Analysis of Disk Access Patterns on File Systems for Content Addressable Storage

Kuniyasu Suzaki, Kengo Iijima, Toshiki Yagi, and Cyrille Artho
National Institute of Advanced Industrial Science and Technology
{ k.suzaki | k-iijima | yagi-toshiki | c.artho } @aist.go.jp

Abstract

CAS (Content Addressable Storage) is virtual disk with deduplication, which merges same-content chunks and reduces the consumption of physical storage. The performance of CAS depends on the allocation strategy of the individual file system and its access patterns (size, frequency, and locality of reference) since the effect of merging depends on the size of a chunk (access unit) used in deduplication.

We propose a method to evaluate the affinity between file system and CAS, which compares the degree of deduplication by storing many same-contents files throughout a file system. The results show the affinity and semantic gap between the file systems (ext3, ext4, XFS, JFS, ReiserFS (they are bootable file systems), NILFS, btrfs, FAT32 and NTFS, and CAS.

We also measured disk accesses through five bootable file systems at installation (Ubuntu 10.10) and at boot time, and found a variety of access patterns, even if same contents were installed. The results indicate that the five file systems allocate data scattered from a macroscopic view, but keep block contiguity for data from a microscopic view.

1 Introduction

Content Addressable Storage (CAS) is becoming a popular method to manage virtual disks for many instances of virtual machines [2, 6]. In CAS systems, data is managed in chunks, and it is addressed not by its physical location but by a name derived from the content of that data (usually a secure hash is used as a unique name). A CAS system can reduce the use of physical disk space by deduplication, which merges same-content chunks with a unique name.

CAS provides a universal virtual block device and accepts any file system on it. The performance depends on data allocations and their access patterns through the file system, because each file system has techniques to optimize space usage and I/O performance. The optimizations include data alignment, contiguous allocation, disk prefetching, lazy evaluation, and so on. These factors make the file system a key factor for the performance of CAS.

From the view of the disk, a file system works as a "filter" to allocate data. Even if the same contents are saved, access patterns differ between file systems. Especially Linux has many file systems, because Linux supports a wide variety of targets, from mobile devices to super computers. In this paper, we propose a method to evaluate the affinity between file system and CAS. The method evaluates the effect of deduplication when many same-content files are stored throughout a file system.

We also analyze the real behavior of bootable file systems on CAS. We measure access patterns at installation (write-centric processing) and boot time (read-centric processing). From the results, we investigate the affinity between file system and CAS behavior.

This paper is organized as follows. Section 2 reviews features of CAS systems and Section 3 describes features of Linux file systems. Section 4 proposes the method to measure the affinity between file system and CAS. Section 5 report the results, the affinity and real behavior at installation and boot time. Section 6 discusses future works. Section 7 summarizes our conclusions.

2 Content Addressable Storage

This section describes features of CAS (Content Addressable Storage) systems. A pioneering CAS system

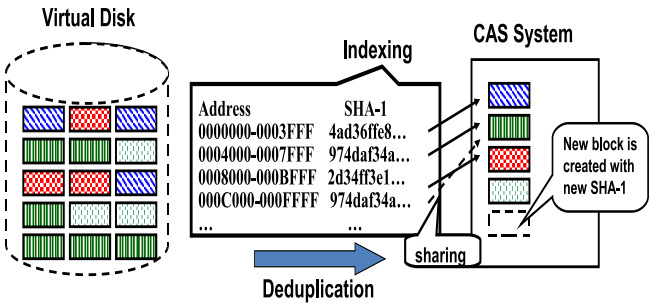

Figure 1: Virtual disk managed by CAS system.

is Venti developed for the Plan9 data archive [9]. Venti is a block storage system in which chunks are identified by a collision-resistant cryptographic hash.

Figure 1 shows a virtual disk managed by CAS. The virtual disk is divided by chunk for each region. Each chunk is named by its hash-value, and stored in a database of the CAS system. The chunks are managed by the mapping table, which translates from address to hash-value. If the contents of chunks are the same, the hash-value is same and the CAS system can reduce its storage space consumption. A chunk is also self-verifying with its hash digest and can keep data integrity.

In this paper we use LBCAS (LoopBack Contents Addressable CAS) version 2 [11, 12]. LBCAS offers a loopback block file (virtual disk) which is managed by FUSE (Filesystem in Userspace) [1]. Chunk data is created when a write access is issued to the virtual disk. The chunks are stored in a Berkeley DB [8] and managed by their SHA-1 hash-value. The size of a chunk is defined by the configuration (32 KB – 512 KB). The driver of LBCAS has a memory cache for 32 chunks. When a chunk overflows from the cache, the data is written to the Berkeley DB.

3 File Systems

Many file systems are developed for Linux, and each of them has its own advantages. In this paper, we treat 9 file systems: ext3, ext4, XFS, JFS, ReiserFS, NILFS, btrfs, FAT32 and NTFS. Unfortunately, not all of them can be used a root file system, because boot loader and installer have to recognize them. We used five file systems (ext3, ext4, XFS, JFS, ReiserFS) to investigate the behavior at installation and boot time.

3.1 Linux File Systems

This section describes the features of 9 file systems used in this paper.

Ext3 is the default file system on many Linux distributions. It extends ext2 with journaling. ext3 keeps compatibility with ext2, including some limitations, such as no extent allocation, no dynamic allocation of i-nodes, etc.

Ext4 [7] succeeds ext3 and extends it with *extent allocation* and *delayed allocation*. Extent allocation keeps contiguous physical blocks for a file and reduces fragmentation. Delayed allocation is a technique to reduce file fragmentation, which is also used by XFS.

JFS is a 64-bit journaling file system originally created by IBM for AIX. JFS uses a B+ tree to accelerate lookups in directories. JFS dynamically allocates space for i-nodes as necessary. JFS increases disk I/O performance by using *allocation groups* and extent allocation. An allocation group is a sub-volume in a file system that keeps track of free blocks and file data on its own. JFS contains effective methods to use allocation groups.

XFS is a high-performance journaling file system originally created by Silicon Graphics. XFS increases disk I/O performance by using allocation groups, extent allocation, delayed allocation, and *variable block size*. Variable block size allows XFS to be created with block sizes ranging between 512 B and 64 KB, increasing I/O bandwidth when large files are created. Delayed allocation makes it possible to allocate a contiguous group of blocks, reducing fragmentation.

ReiserFS (version 3) is the first journaling file system to be included in the standard Linux kernel. ReiserFS has the *tail packing* optimization which allocates last partial blocks of multiple files into a single block. The technique can reduce the internal fragmentation of files.

NILFS [4] is a stackable file system which is also called log-structured file system. NILFS allocates data in succession from the top of a disk. The sequential write achieves high I/O throughput on a real block device. The data on the log-structured format are only appended and never overwritten. In particular, a previous version of a file can be retrieved in the file system.

Btrfs is a new file system which features *copy-on-write*. Copy-on-write is used for creating a snapshot and for

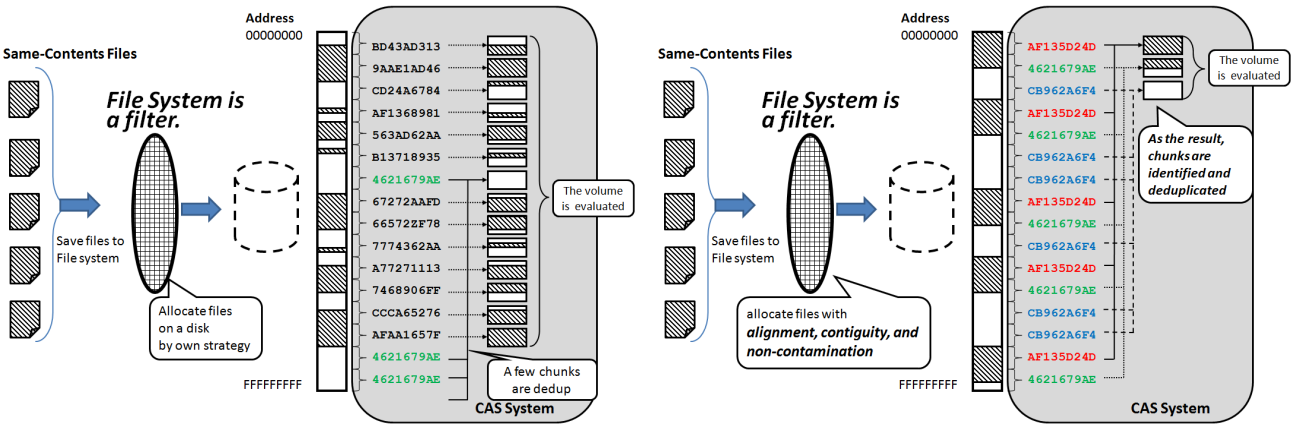

Figure 2: Affinity evaluation between file system and CAS. The left figure shows bad allocation and the right figure shows good allocation for CAS.

cloning. Btrfs has many new features which includes extent allocation implemented on ext4.

FAT32 and NTFS are file systems for Windows. They are not UNIX-style file systems and do not use i-nodes. The data is managed by a unit called "cluster", which is a contiguous groups of hardware sectors. The original development of FAT32 comes from floppy disks and has simple structure. FAT32 manages the clusters with an array, and does not perform well on large disks. NTFS developed for WindowsNT and has several improvements over FAT, which includes extent allocation. NTFS manage a file with the Master File Table (MFT) containing meta-data about every file and directory. The details of NTFS are not open, and the drivers for Linux are developed in many ways. Currently most Linux distributions use the NTFS-3G driver.

3.2 Bootable File System

The boot loader has to recognize a file system in order to load the kernel. The currently popular boot loader *GRUB* recognizes some file systems. In order to analyze the behavior at installation and boot time, we select five popular file systems (ext3, ext4, XFS, JFS, and ReiserFS version3) recognized by GRUB.

3.3 System Installation on a File System

Even if the same applications are installed on a file system, the installer of a Linux distribution recognizes the target file system, and customizes some files for it. For example, the initial ram disk image "initrd" is customized for a file system. A kernel and initrd are loaded by GRUB from a file system at boot time, and the kernel uses the configuration files to mount a file system on a disk as the root file system.

The root file system has to include additional files to maintain itself. Some of them are management tools for the file system. Furthermore, each file system has special features. For example ext3 and 4 file systems have a `lost+found` directory to retrieve lost files, which other file systems lack. However, the differences are small and they are negligible for installation and booting.

4 Affinity between File System and CAS

File systems allocate data on a disk; in doing so, the act as a filter. Each filter changes the location of data by its own strategy. Depending on the location, the effect of deduplication changes. We evaluate the difference from the view of deduplication.

We develop a method to evaluate the affinity between file system and CAS. The idea is simple. Ideally, even if many same-content files are saved on CAS, the total disk usage of CAS will be close to the size of one file, because all files are deduplicated. *Namely, the closer the total disk usage is to the size of one file, the better the allocation strategy for CAS.*

For example, when 1,000 files with 1MB same-content data are stored on a disk through a normal file system, it will use 1,000 MB. However, if deduplication of CAS works perfectly, the increase will amount to only 1MB.

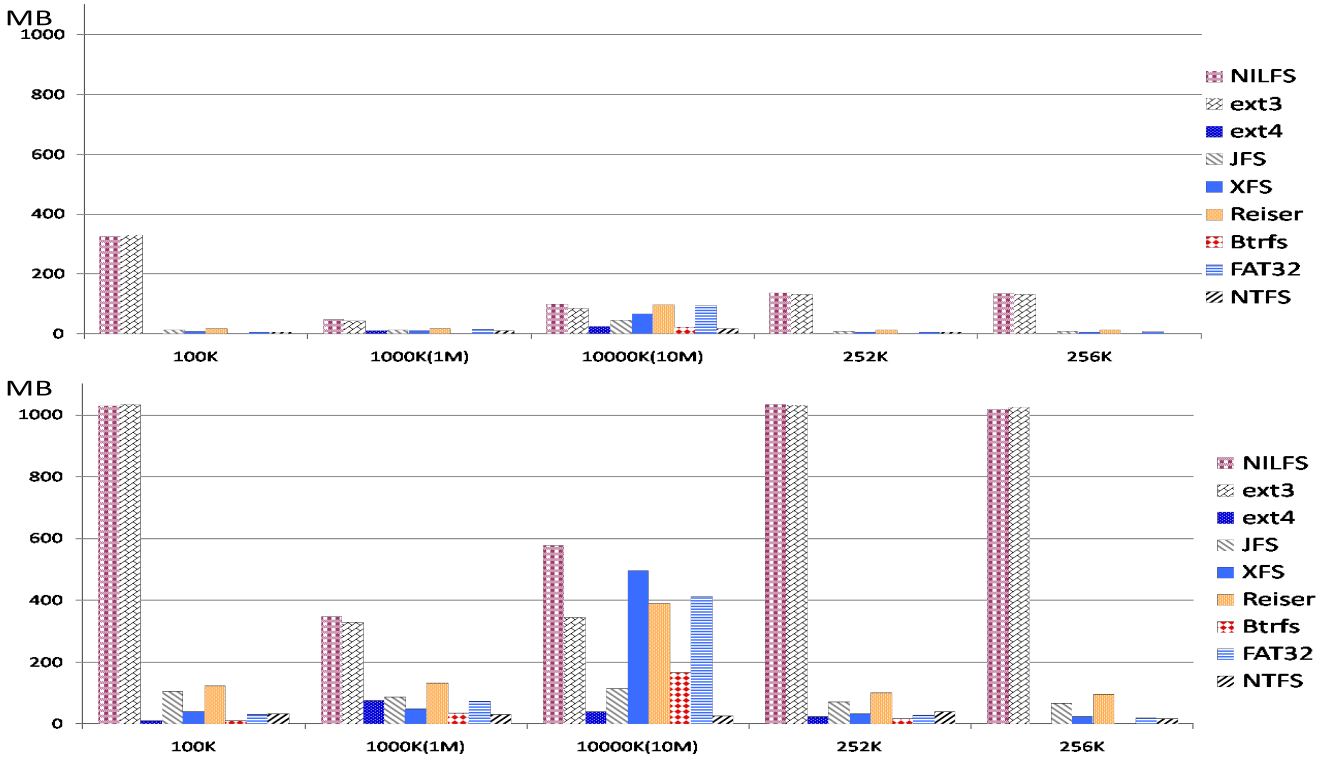

Figure 3: The increase of CAS space when 10,000 100 KB-files, 1,000 1,000 KB-files, 100 10,000 KB-files, 3,968 252 KB-files and 3,906 256 KB-files (which consume nominal 1GB) are stored on each file system (ext3, ext4, XFS, JFS, ReiserFS, NILFS, btrfs, FAT32 and NTFS). The upper figure shows the results on 32 KB LBCAS and the lower figure shows the results on 256 KB LBCAS. A small increase indicates good deduplication.

Figure 2 shows the image of the evaluation. The right of Figure 2 shows poor allocation by a file system. At that time, the locations of data are scattered and will not deduplicated, except the non-used (zero-cleared by initialization) region. The left of Figure 2 shows good allocation. The whole data is deduplicated and the consumption of CAS is close to the size of the file.

The affinity comes from (1) Alignment matching, (2) Contiguous allocation of data blocks, and (3) Non-contamination with other data in a CAS chunk. If the allocation strategy of a file system aligns data of a file at the alignment of a CAS chunk, it increases the chance of the file being deduplicated. Contiguous allocation of data blocks is also important to fill a chunk with same-content data. Non-contamination comes into play when a chunk is not entirely filled up with contiguous allocation of data blocks. At that time, the remainder of a chunk should not be filled with other data. However, some techniques pack data into a small empty region and reduce the chance of data to be deduplicated. For example, tail packing will contaminate a chunk.

The evaluation measures the affinity by determining the total volume used by CAS. Unfortunately, this methodology is still simple, because it does not care of the volume management mechanism (for example, bitmap table to manage free space) and meta-data which is used to identify the locations of contents with file names. A volume management mechanism is fixed-size and the update may be small and negligible. However, meta-data is created for each file and the volume consumption of meta-data is non-negligible when a file is small. On a real evaluation we must care about the use of meta-data.

5 Affinity and Performance Evaluation

This section evaluates the affinity and performance between file system and CAS. The affinity of deduplication is described in section 5.1, and the real performance of CAS on a file system is described in section 5.2.

5.1 Affinity evaluation between File System and CAS

We measure the effect of deduplication, when many same-content files are saved. We used random data as contents, because they are not deduplicated with other files. We tried to save 100 KB, 1,000 KB (1 MB), 10,000 KB (10 MB) random data files to fill 1 GB in 4 GB LBCAS system. Namely, 10,000 files for 100 KB, 1,000 files for 1 MB, 100 files for 10 MB were used.

The evaluation has to consider meta-data for each file. We assume that a meta-data consists of 256 bytes for a file. This means that the 10,000 files consume 256 KB on disk for meta-data. The consumption of meta-data is non-negligible when the deduplicated file is small. For example, at the 100 KB case, 256 KB is used for meta-data (256 B * 10,000) and the ideal consumption of CAS is 356 KB (100 KB + 256 KB). We have to care the increase. However, in the 1,000 KB case, the total is 1,025.6 KB and at 10,000 KB case, the total is 10,002.56 KB.

We also tried to save 256 KB and 252 KB random data files to check the suitable size for CAS deduplication. If file system allocates the files in succession, 256 KB (64 4 KB-file-system-blocks) files will fit to 256 KB and 32 KB chunks many times. We assume a stackable file system corresponds to this case. When 252 KB (63 4 KB-file-system-blocks) files are saved, 256 KB chunks will each fit to 64 allocations. If a block (4KB) is used between 2 contiguous files for meta-data or something, 252 KB data will also fit to 256 KB.

5.1.1 Experimental results

Figure 3 shows the increase of CAS when the files are stored on 32 KB LBCAS and 256 KB LBCAS. The results are the average of three trials. They show the different effect of deduplication on each file system. When the chunk size is larger than the size of test file, a file does not fill a chunk, even if a file is allocated continuously. At that time, a chunk is subject to contamination by other data. For example, a 256 KB chunk is not filled up with 100 KB, and 252 KB files. 256 KB file can fill up a 256 KB chunk, but there is no big difference between the 252 KB and 256 KB cases on any file systems.

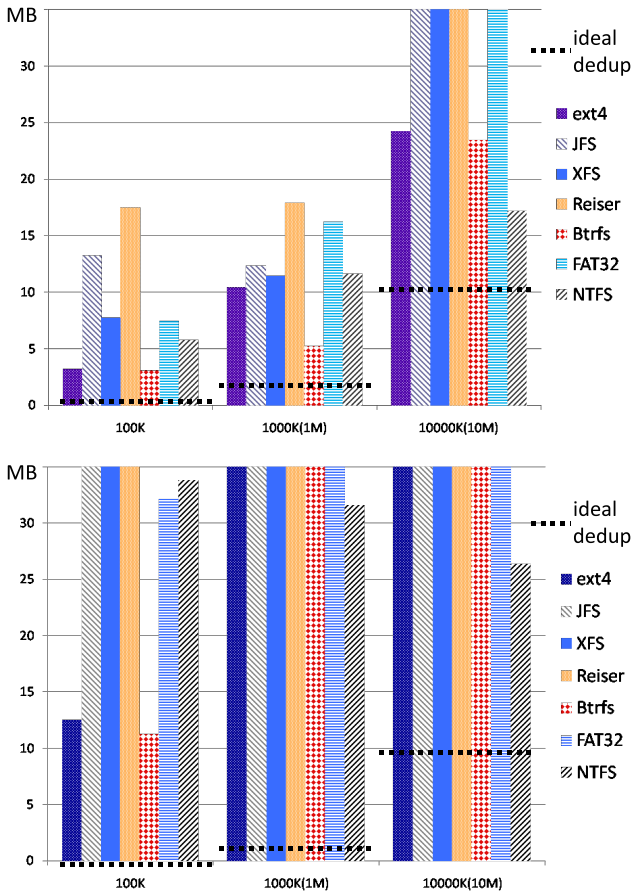

Figure 4: Details on the increase of CAS described in Figure 3. The maximum range is 35 MB. The figure includes line which indicates ideal deduplication. The upper figure shows the results on 32 KB LBCAS and the lower figure shows the results on 256 KB LBCAS. NTFS and ext3 are eliminated because they are out of range. The results of 252 KB and 256 KB files are also eliminated because we could not get alignment matching cased by contiguous allocation of data blocks.

Ext3 and NILFS show the worst results on 100 KB, 252 KB, and 256 KB files on 256 KB CAS. Their total space usage reaches 1 GB, which means no deduplication. The results become better on 32 KB CAS, but they are still worse than on the other file systems. The results of ext3 and NILFS on 100 KB files are about 2.5 times worse than for 252 KB and 256 KB on 32 KB CAS. We guess the results come from the ease of contamination when the chunk size is larger than a file. If a file is allocated at a contiguous region, such as a stackable file system, the chance to being contaminated is inversely proportional to the size of file.

XFS and FAT32 show good results except for the 10 MB file case on 256 KB CAS. We guess the data in a large file is not allocated contiguously, which hampers deduplication.

ReiserFS shows bad results on 256 KB CAS. We guess it comes from tail packing, which contaminates a chunk. 256 KB chunks are large and the penalty of not to being deduplicated has a big impact. ReiserFS also shows bad results on the 10 MB file case on 256 KB CAS. This is also caused by non-contiguous allocation for large files.

Figure 4 shows details of Figure 3, which limits the maximum (35 MB), eliminates two file systems (ext3 and NILFS) and 2 trials (252 KB and 256 KB files), and includes a line which indicates ideal deduplication. The results indicate ext4, btrfs, and NTFS on 32 KB CAS are close to the ideal case on any file size. They keep three features; alignment matching, contiguous allocation of data blocks, and non-contamination with other data. On 256 KB CAS, ext4 and btrfs show bad results on 10,000 KB and 100,000 KB files, although they show good results on 100 KB files. We guess ext4 and btrfs cannot keep contiguous allocation of data blocks, and suffer from contamination with other data on larger files.

5.1.2 Impact by chunk size

Figure 5 shows the ratio of consumption between 32 KB and 256 KB CAS. The ratio indicates the improvement by the smaller 32 KB chunk size. In the ideal case the ratio is 8 times. It means 256 KB chunks are not aligned or include slightly different data. For example, JFS in the 100 KB file case reduces the space consumption by 8 times on 32 KB CAS compared to 256 KB CAS. From another view, small ratios indicate that there is no improvement for small chunks when compared to

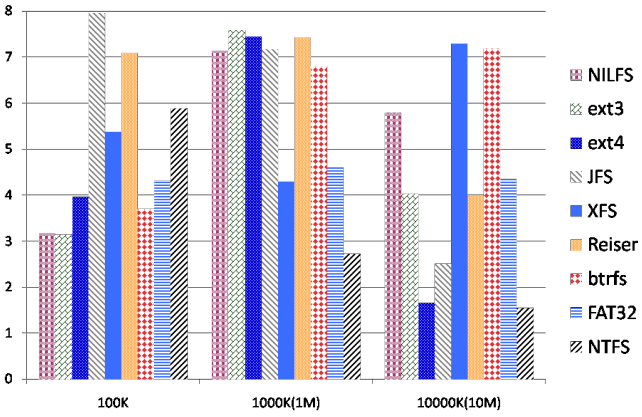

Figure 5: The ratio of space consumption on each file system. It shows the improvement of 32 KB CAS from 256 KB CAS.

larger 256 KB chunks. At that time, the user should use 256 KB chunks because the mapping table of 256 KB CAS is 8 times smaller than 32 KB CAS. For example, ext4 and NTFS show that there is a little impact on 10 MB file case.

5.1.3 Future work

The experiments were tried on an initial disk image which does not have fragmentation. We eliminate such evaluations, because of it is not clear which metrics to use to compare fragmented file systems, and we cannot decide factors for deduplication. We recognize these conditions to be important in real cases. Evaluations under these conditions are our next challenge.

5.2 CAS performance at installation and at boot time

We evaluated LBCAS performance at Linux installation and at boot time. We installed Ubuntu 11.04 desktop (Linux 2.6.38) on five file systems (ext3, ext4, XFS, JFS, and ReiserFS) of 4 GB LBCAS on KVM [3] virtual machine with 768 MB memory. KVM ran on a ThinkPAD T400 with an Intel Core2 Duo processor with 2 GB of memory. We compared the effect of 32 KB chunk and 256 KB chunk of LBCAS.

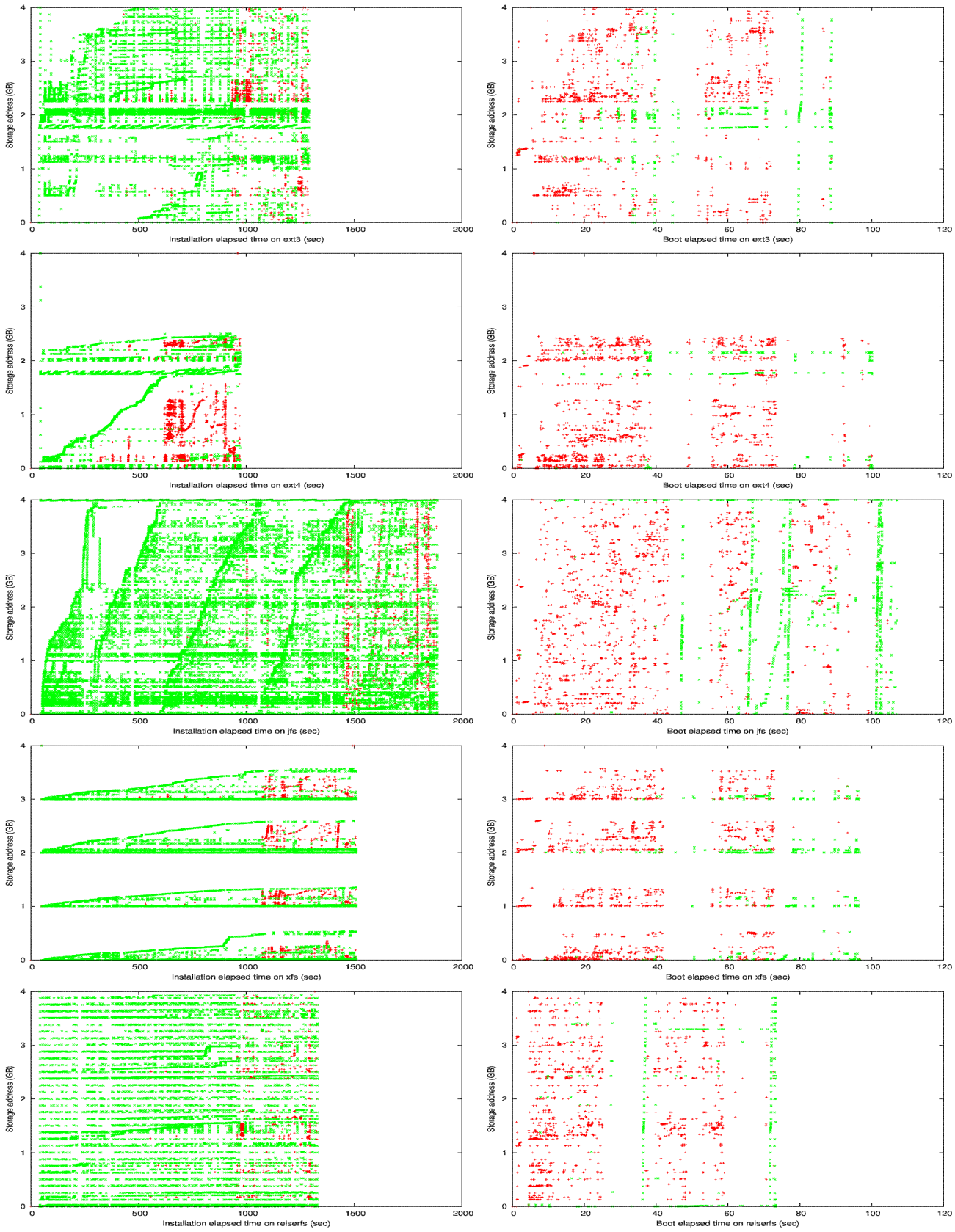

Figure 6: Access pattern at installation (left) and boot time (right) of Ubuntu on each file system. The X axis indicates elapsed time and Y indicates the address of the disk access (4GB). The green "X" plots indicate write accesses, and red "+" plots indicate read accesses. The file systems are ext3, ext4, JFS, XFS, and ReiserFS from top to bottom.

5.2.1 Access Trace at installation and boot time

Figure 6 shows the trace of access on each file system at installation and boot time. The graphs indicate that access patterns are different on each file system.

The left graphs show the installations. Most accesses are write operations. The installation includes creating the file system on CAS. The creation of a file system is started after 50 sec. Before 50 sec, the installation requires some preparations in memory.

The graphs show that the accesses of the five file systems are scattered in the 4 GB disk space. This property remains the same even if the disk size is changed to 2 GB or 8 GB. The results indicate that the five file systems have different allocation strategies and allocate data scattered from the macroscopic view of disk fragmentation.

The right graphs show booting. Booting is a read-centric process, but there are write operations at the end of boot, because several configuration files are updated. The access distributions follow the written data at installation, and there are no localities of reference from the macroscopic view.

5.2.2 Installation Time (Dynamic Feature)

Table 1 shows statistics of read and write accesses on each file system at installation time. Installation is a write-centric process and writes about 10 times as much data as it reads.

The upper three rows show the accesses issued by file system, access times, total volume of accesses, and the average. The results show the ability of a file system. Lower access times and fewer accesses are better. Ext4 shows the fewest access times and largest average access on write operations. This result may be the effect of delayed allocation, and it yields the fastest installation time (Figure 6). JFS shows the smallest total volume on write operations, but installation time is the worst, because the number of accesses is large. A CAS system is sensitive to access times, because the access unit is the chunk size even if an access is only 1 bit. The boot times of five file systems are almost proportional to the write access times, and do not follow the total volumes. Fewer accesses are better on a CAS system.

The lower four rows in Table 1 show the number of read and write chunks of 32 KB and 256 KB size. Fewer chunks are better. ReiserFS shows the best performance on read and write, respectively, on both 32 KB and 256 KB chunks. This indicates ReiserFS is good at locality of access, but accesses the same chunks many times.

5.2.3 Disk Image of LBCAS (Static Feature)

Table 2 shows statistics of a static LBCAS disk image with 32 KB and 256 KB chunks.

ReiserFS shows the fewest chunks, the smallest total volume used by chunks, and the largest volume of zero-cleared chunks on 32 KB and 256 KB chunks. The results of the number of chunks and total volume is about 10% less than other file systems, which might come from tail packing to reduce disk consumption. This is a good feature for a normal disk but it ruins the effect of deduplication mentioned in Section 5.1.1.

The efficiency, which indicates the ratio of effective data in a chunk, is more than 99% on all file systems using 32 KB chunks, although Figure 6 shows that data accesses look to be scattered. It shows that the allocation strategies of the file systems pack data in small region. However, the efficiencies of ext4 and XFS decrease to less than 94% when using 256 KB chunks. These results indicate that ext4 and XFS allocate data discretely for larger units. This feature suggests a suitable chunk size of a file system.

On deduplication, ext4 is the best on 32 KB and 256 KB chunks, which indicates that many same-content chunks are created. We guess the effect comes from alignment matching, contiguous allocation of data blocks, and non-contamination. This effect is predicted by the results mentioned in Section 5.1.1 Figure 4. Currently btrfs and NTFS are not bootable file systems and they are out of scope on current experiments.

We compared the ratio of same chunks between 2 CAS images which are installed same OS. Figure 7 shows the image of comparison. The ratios are measured on two types. One type is the ratio between 2 CAS images of different file systems. The results indicate the affinity between file systems from the view of CAS. It helps the understanding the effect of mixture of CAS images which has different file systems.

File system	ext3		ext4		JFS		XFS		ReiserFS	
	read	write	read	write	read	write	read	write	read	write
Access times	13,618	182,971	14,945	**164,124**	13,409	247,981	11,650	265,981	**10,739**	186,432
Total volume(MB)	385.6	3,808.2	413.3	3,745.1	393.3	**3,194.8**	385.4	4,740.6	**327.0**	3,946.4
Average (KB)	29.0	21.3	28.3	**23.4**	30.0	13.2	**33.9**	18.3	31.2	21.7
32KB LBCAS										
Number of chunks	13,152	68,759	14,978	70,401	14,209	65,141	15,260	66,337	**11,784**	**57,087**
Total volume (MB)	411.0	2,148.7	468.1	2,200.0	444.0	2,035.7	476.9	2,073.0	**368.3**	**1,784.0**
256KB LBCAS										
Number of chunks	2,314	8,684	2,712	9,222	2,687	8,207	2,839	8,499	**2,066**	**7,170**
Total volume (MB)	578.5	2,171	678.0	2,305.5	671.8	2,051.8	709.8	2,124.8	**516.5**	**1,792.5**

Table 1: Statistics of read/write accesses on each file system at installation time (dynamic feature). The bold figures indicate the best performance.

File system	ext3	ext4	JFS	XFS	ReiserFS
32 KB LBCAS					
Number of chunks	67,157	67,819	64,770	65,415	**56,671**
Total volume (MB)	2,148.1	2,192.3	2,035.0	2,059.2	**1,783.7**
Zero-cleared chunk (MB)	1,947.9	1,903.7	2,061.0	2,036.8	**2,312.3**
Effectiveness (%)	99.88	99.93	99.90	**99.99**	99.94
Dedupication (Total MB / Unique MB)	49.47/8.75	**73/19.16**	10.94/3.75	15.03/7.06	12.69/5.44
256 KB LBCAS					
Number of chunks	8,554	9,020	8,190	8,475	**7,156**
Total volume (MB)	2,169.8	2304.0	2,050.5	2,124.0	**1,792.0**
Zero-cleared chunk (MB)	1,926.3	1,792.0	2,045.5	1,972.0	**2,304.0**
Effectiveness (%)	98.40	93.13	99.12	92.10	**99.47**
Dedupication (Total MB / Unique MB)	31.25/2.0	**49.0/8.25**	3.0/1.75	5.25/1.75	3.0/0.5

Table 2: Statistics of a virtual disk for each file system (static feature). The row of effectiveness shows the ratio of blocks of file system in a chunk. It shows coverage of effective region in a chunk. The row of deduplication shows two data; total and unique. The total indicates the summation of same-content chunks. The unique indicates the summation of merged chunks with deduplication. The bold figures indicate the best performance.

File system	ext3		ext4		JFS		XFS		ReiserFS	
	read	write	read	write	read	write	read	write	read	write
Access times	6,115	3,653	5,663	3,458	6,260	2,894	6,199	4,383	**5,195**	**2,625**
Total volume (MB)	209.7	39.7	228.0	40.7	198.1	**17.2**	216.8	22.7	**187.7**	27.2
Average (KB)	35.1	11.1	41.2	**12.1**	32.4	6.1	35.8	5.31	**37.0**	10.6
32 KB LBCAS										
Number of chunks	8,065	1,491	8,548	1,522	8,130	1,204	8,507	**1,094**	**7,402**	1,295
Total volume (MB)	252.0	46.6	267.1	47.6	254.1	37.6	265.8	**34.2**	**231.3**	40.5
256KB LBCAS										
Number of chunks	1,508	292	1,624	**247**	1,941	712	1,767	372	**1,500**	367
Total volume (MB)	377.0	73.0	406.0	**61.8**	485.3	178	441.7	93.0	**375.0**	91.8

Table 3: Statistics of read/write accesses on each file system at boot time (dynamic feature). The bold figures indicate the best performance.

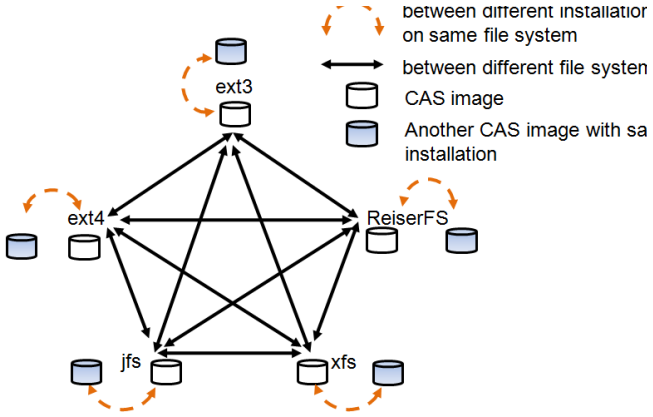

Figure 7: Compare ratio of same chunks between 2 CAS images which are installed same OS. The ratios are measured on 2 types. One type is the ratio between 2 CAS images of different file systems. Another type is the ratio between 2 CAS images which install same OS on same file system at different time.

Another type of measurement is the ratio between 2 CAS images which install same OS on same file system at different time. The results indicate the suitable file systems which reduce consumption of physical resources, when some users install same OS on the CAS system with same file system.

Figure 8 shows the results. We measure the ratio on different chunk sizes from 4KB to 256KB, in order to know the effect. The upper figure shows the ratio between different file systems, which are illustrated in Figure 7 with solid lines. The results indicate that there are small differences on any combination. It means there are not strong affinities among the 5 file systems. The ratios of same chunks depend on chunk size. On 4KB chunk size, the ratio is very high from 80% to 90%, because chunk size matches the block size of most file systems, and most data blocks having the same contents will be same. The most difference comes from meta-data and file system management data, except ReiserFS which has tail packing. Tail packing reduced the consumption of storage 10% more than other file systems in Table 2. Unfortunately, it was known to cause negative impact on deduplication, because it contaminates a block for a file. The effect was measured in a single CAS which had many same files, mentioned in Section 5.1.1. However, the ratio of same chunks between 2 CAS images on 4KB chunk size is almost same to other file systems. It means that tail packing assigned same fractions of files in a block and keeps same chunk on different installation.

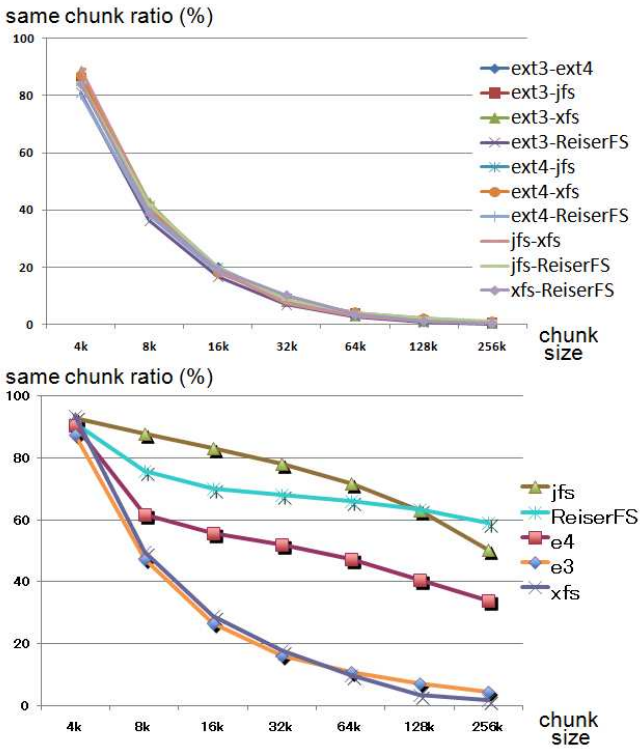

Figure 8: Ratio of same chunks between 2 CAS images on different chunk size from 4KB to 256KB. The upper figure shows the ratio between different file systems. The lower figure shows the ratio between different installations on same file system.

More than 8KB chunks show a ratio reduced inversely proportional to chunk size. 8KB and 16KB chunk sizes have 40% and 20% same chunks respectively on any file systems. The results indicate it is difficult to get same chunks between different file systems. It means we should not use different file systems on CAS system.

The lower in Figure 8 of shows ratio of same chunks between 2 CAS images which install same OS on same file system at different time. The ratios also depend on chunk size but the effects are different. Both jfs and Reiser do not reduce the ratio inversely proportional to chunk size. They keep high ratio of same chunk on larger chunk sizes. Especially jfs and ReiserFS keep 50% same chunks on 256KB chunk size. The results indicate that jfs, Reiser and ext4 allocate most files at same addresses on an installation, but ext3 and xfs allocate different addresses. We will investigate block allocation repeatability in next challenge. The results means we should use same file system on CAS system and the

file system is one of jfs or ReiserFS.

Figure 9 shows the statistics of installed files on ext3. The upper figure indicates the number of files classified by size, and lower figure indicates total volume occupied by files classified by size. The number of files shows the case on ext3 but the results on different file systems are almost same. The lower figure is calculated using a minimum unit of 4KB block and each file is rounded up by 4KB. The calculation causes the big difference on ReiserFS case measured in Table 2, because tail packing reduces the consumption.

The upper figure indicates 77.9% files are less than 4KB. Less than 4KB file use only 1 block and do not affect contiguous allocation of data blocks. The result implies that we do not need to care about contiguous allocation, but less than 4KB files use only 20.1% of the storage showed in lower figure. The remaining portion, consisting of files larger than 4KB, requires contiguous allocation in order to achieve high deduplication. The investigation of the relation of file size and deduplication is not finished. We will continue the research.

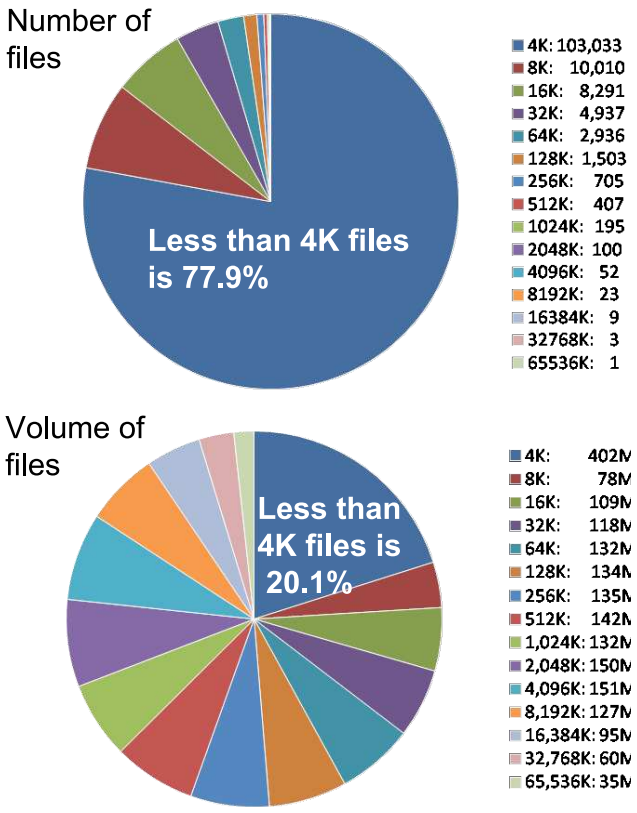

Figure 9: Statistics of installed files on ext3. The upper figure indicates the number of files classified by size, and lower figure indicates total volume occupied by files classified by size. Circular graphs show the percentage of each items (total 100%).

5.2.4 Boot Time (Dynamic Feature)

Table 3 shows statistics of read and write accesses for each file system at boot time. Boot is a read-centric process and has about twice as many read than write operations. The table format is the same as Table 1.

From the view of disk accesses (upper three rows), ReiserFS and XFS are the best in read and write operations, respectively. These features may be responsible for the fastest boot time shown in Figure 6. The largest average access size, however, occurs under ext4 for both operations. It might be a result of disk-prefetching contiguous data blocks allocated by extent allocation.

The lowest number of chunks on 32 KB occurs on ReiserFS and XFS on read and write operations, respectively. The lowest number of chunks on 256 KB occurs on ReiserFS and ext4 on read and write operations, respectively. The lowest number of read operations on ReiserFS explains the fast boot time.

6 Discussions

In this paper we treat CAS, which offers block-level deduplication, but there is another level of deduplication. We compare them in Section 6.1. The results in Section 5.2 lead us the importance of optimization on file system and CAS. We discuss two type of optimization in Sections 6.2 and 6.3.

6.1 Deduplication on file system level

CAS offers block-level deduplication, but deduplication is not limited to the block level. Deduplication can be applied at the file system level, as implemented by lessfs [5] and SDFS [10]. In this case, file system is limited to the original one, and there is no affinity problem with the file system.

File system deduplication means that file system includes the function of deduplication. It detects identical content in files, and merges the same content at the file system level. It does not care about block level restrictions. Namely, it does not care about block alignment matching, contiguous allocation, and contamination by other files.

The evaluation we proposed in this paper is not applied on file system deduplication, because all same-content files are deduplicated perfectly. We already confirm the effect of our evaluation method on lessfs and SDFS. They deduplicate all files well. In order to evaluate file system deduplication, we should use partially-similar-content files. For example, we tried files in which 256-bytes or 257-bytes of random data are repeated. The case with 256-bytes is deduplicated well, but files containing 257-bytes of repeated random data are not deduplicated well on lessfs and SDFS. It means they offer fixed-length deduplication. On fixed-length type, location of same-content data in a file is very important. Variable-length deduplication does not care about location and deduplicates both files well, but requires more comparison time.

File system deduplication has another disadvantage. A file system which has deduplication is usually a pseudo file system and is not usable as a bootable file system, because it is not recognized by boot loader. An operating system on a virtual machine has to use a loop-back file which is a pseudo block device, to install bootable file system. Therefore the affinity problem between file system on a virtual machine and loop-back file supported by file system deduplication will occur again.

6.2 FS Optimization for CAS

Boot time optimization for CAS is proposed in paper [12]. It takes a trace of block accesses on ext3 and reallocates data blocks in the file system. The data blocks in ext3 which are required to boot are arranged in line on the disk. This increases the read-ahead coverage of kernel prefetching. As a result, both the number of accesses and the number of CAS chunks are reduced.

This optimization is necessary for each file system on CAS. Optimization should consider the access profile as well as storage deduplication. Storage deduplication could be further increased by using a binary patch technique. We will investigate a delta encoding method which reuses existing block data.

6.3 CAS Optimization for FS

In cloud computing, the storage system can optimize a virtual disk for the file system used. Classically file systems have been optimized for a disk device. However, a virtual disk on cloud computing, which is managed by key-value storage, could change its behavior for a file system. For example, when a file system prefetches extra data, virtual storage could push the data to memory in advance. We will investigate an intelligent virtual storage based on the analysis of file system features.

7 Conclusions

We analyzed the affinity between nine Linux file systems (ext3, ext4, XFS, JFS, ReiserFS, which are bootable file systems, and NILFS, btrfs, FAT32 and NTFS) and CAS with 32 KB and 256 KB chunks. We proposed a method to evaluate the degree of deduplication by storing many same-content files through a file system and showed the affinity between file system and CAS. We also evaluated file systems on CAS by measuring the access patterns at installation and boot time.

The evaluations with same-content files indicate the degree of deduplication in a file system and show the affinity between file system and CAS. We estimate the effects come from the alignment matching, contiguous allocation of data blocks, and non-contamination with other data. Ext4, btrfs, and NTFS show good affinity for CAS.

At installation and boot time, ReiserFS shows good results, attributable mainly to reduced read and write accesses. The effect of deduplication on ReiserFS is not so high in a single image. ext4 shows good results on deduplication. The affinities between different file systems are little from the view of same chunks in CAS, but jfs and ReiserFS have many same chunks between different installations respectively. The results suggest that there is block allocation repeatability on jfs and ReiserFS. We will investigate it as next challenge.

The results of two types of experiments suggest the possibility of optimization of a file system and a virtual disk. On cloud computing, an intelligent storage system could change its behavior for a file system.

References

[1] FUSE: Filesystem in userspace, *http://fuse.sourceforge.net/*

[2] K. Jin and E. L. Miler, *The Effectiveness of Deduplication on Virtual Machine Disk Images*, The Israeli Experimental Systems Conference, SYSTOR 2009.

[3] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, *kvm: the Linux Virtual Machine Monitor*, Proceedings of Linux Symposium 2007, Volume 1, pp. 225–230, June 2007.

[4] R. Konishi, Y. Amagai, K. Sato, H. Hifumi, S. Kihara, and S. Moriai, *The Linux implementation of a log-structured file system*, ACM Operating Systems Review, Vol. 40 Issue 3, 2006.

[5] lessfs: open Source data deduplication for less, *http://www.lessfs.com/*

[6] A. Liguori, E.V. Hensbergen, *Experiences with Content Addressable Storage and Virtual Disks*, First Workshop on I/O Virtualization (WIOV), December, 2008.

[7] A. Mathur, M.Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, *The new ext4 filesystem: current status and future plans*, Proceedings of Linux Symposium 2007, Volume 2, pp. 21–34, June 2007.

[8] M.A. Olson, K. Bostic, and M. Seltzer, *Berkeley DB*, Proceedings of USENIX FREENIX 1999, pp. 183–191, June 1999.

[9] S. Quinlan and S. Dorward, *Venti: A New Approach to Archival Storage*, Proceedings of the 1st USENIX Conference on File and Storage Technologies, Monterey CA, January, 2002.

[10] SDFS: A user space deduplication file system, *http://www.opendedup.org/*

[11] K. Suzaki, T. Yagi, K. Iijima, and N.A. Quynh, *OS Circular: Internet Client for Reference*, Proceedings of the 21st Large Installation System Administration Conference, pp.105-116, Dallas TX, November, 2007.

[12] K. Suzaki, T. Yagi, K. Iijima, C. Artho and Y. Watanabe, *Effect of Disk Prefetching of Guest OS on Storage Deduplication*, ASPLOS Workshop RESoLVE 2011.