a unique perspective in the published literature, as we are able to track more than a decade of trends file and file system metadata. On a number of graphs, we took the lines from 2000 and 2004 from an earlier study [1] and plotted them on our graphs to make comparisons easier. Only the 2009 data is novel to this paper. Some graphs contain both CDF and histogram lines. In these graphs, the CDF should be read from the left-hand *y*-scale and the histogram from the right. We present much of our data in the form of cumulative density function plots. These plots make it easy to determine the distributions, but do not easily show the mean. Where appropriate, we list the mean of the distribution in the text.

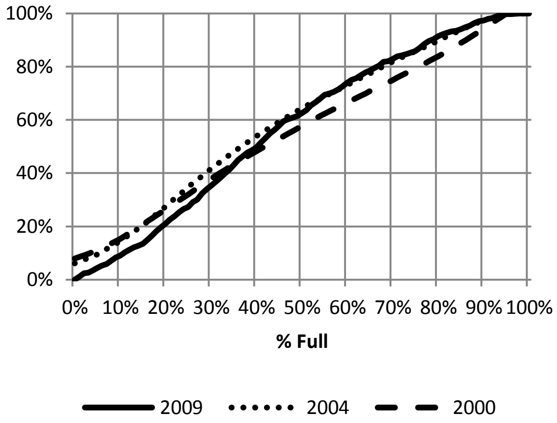

Figure 5: CDF of File Systems by Fullness

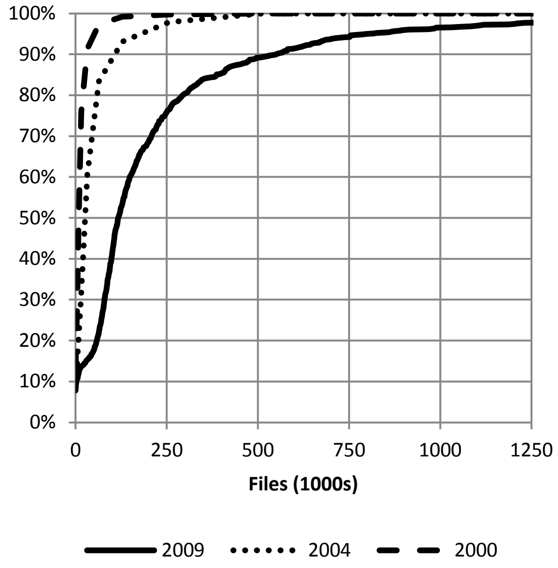

Figure 6: CDF of File Systems by Count of Files

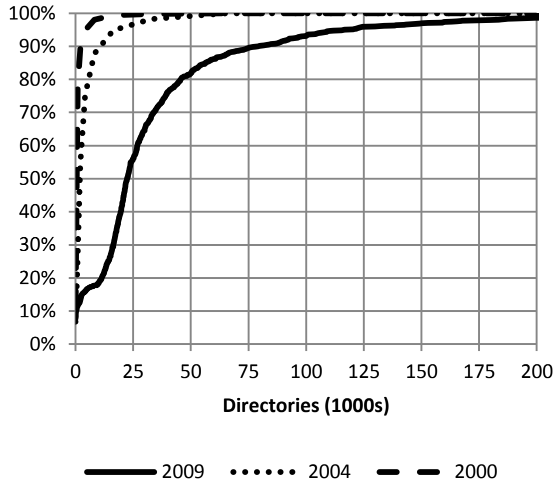

Figure 7: CDF of File Systems by Count of Directories

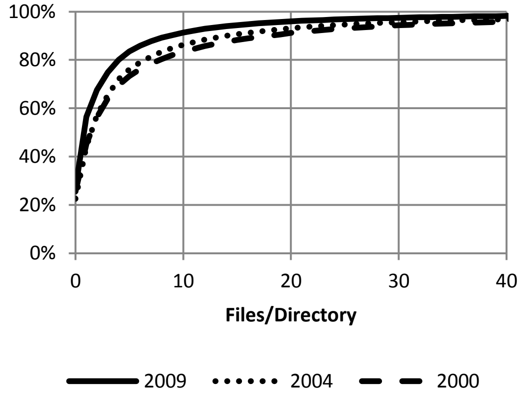

Figure 8: CDF of Directories by Count of Files

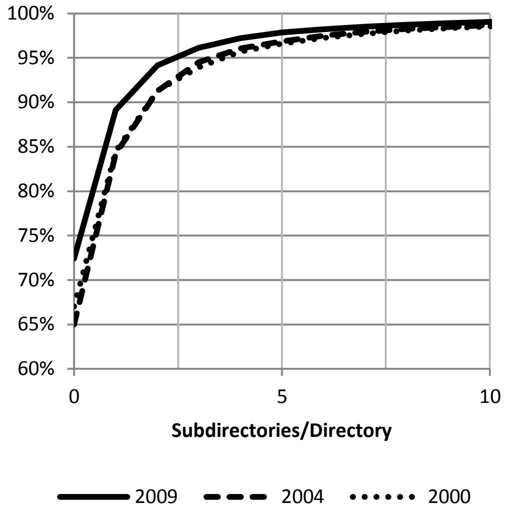

Figure 9: CDF of Directories by Count of Subdirectories

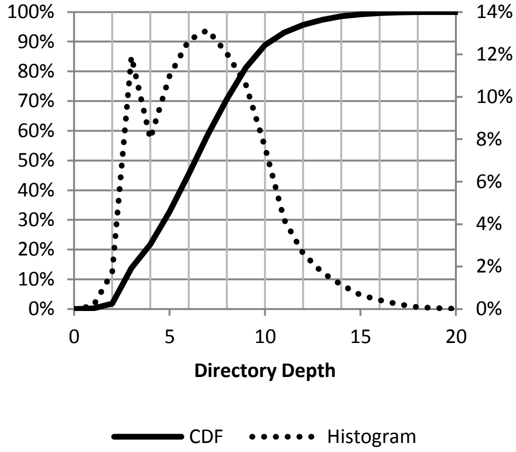

Figure 10: Files by Directory Depth

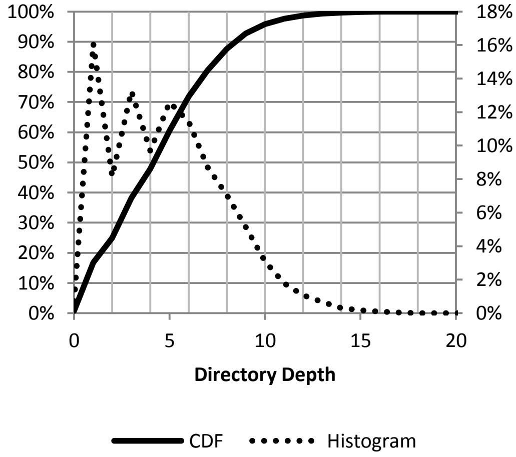

Figure 11: Bytes by Directory Depth

4.1 Physical Machines

Our data set contains scans of 857 file systems hosted on 597 computers. 59% were running Windows 7, 20% Windows Vista, 18% Windows Server 2008 and 3% Windows Server 2003. They had a mean and median physical RAM of about 4GB, and ranged from 1-10GB. 5% had 8 processors, 44% 4, 49% 2 and 3% were uniprocessors[3].

4.2 File systems

We analyze file systems in terms of their age, capacity, fullness, and the number of files and directories. We present our results, interpretations, and recommendations to designers in this section.

4.2.1 Capacity

The mean file system capacity is 194GB. Figure 4 shows a cumulative density function of the capacities of the file systems in the study. It shows a significant increase in the range of commonly observed file system sizes and the emergence of a noticeable step function in the capacities. Both of these trends follow from the approximately annual doubling of physical drive capacity. We expect that this file system capacity range will continue to increase, anchored by smaller SSDs on the left, and continuing step wise towards larger magnetic devices on the right. This will either force file systems to perform acceptably on an increasingly wide range of media, or push users towards more highly tuned special purpose file systems.

4.2.2 Utilization

Although capacity has increased by nearly two orders of magnitude since 2000, utilization of capacity has dropped only slightly, as shown in Figure 5. Mean utilization is 43%, only somewhat less than the 53% found in 2000. No doubt this is the result of both users adapting to their available space and hard drive manufacturers tracking the growth in data. The CDF shows a nearly linear relationship, with 50% of users having drives no more than 40% full, 70% at less than 60% utilization, and 90% at less than 80%. Proposals to take advantage of the unused capacity of file systems [2, 11] must be cautious that they only assume scaling of the magnitude of free space, not the relative portion of the disk that is free. System designers also must take care not to ignore the significant contingent (15%) of all users with disks more than 75% full.

4.3 File system Namespace

Recently, Murphy and Seltzer have questioned the merits of hierarchical file systems [22], based partly on the challenge of managing increasing data sizes. Our analysis shows many ways in which namespaces have become more complex. We have observed more files, more directories, and an increase in namespace depth. While a rigorous comparison of namespace organization structures is beyond the scope of this paper, the increase in namespace complexity does lend evidence to the argument that change is needed in file system organization. Both file and directory counts show a significant increase from previous years in Figures 6 and 7 respectively, with a mean of 225K files and 36K directories per file system.

The CDF in Figure 8 shows the number of files per directory. While the change is small, it is clear – even as users in 2009 have more files, they have fewer files per directory, with a mean of 6.25 files per directory.

Figure 9 shows the distribution of subdirectories per directory. Since the mean subdirectories per directory is necessarily one[4], the fact that the distribution is more skewed toward smaller sizes indicates that the directory structure is deeper with a smaller branching factor. However, the exact interpretation of this result warrants further study. It is not clear if this depth represents a conscious organization choice, is the result of users being unable effectively to organize their hierarchical data or is simply due to the design of the software that populates the tree. Figure 10 shows the histogram and CDF of files by directory depth for the 2009 data; similar results were not published in the earlier studies.

The histogram in Figure 11 shows how the utilization of storage is related to namespace depth. There is a steep decline in the number of bytes stored more than 5 levels deep in the tree. However, as we will see in Section 4.4, this does not mean the deeply nested files are unimportant. Comparing it with Figure 10 shows that files higher in the directory tree are larger than those deeper.

4.4 Files

Our analysis of files in the dataset shows distinct classes of files emerging. The frequently observed fact that most files are small and most bytes are in large files has intensified. The mean file size is now 318K, about three times what it was in 2000. Files can be classified by

[3] The total is 101% due to rounding error.

[4] Ignoring that the root directory isn't a member of any directory.

their update time as well. A large class of files is written only once (perhaps at install time).

4.4.1 File Size

In one respect, file sizes have not changed at all. The median file size remains 4K (a result that has been remarkably consistent since at least 1981 [27]), and the distribution of file sizes has changed very little since 2000. Figure 12 shows that the proportion of these small files has in fact increased with fewer files both somewhat larger and somewhat smaller than 4K. There is also an increase in larger files between 512K and 8MB.

Figure 13 shows a histogram of the total number of bytes stored in files of various sizes. A trend towards bi-modality has continued, as predicted in 2007 [1], though a third mode above 16G is now appearing. Figure 14 shows that more capacity usage has shifted to the larger files, even though there are still few such files in the system. This suggests that optimizing for large files will be increasingly important.

Viewed a different way, we can see that trends towards very large files being the principle consumers of storage have continued smoothly. As discussed in Section 4.5, this is a particular challenge because large files like VHDs have complex internal structures with difficult to predict access patterns. Semantic knowledge to exploit these structures, or file system interfaces that explicitly support them may be required to optimize for this class of data.

4.4.2 File Times

File modifications time stamps are usually updated when a file is written. Figure 15 shows a histogram and CDF of time since file modification with log scaling on the *x*-axis[5]. The same data with 1 month bins is plotted in Figure 16. Most files are modified between one month and a year ago, but about 20% are modified within the last month.

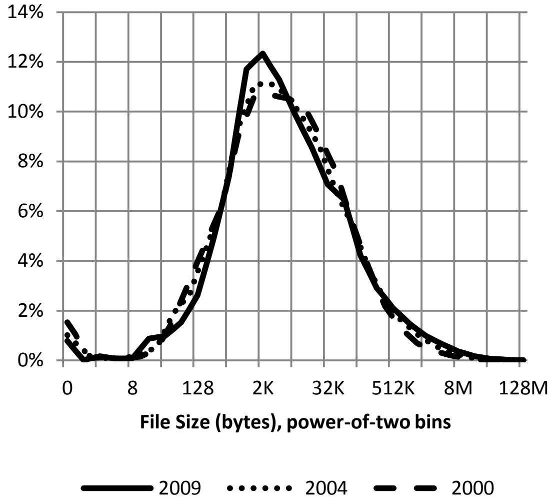

Figure 12: Histogram of Files by Size

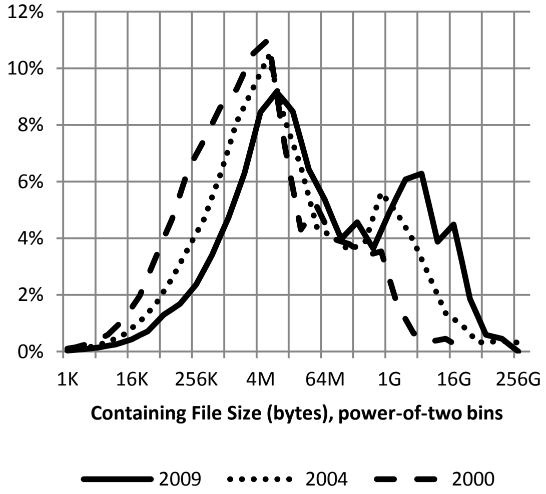

Figure 13: Histogram of Bytes by Containing File Size

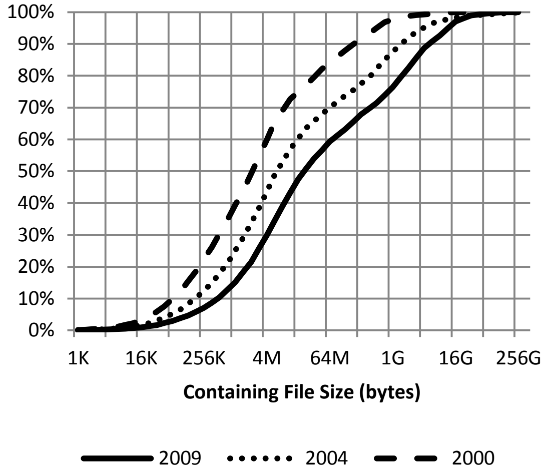

Figure 14: CDF of Bytes by Containing File Size

[5] Unlike the other combined histogram/CDF graphs, this one has both lines using the left *y*-axis due to a bug in the graphing package.

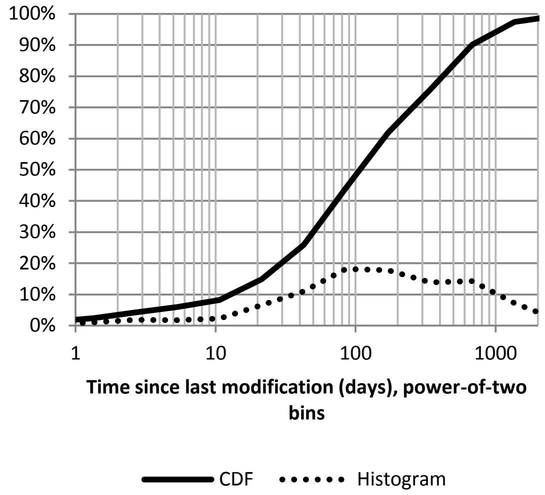

Figure 15: Time Since Last File Modification

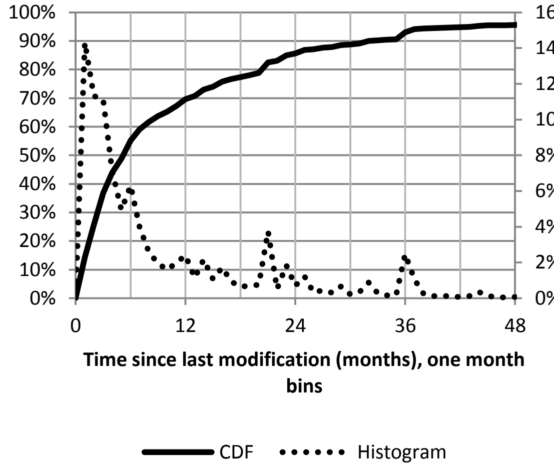

Figure 16: Time Since Last File Modification

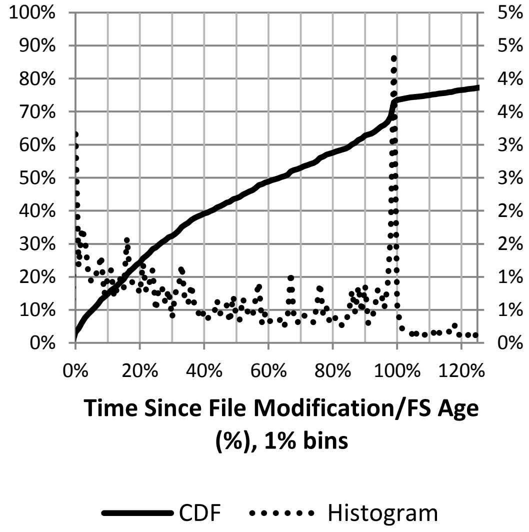

Figure 17: Time Since Last File Modification as a Fraction of File System Age.

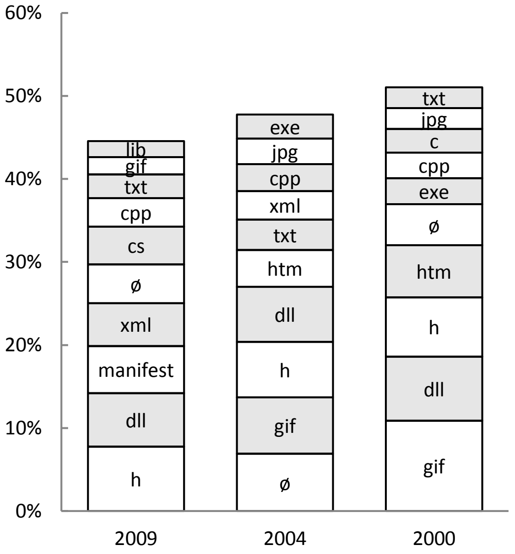

Figure 18: Popularity of Files by Extension

Figure 17 relates file modification time to the age of the file system. The *x*-axis shows the time since a file was last modified divided by the time since the file system was formatted. This range exceeds 100% because some files were created prior to installation and were subsequently copied to the file system, preserving their modification time. The spike around 100% mostly consists of files that were modified during the system installation. The area between 0% and 100% shows a relatively smooth decline, with a slight inflection around 40%.

NTFS has always supported a last access time field for files. We omit any analysis because updates to it are disabled by default as of Windows Vista [18].

4.5 Extensions

Figure 18 shows only modest change in the extensions for the most popular files. However, the extension space continues to grow. The ten most popular files extensions now account for less than 45% of the total files compared with over 50% in 2000.

Figure 19 shows the top storage consumers by file extension. Several changes are apparent here. First, there is a significant increase in storage consumed by files with no extension, which have moved from 10th place in all previous years to be the largest class of files today, replacing DLLs. VHD and ISO files are virtual disks and images for optical media. They have increased in relative size, but not as quickly as LIB files. Finally, the portion of storage space consumed by the top extensions has increased by nearly 15% from previous years.

5 On-disk Layout

The behavior and characteristics of magnetic disks continue to be a dominant concern in storage system optimization. It has been shown that file system performance changes over time, largely due to fragmentation [28]. While we have no doubt that the findings were true in 1997, our research suggests that this observation no longer holds in practice.

We measure fragmentation in our data set by recording the files' retrieval pointers, which point to NFTS's data blocks. Retrieval pointers that are non-linear indicate a fragmented file. We find such fragmentation to be rare, occurring in only 4% of files. This lack of fragmentation in Windows desktops is due to the fact that a large fraction of files are not written after they are created and due to the defragmenter, which runs weekly by default[6]. However, among files containing at least one fragment, fragments are relatively common. In fact, 25% of fragments are in files containing more than 170 fragments. The most highly fragmented files appear to be log files, which (if managed naively) may create a new fragment for each appending write.

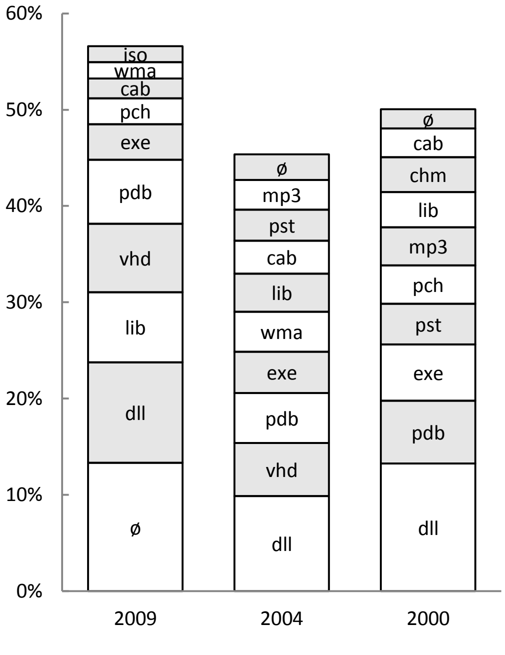

Figure 19: Bytes by File Extension

6 Related Work

Studies of live deployed system behavior and usage have long been a key component of storage systems research. Workload studies [30] are helpful in determining what file systems do in a given slice of time, but provide little guidance as to the long term contents of files or file systems. Prior file system content studies [1, 9] have considered collections of machines similar to those observed here. The most recent such study uses 7 year old data, while data from the study before it is 11 years old, which we believe justifies the file system portion of this work. However, this research also captures relevant results that the previous work does not.

Policroniades and Pratt [24] studied duplication rates using various chunking strategies on a dataset about 0.1% of the size of ours, finding little whole-file duplication and a modest difference between fixed-block and content-based chunking. Kulkarni *et al*. [12] found combining compression, eliminating duplicate identical-sized chunks and delta-encoding across multiple datasets to be effective. Their corpus was about 8GB.

We are able to track file system fragmentation and data placement, which has not been analyzed recently [28] or at large scale. We are also able to track several forms of deduplication, which is an important area of current research. Prior work has used very selective data sets usually focusing either on frequent full backups [3, 8], virtual machine images [6, 13], or simulation [10]. In the former case, data not modified between backups can be trivially deduplicated, and in the latter disk images start from a known identical storage, and diverge slowly over time. In terms of size, only the DataDomain [33] study rivals ours. It is less than half the size presented here and was for a highly self-selective group. Thus, we not only consider a more general, but also a larger dataset than comparable studies. Moreover, we include a comparison to whole-file deduplication, which has been missing in much of the deduplication research to date. Whole file deduplication is an obvious alternative to block-based deduplication because it is light-weight and as we have shown, nearly as effective at reclaiming space.

[6] This is true for all of our scans other than the 17 that came from machines running Windows Server 2003.

7 Conclusion

We studied file system data, metadata, and layout on nearly one thousand Windows file systems in a commercial environment. This new dataset contains metadata records of interest to file system designers, data content findings that will help create space efficiency techniques, and data layout information useful in the evaluation and optimization of storage systems.

We find that whole-file deduplication together with sparseness is a highly efficient means of lowering storage consumption, even in a backup scenario. It approaches the effectiveness of conventional deduplication at a much lower cost in performance and complexity. The environment we studied, despite being homogeneous, shows a large diversity in file system and file sizes. These challenges, the increase in unstructured files, and an ever-deepening and more populated namespace pose significant challenge for future file system designs. However, at least one problem – that of file fragmentation, appears to be solved, provided that a machine has periods of inactivity in which defragmentation can be run.

Acknowledgements

We would like to thank the hundreds of Microsoft employees who were willing to allow us to install software that read the entire contents of their disks, Richard Draves for helping us with the Microsoft bureaucracy, Microsoft as whole for funding and enabling this kind of research, our program committee shepherd Keith Smith and the anonymous reviewers for their guidance as well as detailed and helpful comments, and Fred Douglis for some truly last-minute comments and proof-reading.

References

[1] N. Agrawal, W. Bolosky, J. Douceur and J. Lorch. A five-year study of file-system metadata. In *Proc. 5th USENIX Conference on File and Storage Technologies*, 2007.

[2] M. Bhadkamkar, J. Guerra, L. Useche, S. Burnett, J. Liptak, R. Rangaswami, and V. Hristidis. Borg: block-reorganization for self-optimizing storage systems. In *Proc. 7th USENIX Conference on File and Storage Technologies*, 2009.

[3] D. Bhagwat, K. Eshghi, D. Long, and M. Lillibridge, Extreme binning: scalable, parallel deduplication for chunk-based file backup, In *Proc. 17th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2009.

[4] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM* 13(7): 422—426, 1970.

[5] W. Bolosky, S. Corbin, D. Goebel and J. Douceur. Single instance storage in Windows 2000. In *Proc. 4th USENIX Windows Systems Symposium*, 2000.

[6] A. Clements, I. Ahmad, M. Vilayannur, J. Li. Decentralized deduplication in SAN cluster file systems. In *Proc. USENIX Annual Technical Conference*, 2009.

[7] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane. Tradeoffs in scalable data routing for deduplication clusters. In *Proc. 9th USENIX Conference on File and Storage Technology*, 2011.

[8] S. Dorward and S. Quinlan. Venti: A new approach to archival data storage. In *Proc. 1st USENIX Conference on File and Storage Technologies*, 2002.

[9] J. Douceur and W. Bolosky. A large-scale study of file-system contents. In *Proc. 1999 ACM SIGMETRICS International Conference on Measurement and Modelling of Computer Systems,* 1999.

[10] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki. Hydrastor: a scalable secondary storage. In *Proc. 7th USENIX Conference on File and Storage Technologies*, 2009.

[11] H. Huang, W. Hung, and K. G. Shin. Fs2: dynamic data replication in free disk space for improving disk performance and energy consumption. In *Proc. 20th ACM Symposium on Operating Systems Principles*, 2005.

[12] P. Kulkarni, F. Douglis, J. LaVoie, and J. Tracey. Redundancy elimination within large collections of files. In *Proc. USENIX 2004 Annual Technical Conference*, 2004.

[13] K. Jin and E. Miller. The effectiveness of deduplication on virtual machine disk images. In *Proc. SYSTOR 2009: The Israeli Experimental Systems Conference*, 2009.

[14] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble. Sparse indexing: large scale, inline deduplication using sampling and locality. In *Proc. 7th USENIX Conference on File and Storage Technologies*, 2009.

[15] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier. The new ext4 filesystem: current status and future plans. In *Proc. of the Linux Symposium*, June, 2007.

[16] Microsoft Corporation. BackupRead Function. MSDN. [Online] 2010. [Cited: August 17, 2010.] http://msdn.microsoft.com/en-us/library/aa362509(VS.85).aspx.

[17] Microsoft Corporation**.** Description of the scheduled tasks in Widows Vista. *Microsoft Support.* [Online] July 8, 2010. [Cited: August 9, 2010.] http://support.microsoft.com/kb/939039.

[18] Microsoft Corporation. Disabling Last Access Time in Windows Vista to Improve NTFS Perfomance. The Storage Team Blog. [Online] 2006. [Cited November 2, 2010.] http://blogs.technet.com/b/filecab/archive/2006/11/07/disabling-last-access-time-in-windows-vista-to-improve-ntfs-performance.aspx.

[19] Microsoft Corporation. File systems. Microsoft TechNet. [Online] 2010. [Cited: August 9, 2010.] http://technet.microsoft.com/en-us/library/cc938929.aspx.

[20] Microsoft Corporation. Volume Shadow Copy Service. MSDN. [Online] 2010. [Cited August 31, 2010.] http://msdn.microsoft.com/en-us/library/bb968832(VS.85).aspx

[21] D. R. Miller. Storage Economics: Four Principles for Reducing Total Cost of Ownership. *Hitachi Corporate Web Site.* [Online] May 2009. [Cited: August 17, 2010.] http://www.hds.com/assets/pdf/four-principles-for-reducing-total-cost-of-ownership.pdf.

[22] N. Murphy and M. Seltzer. Hierarchical file systems are dead. In *Proc. 12th Workshop on Hot Topics in Operating Systems*, 2009.

[23] R. Nagar. *Windows NT File System Internals*. O'Reilly, 1997

[24] C. Policroniades and I. Pratt. Alternatives for detecting redundancy in storage systems. In *Proc. USENIX 2004 Annual Technical Conference*, 2004.

[25] M. Rabin. Fingerprinting by Random Polynomials. Harvard University Center for Research In Computing Technology Technical Report TR-CSE-03-01, 1981. Boston, MA.

[26] R. Rivest. *The MD5 Message-Digest Algorithm*. [Online] April 1992. [Cited: August 17, 2010.] http://tools.ietf.org/rfc/rfc1321.txt.

[27] Satyanarayanan, M. A study of file sizes and functional lifetimes. In *Proc. 8th ACM Symposium on Operating Systems Principles*, 1981.

[28] M. Seltzer and K. Smith. File system aging: increasing the relevance of file system benchmarks. In *Proc. 1997 ACM SIGMETRICS,* June 1997.

[29] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck. Scalability in the XFS file system. In *Proc. 1996 USENIX Annual Technical Conference*, 1996.

[30] W. Vogels. File system usage in windows NT 4.0. In *Proc. 17th ACM Symposium on Operating Systems Principles*, 1999.

[31] C. Ungureanu, B. Atkin, A. Aranya, S. Gokhale, S. Rago, G. Cakowski, C. Dubnicki, and A. Bohra. Hydrafs: A high-throughput file system for the Hydrastor content-addressable storage system. In *Proc. 8th USENIX Conference on File and Storage Technologies*, 2010.

[32] E. Ungureanu and C. Kruus. Bimodal content defined chunking for backup streams, In *Proc. 8th USENIX Conference on File and Storage Technologies*, 2010.

[33] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the Data Domain deduplication file system. In *Proc. 6th USENIX Conference on File and Storage Technologies*, 2008, pp. 1-14.