Effect of readahead and file system block reallocation for LBCAS

(LoopBack Content Addressable Storage)

Kuniyasu Suzaki, Toshiki Yagi, Kengo Iijima, Nguyen Anh Quynh,
Yoshihito Watanabe

National Institute of Advanced Industrial Science and Technology
Alpha Systems Inc.

{k.suzaki,yagi-toshiki,k-iijima,nguyen.anhquynh}@aist.go.jp
watanays@alpha.co.jp

Abstract

Disk pre-fetching, known as "readahead" of Linux kernel, arranges its coverage size by the rate of cache hit. Fewer readaheads of large window can hide the slow I/O, especially it is effective for virtual block device of virtual machine. High cache hit ratio is achieved by increasing locality of reference, namely, file system block reallocation based on an access profile.

We have developed a data block reallocation tool for ext2/3, called "ext2/3optimizer". The relocation is applied to Linux booting on KVM virtual machine with a virtual disk called LBCAS (LoopBack Content Addressable Storage). We compared the effect with the Linux system call "readahead" which populates the page cache with data of a file in advance. From the experiment, we confirmed that the reallocation of ext2/3optimizer kept larger coverage of readahead and fewer I/O requests than the system call readahead. The reallocation also reduced the overhead of LBCAS and made quick boot.

1 Introduction

We have developed a framework of Internet Disk Image Distributor for anonymous operating systems, called "OS Circular[1, 2, 3]". It enables to boot anonymous OS on any real/virtual machines without installation. The disk image is distributed by LBCAS (LoopBack Content Addressable Storage) which manages the virtual disk efficiently. The transferred OS is maintained periodically on the server and fixed vulnerable applications. The partial update is managed by LBCAS efficiently and the user can rollback to old OS.

LBCAS is a kind of loopback block device managed by CAS (Content Addressable Storage) [4, 5, 6]. CAS retrieves a data-block with the hash value of its content. Thus, CAS is a kind of indirect access management of physical address. It is used for permanently information archive because CAS distinguishes every data-block with hash value and reserves old blocks, although direct physical access overwrites the previous data. When block contents are same, they are held together with same hash value and reduce total volume.

Unfortunately, CAS is known as an archive method which behavior is affected by feature of stored data and access patterns [5, 6]. The characteristics differ from a real device and require careful handling. A performance gap is caused by the coordination of disk pre-fetching (i.e., page cache) of existing OS. For example, Linux kernel has the function called "readahead [7, 8]" which pre-fetches some extra blocks from block device. The coverage of readahead is changed by heuristics of page cache. LBCAS should be optimized for the access patterns, namely locality of reference.

In order to increase the locality of reference, we have developed "ext2/3optimizer [9]" to reallocate the data block of ext2/3 file system. The reallocation follows the access profile. The accessed data blocks are arranged to be in line. In this paper we compare the effect of ext2/3optimizer with the Linux system call "readahead" which populates the page cache with data of a file in

advance.

The remainder of this paper is organized in six sections. In section 2 the detail of LBCAS is described. Readahead and its relation to LBCAS are described in section 3. Section 4 reallocation method is described. The performance is evaluated in section 5. Some related topics are discussed in section 6 and conclusion is mentioned in section 7.

2 LBCAS: LoopBack Content Addressable Storage

LBCAS is made from an existing block device. The block device is divided by a fixed block size and saved to each small block file. Saved data are also compressed. Each block file has a name of SHA-1 of its contents. The address of block files is managed by mapping table. The mapping table has the relation information of physical address and SHA-1 file name. A virtual block device is reconstructed with the mapping table file on a client. Figure 1 shows the creation of block files and mapping table file “map01.idx”.

Block files are treated as network transparent between local and remote. Local storage acts as a cache. The files are measured with SHA-1 hash value of its contents when they are mapped to virtual disk. It keeps the integrity for Internet block device.

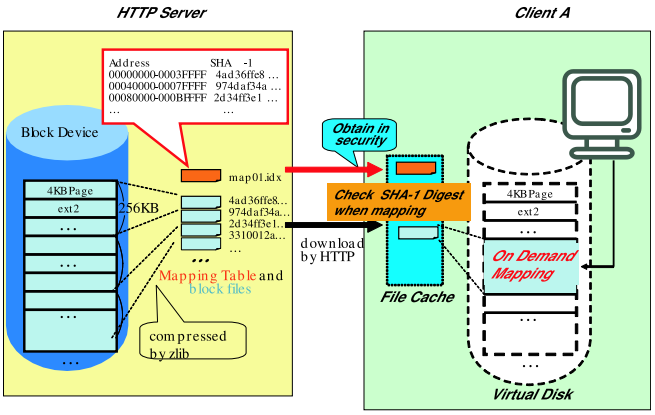

Figure 1: Creation of block files from OS image.

Figure 2 shows the diagram of LBCAS structure. A loopback file is re-constructed with downloaded block files on a client. The main program is implemented as a part of FUSE (File system in USEr space [10]) wrapper program. LBCAS has two level of cache to prevent redundant download and uncompression, which is

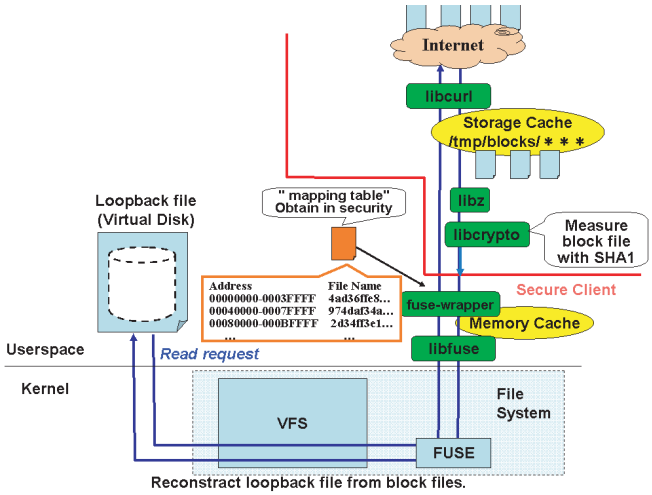

Figure 2: Creation of block files from OS image.

called “storage cache” and “memory cache”. The detail of cache is described in next subsection.

A client has to obtain a mapping table file in security. The mapping table file is used to setup LBCAS. When a read request is issued, LBCAS driver searches a relevant block file with the mapping table. If a relevant file exists on a storage cache, the file is used. If not, the file is downloaded from a HTTP server with “libcurl”. Each downloaded file is uncompressed by “libz” and measured with the SHA-1 value by “libcrypto”. The measurement of SHA-1 value of block file is logged. Even if a block file is broken or falsified, the block is detected. Figure 3 shows the case to detect a falsified block file.

2.1 Two level of Cache

Current LBCAS has two level of cache, which is called “storage cache” and “memory cache”.

Storage cache saves the downloaded block files at a local storage. It eliminates the download of same block file. If the necessary block files are saved at storage cache, the LBCAS works without network connection. The volume of storage cache is managed by water mark algorithm of LIFO in current implementation. The latest downloaded block files are removed when the volume is over the water mark, because aged block files might be used for boot time.

Memory cache saves the uncompressed block file at the memory of LBCAS driver. It eliminates uncompression

```
1150452051.109: #00000000(845b31ded38e15c1fa8febf97fe0781f23af98c3) :missed.
1150452051.112: #00000000(845b31ded38e15c1fa8febf97fe0781f23af98c3) :hits.
1150452051.112: #00000001(166cbaedbb1cc836e7c95d7d9943efde5a53829e) :missed.
1150452051.113: #00000002(29c4e363dbad648072751ca1f856e5780dd2981d) :missed.
1150452051.114: #00000003(fa8ad05b713a9cf8a701636ca6c353dc58fd6bfd) :missed.
1150452051.114: #00000004(1f82a543fa9310c44eff6a13618beca3cacffc12) :missed.
1150452051.128: #00000004(1f82a543fa9310c44eff6a13618beca3cacffc12) :hits.
1150452051.128: #00000005(916f62a6e2caedc1279a0a74975a406ddb60ec25) :missed.
1150452051.129: #00000006(19111dfc877a4fe241e125d10176d85a99b4bb86) :missed.
1150452051.130: #00000007(950c1d7623b374f8e03309a93041f5adfa3af80f) :missed.
1150452051.130: #00000008(486472b0ee27157d755bd59d623179cfc0034747) :missed.
```

```
1150452375.989: #00000000(845b31ded38e15c1fa8febf97fe0781f23af98c3) :missed.
1150452375.993: #00000000(845b31ded38e15c1fa8febf97fe0781f23af98c3) :hits.
1150452375.993: #00000001(166cbaedbb1cc836e7c95d7d9943efde5a53829e) :missed.
1150452375.994: #00000002(29c4e363dbad648072751ca1f856e5780dd2981d) :missed.
1150452375.995: #00000003(fa8ad05b713a9cf8a701636ca6c353dc58fd6bfd) :missed.
1150452375.996: #00000004(1f82a543fa9310c44eff6a13618beca3cacffc12) :missed.
1150452375.997: #00000004(1f82a543fa9310c44eff6a13618beca3cacffc12) :hits.
1150452375.997: #00000005(916f62a6e2caedc1279a0a74975a406ddb60ec25) :missed.
1150452375.998: #00000006(19111dfc877a4fe241e125d10176d85a99b4bb86) :missed.
E: can't validiate block.
```

Figure 3: Log of LBCAS. The upper shows the correct downloading of block files and the lower shows a falsified block file is detected. “missed” indicates downloading a block file. “hits” indicated finding a block file at local storage.

when same block file is accessed in succession. Memory cache saves 1 block file and the coverage is the size of block file. It should be coordinated with the page cache of existing OS.

2.2 Partial Update by Adding Block Files

The update of LBCAS is achieved by adding block files and renewing the mapping table file. The rest block files are reusable. To achieve this function, the file system on LBCAS has to treat block-unit update as ext2/3 file system. ISO9660 file system is not suitable because partial update of ISO9660 changes the location of following blocks.

The updated block is saved to a file with new file name of SHA-1. Collision of file name will be rarely happened. Even if a collision happens, we can check and fix it before uploading the block files on the servers. We can rollback to the previous file system if the old mapping table and block files exist.

2.3 Issues of Performance and Behavior on LBCAS

The behavior of CAS is affected by feature of stored data and access patterns [5, 6].

The block size of CAS causes problems of fragmentation and boundary. One problem comes from the size mismatch of the block size of file system. Most file system assumes their block size is 4KB but LBCAS uses larger block size because of efficiency of loopback device and network download. It results in low occupancy rate, redundant download and unnecessary uncompression.

When the occupancy, which is a ratio of effective data in a block file, is low, the overhead of LBCAS is not negligible. The block size of LBCAS should be considered the occupancy. The problem is closely related to locality of reference on transferred OS.

When a read request crosses over the boundary of CAS, CAS requires multiple blocks. If the block size of CAS is too small and requires many blocks for an I/O request, performance gap stands out. The block size of CAS must be balanced to I/O request size. The problem is related to the window size of pre-fetching.

3 Readahead(Disk Pre-fetching)

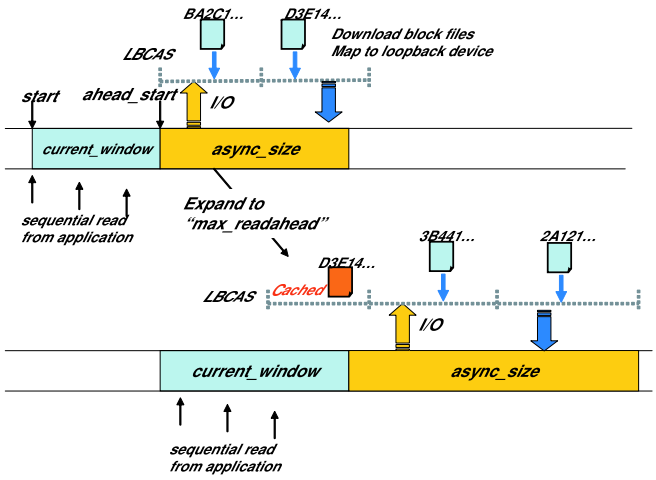

Figure 4: Behavior of readahead on LBCAS

Most operating systems have the function of page cache to reduce I/O operation. When a read request is issued, the kernel reads extra data and saves them to main memory (page cache). It reduces the number of I/O operation and hides the I/O delay. The function in Linux kernel is called “readahead [7, 8]”. The coverage of readahead is extended or shrank by the profile of cache hit and miss-hit. Figure 4 shows the action of readahead on LBCAS. When a readahead operation is issued, some block files are downloaded and mapped to the loopback device. When a same block file is required sequentially,

the block file is stored on the memory cache of LBCAS and the uncompression is eliminated. When page cache does not hit, the next readahead shrink the coverage size. The suitable size of coverage achieves efficient usage of cache memory and I/O request.

The readahead causes performance gaps on LBCAS, when the extra coverage crosses over the boundary of LBCAS block (Figure 5). The third read request in the figure crosses over the boundary, and extra block file is downloaded which is never used. It causes big performance penalty compared to real block device. The problem is cased by un-contiguous of necessary blocks.

The un-contiguous blocks are improved by ext2/3optimizer described in next subsection. High hit ratio of page cache and large coverage of readahead means the less I/O requests. The access pattern will be efficient on the LBCAS.

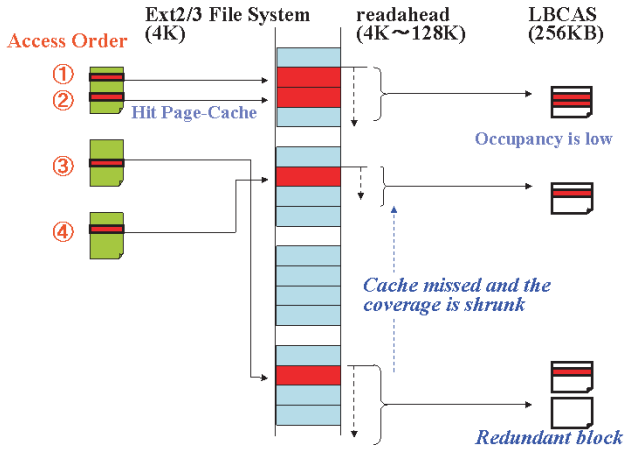

Figure 5: Behavior of readahead on LBCAS

3.1 system call “readahead”

Linux kernel has the system call “readahead” from 2.4.13. The system call populates the page cache with data of a file. It is not directly related to the disk pre-fetching but it can achieve the same function from user space, because subsequent reads from that file will not block on disk I/O. Linux distributions have a tool to utilize the readahead system call to make quick boot. The files opened at boot time are listed at “/etc/readahead/boot” and the data of the files are populated on the page cache in advance at boot time.

Unfortunately it requires much memory and has no dynamic flow control. The speed-up depends on individual machine. In this paper we confirm the effect on a virtual machine.

4 Block reallocation of File System

Most file systems have defragmentation tools to reallocate blocks of file system. For examples, defrag and ext2resize are tools for ext2. The tools however reallocate blocks from the view of continuation of file and expansion of spare space. Quick access is a side effect of continuation of file.

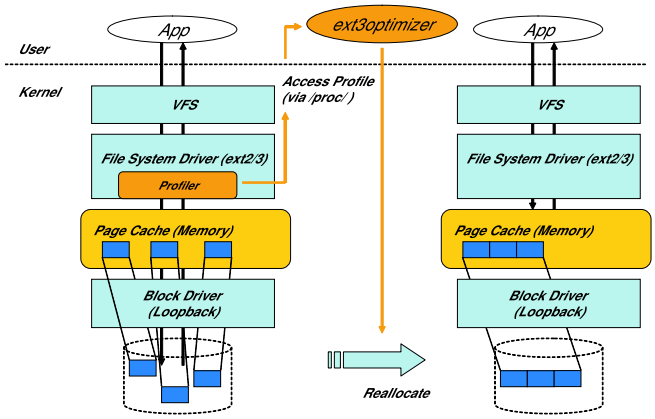

Figure 6: Access profiling and reallocation which increase cache hit ratio and coverage of readahead.

In order to solve the problem, we developed “ext2/3optimizer” [9], which was called ext2optimizer. Ext2/3optimizer takes the profile of accessed blocks of ext2/3 and reallocates the blocks in line. Figure 6 shows the image of profiling and reallocation. The reallocation increases the cache hit ratio and expands the coverage size of readahead. The effect is described in next section.

Ext2/3optimizer change pointers of data blocks of i-node only. It aggregates the data blocks at the head of device and increase locality of reference. The other structure of ext2/3, namely meta-data of ext2/3, is reserved. Figure 7 shows the image of reallocation of ext2/3optimizer.

Block size mismatch problem between file system and LBCAS is reduced by the aggregation of data blocks, because it increases the occupancy of effective data in a block file. Figure 7 shows the higher occupancy reduces the necessary block files. In this case the data on 3 block files is aggregated in 1 block file. The effect is also described in next section.

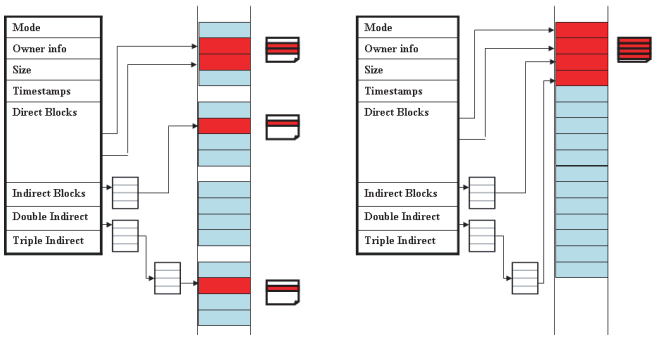

Figure 7: Reallocation of ext2/3optimizer.

5 Performance of ext2/3optimizer and user-level readahead on LBCAS

We compared the effect of ext2/3optimizer and user-level readahead (system call readahead) on LBCAS was evaluated. We applied both of them to the gust OS on KVM virtual machine [11] (version 60) with LBCAS, which shows the feasibility of OS migration. Ubuntu 9.04 (Linux kernel 2.6.28) was used for the transferred OS. Ubuntu was installed on 8GB loopback file with ext3 file system on KVM using normal installer. The total volume was 1.98GB. The block files of LBCAS were made from the loopback file.

The access pattern of boot procedure is random and does not read whole contents in a file. Sparse access will be increased, and the coverage of readahead will be narrow. As a consequence, the occupancy of block file will be low and the efficiency of LBCAS becomes worse. In this section we confirmed the characteristics and applied optimizations for it. From after we refer user-level readahead as "u-readahead" in order to distinguish the disk pre-fetch readahead.

5.1 Block Reallocation: ext2/3optimizer

Figure 8 shows the data allocation on ext3, which is visualized by DAVL (Disk Allocation Viewer for Linux) [12]. The left figure shows the original data allocation, and right figure shows the data allocation optimized by ext2/3 optimizer.

The green plots in the figure indicate the allocation of meta data of ext3 which was arranged at the right edge. We confirmed that ext2/3optimizer keeps the structure

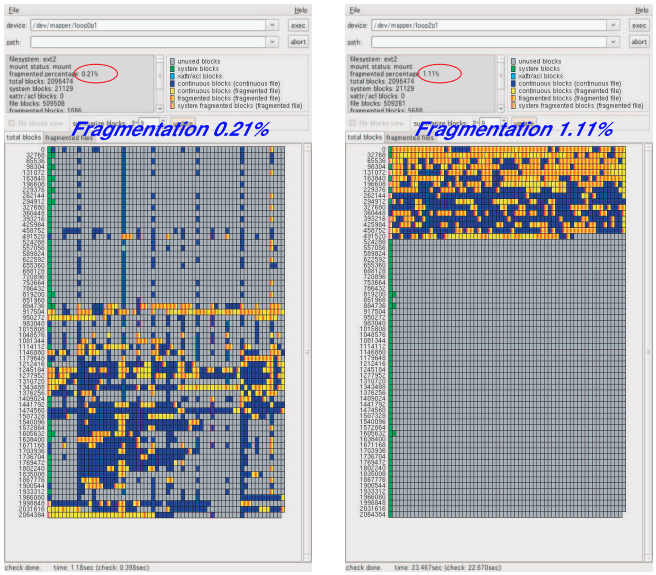

Figure 8: Visualization of data-allocation on ext3 (left is normal and right is ext2/3optimizer) by DAVL.

of ext3. The blue plots indicate the contiguous allocation of data block of file and the yellow plots indicate the non-contiguous allocation. We confirmed that ext2/3optimizer reallocates non-contiguous data at the head of disk. It was the result that ext2/3optimizer exploited the profiled data blocks and aggregated them to the head of the disk. As the result, ext2/3optimizer increased fragmentation from the view of file. DVAL showed that normal ext3 had 0.21% fragmentation but the ext3 optimized by ext2/3optimizer had 1.11%. The relocation however was good for page cache. The coverage of readahead was expected to keep large and occupancy of block file of LBCAS would be high.

Figure 9 shows the access trace of the boot procedure. The x axis indicates the physical address and y axis indicates the elapsed time. The red "+" plots indicate the access on the normal ext3 and the blue "X" plots indicate the access on the ext3 optimized by ext2/3optimizer. The figure showed that the accesses to the normal were scattered. The locality of reference was not good and the effect of page cache and the occupancy of block file of LBCAS would be low. On the other hand, the access to the ext2/3optimizer increased the locality of reference, because the most accesses were the head of disk. The rest spread accesses were the meta data and the volume was little.

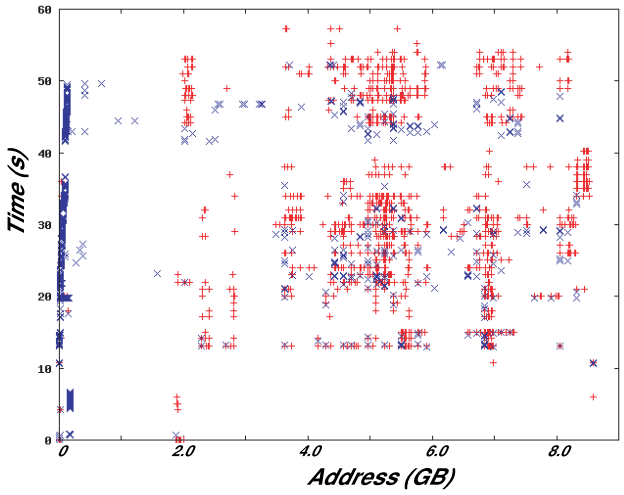

Figure 9: Access trace of boot procedure (RED "+" indicates normal and BLUE "X" indicates ext2/3optimizer.)

5.2 User level readahead: system call "readahead"

Ubuntu has the mechanism to populate the page cache with files required at boot time. The files are described at "/etc/readahead/boot" and "/etc/readahead/desktop". The former file listed 937 files and the total volume was 54.1MB. The latter file listed 281 files and the total volume was 25.0MB. the listed files are not all files required boot time. Ubuntu 9.04 requires 2,250 files (203MB) and the half of them are populated on the page cache before they are truly required.

Figure 10 shows the log of bootchart [13], which visualizes the behavior of CPU, I/O, and creation of process at boot time. We confirmed that the same processes were executed at boot time on normal, u-readahead and ext2/3optimizer. The result of u-readahead shows the utilization of I/O increased when u-readahead started. It caused the spike of I/O but the subsequent I/O was little. On the other hand, the I/O was issued on-demand on the normal and ext2/3optimizer. The I/O of ext2/3optimizer was less than the normal. The detail is described in Section 5.3.

Table 1 shows the utilization of CPU and I/O on normal, u-readahead and ext2/3optimizer on 64KB, 128KB, 256KB and 512KB LBCAS. The results shows the u-readahead had higher I/O utilization. It was caused by the redundant read request, because u-readahead read the whole data of files. The I/O utilization of u-readahead was 5-2 times higher than ext2/3optimizer. The results of 512KB LBCAS showed bad I/O utilization on any case. It was caused by the slow response of 512KB LBCAS.

5.3 Effect of readahead

Figure 11 shows the Frequency for each readahead coverage size on normal, u-readahead, and ext2/3optimizer.

The figure shows that ext2/3optimizer reduced the small I/O requests. As the result, the frequency of I/O request was reduced to 2,129 from 6,379 and the coverage of readahead was changed to 67KB from 33KB. The total I/O was 140MB and 208MB on ext2/3optimizer and normal respectively. The I/O request is 2 times wider and the frequency of I/O request is 1/3. The effect of frequency is not the inverse of magnification of I/O. The results indicated that the locality of reference is much improved.

On the other hand, u-readahead showed same tendency with normal. The small requests were reduced and the big request were increased a little bit. The total I/O of u-readahead was increased to 231MB from 208MB of the normal. The coverage of readahead was expanded to 41KB but it was small than ext2/3optimizer. The result came from that the u-readahead could not decrease the small I/O, which was cased by the locality of reference. The frequency of I/O was 5,827, which was less than normal 6,379, although the total I/O was increased. The results indicated that ext2/3optimizer was much effective than u-readahead from the view of disk pre-fetch readahead.

5.4 Total performance

Figure 12 shows the detail of Ubuntu boot time on KVM from LBCAS. The upper figure shows the ratio of time consumed by LBCAS for each block size. The lower figure shows the consumed time in LBCAS, which was consisted of download time, file read time from storage cache, uncompression time, and time for others.

From the upper figure, we confirmed that ext2/3optimizer was effective for LBCAS at any block size. The LBCAS consuming time was more than 20% on normal, but it was reduced to less than 14% on ext2/3 optimizer. Although the total I/O on u-readahead was more than the normal, the boot time on u-readahead

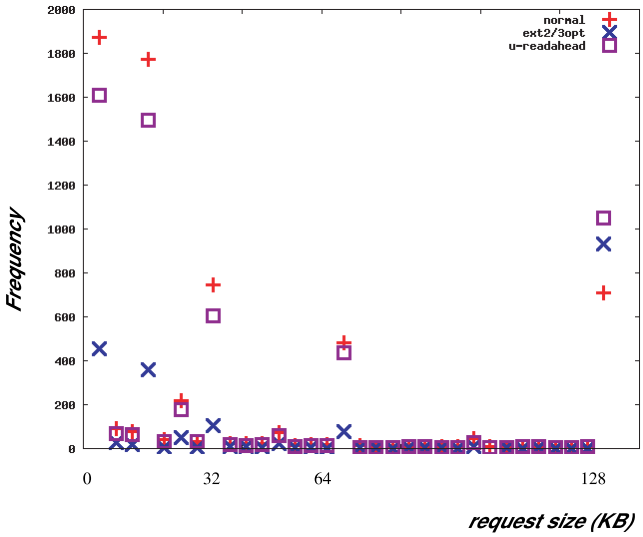

Figure 11: Frequency for each readahead coverage size.

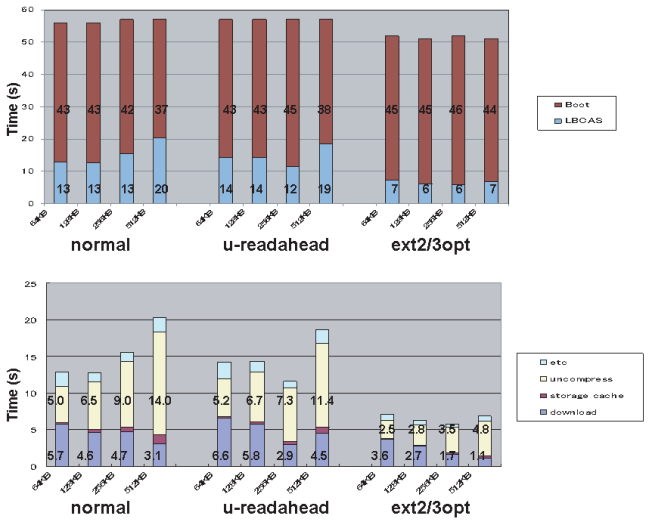

Figure 12: The ratio of consumed time. Upper indicates the ratio of LBCAS in boot procedure. Lower indicates the contents ratio of LBCAS.

was almost same result on normal, The result indicated the page cache populated by u-readahead but it was not effective on KVM.

The lower figure shows the time consuming components of LBCAS. The result shows the most time was consumed by download and uncompression. The download time was longer than the uncompression time at small LBCAS but it was changed at large LBCAS. It was caused by the locality of reference because the small LBCAS was effective from the view of occupancy but it required many block files. On large LBCAS the time of uncompression was increased because of low occupancy in a block file, but the time of download became short because the number of download was fewer and cached on the storage cache. On the 512KB LBCAS the time of uncompression was increased and the total time of LBCAS was the worst.

Table 2 shows the volume transitions at each processing level. The upper table shows the total volume requested from transferred OS: the volume of files which opened by the boot procedure, the block volume which is purely required by the boot procedure, the volume accessed to LBCAS (it includes redundant data covered by readahead). The bottom table shows the status of LBCAS for each block size: the volume of downloaded block files, the volume of uncompressed block files, and the occupancy of effective data in the LBCAS.

From the result, we know that the purely used block was 63% (127MB/203MB) of volume of opened files at boot time. It meant that 37% was not used and it caused inefficient access request of readahead. The readahead for normal ext2 required 208MB access to the LBCAS. The result shows the 81MB (208MB - 127MB) was redundant access. The u-readahead made much worse and 104MB was redundant access. The problem was solved by ext2/3optimizer significantly. The readahead for ext2/3optimize required 140MB. The ext2/3optimizer made 67% better than the normal.

The bottom table shows the status of LBCAS. We confirmed that downloaded files were less than 56MB at any LBCAS size on ext2/3optimizer. However, the normal of 512KB LBCAS requires 144MB, which is 1.67 much larger than 64KB LBCAS (86.1MB). It was caused by bad locality of reference. On ext2/3optimize, the occupancy was almost same on any LBCAS size but it was decreased from 51.5% at 64KB LBCAS to 26.9% at 512KB LBCAS on normal. The result indicated that block reallocation was necessary for LBCAS.

Table 3 shows the frequency of each function of LBCAS for normal, u-readahead, and ext2/3optimizer. I/O requests were issued by guest OS and the frequency was independent of LBCAS. The rest columns indicated the function of LBCAS. The number of uncompress is summation of the number of download and storage cache. The summation of uncompress and memory cache is the total used files on the LBCAS.

The results showed storage cache and memory cache worked well. Especially the two caches were effective on large LBCAS size. The frequency of storage cache and memory cache were more than the frequency of download and uncompression.

The uncompression of ext2/3optimizer was less than half of normal case at each LBCAS size. The result corresponds to time of uncompression at the Figure 12. The decrease affected the performance of LBCAS.

Figure 13 shows the amount of downloaded block file at boot time. The LBCAS size was 256KB. The result shows the ext2/3optimizer reduced the amount of download and made quick boot. The u-readahead downloads many block files around 15 second because it populated the page cache with listed files. It increased the total download but made quick boot. However the boot time of u-readahead was slower than ext2/3optimizer.

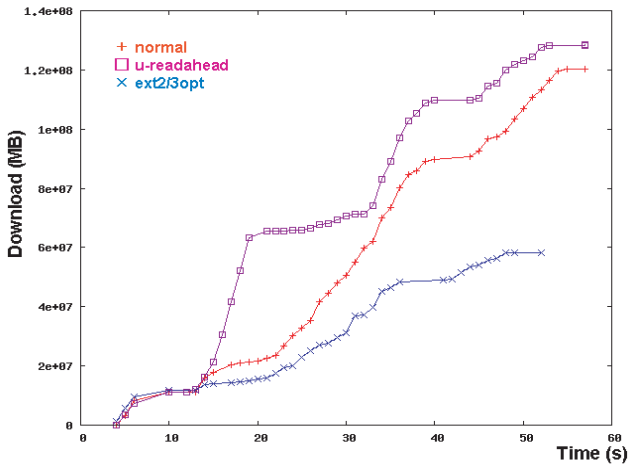

Figure 13: Amount of Downloaded Block File (256KB) at boot time.

6 Discussions

The data blocks are reallocated in order to be in line according to the access profile. It results in keeping large coverage of readahead at the boot procedure. It makes quick boot but the data blocks are fragmented from the view of file. The optimization is too tight and it would not fit to another access pattern. If the reallocated data blocks are used in another application, the access pattern can not get large coverage of readahead. However, boot procedure is special and several files are used at boot procedure only. We have to estimate the special files and its ratio, which are not used for other applications.

Most reallocation tools aim to reduce fragmentation and quick access is side effect. Unfortunately the effect of quick access looks to be insufficient, even if the File System has no fragmentation. The original disk image used in section 5 was first install image and there are few fragmentation. The trace of boot procedure, however, showed discrete access. In order to make quick access we should reallocate blocks based on access profile.

A feature of CAS is sharing of block with same hash value. It reduces the total volume of contents. The sharing is not effective on a single OS image but it is effective on some Linux distributions [14] and multi user environment [15]. [14] told the CAS block sharing on Fedora, Ubuntu, and OpenSuse was 10% - 30%. Although our paper does not describe the effect of sharing, the ext2/3optimizer will reduce the effect because it reallocates most of data blocks in ext2/3 file system. If the sharing is important factor, the reallocation tool has to consider the sharing as far as possible.

7 Conclusions

We offered an virtual block device called “LBCAS”, which manages each block by indirect mapping of SHA-1 value of its contents. The performance was affected by the number of I/O request which is issued by readahead of disk pre-fetching. The number of I/O request is related to the coverage size of readahead. The coverage is expanded by high hit ratio of page cache The hit ratio is increased by locality of reference.

We developed ext2/3optimizer to reallocate the data block of ext2/3 according to the access profile. We applied ext2/3optimizer on Ubuntu 9.04 according to the access profile of boot procedure. We compared the effect with the user-land readahead which uses Linux system call “readahead” and populates the page cache with data of files in advance. As the result, the coverage of readahead expanded to double and the I/O requests reduced in half on ext2/3optimizer. The user-land readahead could also expand the coverage of readahead and reduce the I/O requests but the effect was less than ext2/3optimizer.

The key was the locality of reference which is improved by ext2/3optimizer. The effect of locality of reference also reduced the necessary block files on LBCAS at boot time. The result showed that the optimization was necessary for the OS migration with LBCAS, which is aimed of the OS Circular project.

The source code of tools are available at the project home page.

References

[1] http://openlab.jp/oscircular/

[2] K. Suzaki, *OS Circular: Internet bootable OS Archive*, LinuxConf.Australia, January, 2009.

[3] 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.

[4] 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.

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

[6] Mechiel Lukkein, *Venti analysis and memventi implementation*, Master's thesis of University of Twente, 2008.

[7] WU. Fengguang, XI. Hongsheng, and XU. Chenfeng, *On the design of a new Linux readahead framework*, ACM SIGOPS Operating Systems Review, Volume 42, Issue 5, pp. 75–84, July, 2008.

[8] WU. Fengguang, XI. Hongsheng, J. Li, and N. Zou, *Linux readahead: less tricks for more*, Proceedings of the Linux Symposium, Vol.2, pages 273–284, 2007.

[9] K. Kitagawa, H. Tan, D. Abe, D. Chiba, K. Suzaki, K. Iijima, and T. Yagi, *File System (Ext2) Optimization for Compressed Loopback Device*, 13th International Linux System Technology Conference, pp. 25–33, Nurnberg Germany, September, 2006.

[10] http://fuse.sourceforge.net/

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

[12] http://sourceforge.net/projects/davl/

[13] http://www.bootchart.org/

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

[15] P. Nath, M.A. Kozuch, D.R. O'Hallaron, J. Harkes, M. Satyanarayanan, N. Tolia, and M. Toups, *Design Tradeoffs in Applying Content Addressable Storage to Enterprise-scale Systems Based on Virtual Machines*, USENIX Annual Technical Conference, pp. 71–84, Boston MA, 2006.

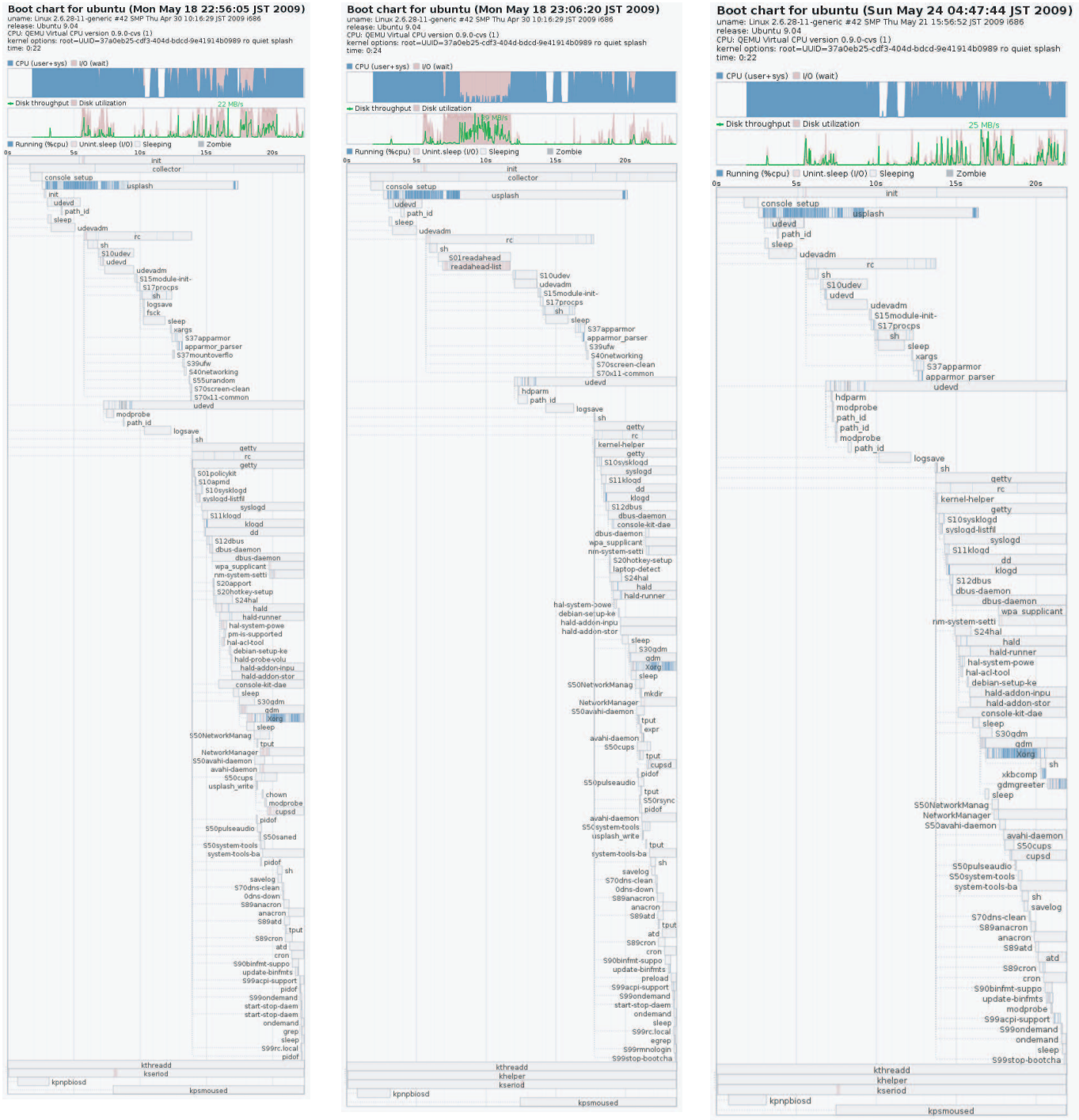

Figure 10: BootChart. The visualization of CPU utilization, I/O utilization and created processes at boot time. The left is normal, the middle is u-readahead, and the right is ext2/3optimizer.

LBCAS size	normal		u-readahead		ext2/3opt	
	CPU (%)	I/O (%)	CPU (%)	I/O (%)	CPU (%)	I/O (%)
64KB	87.8	12.2	74.1	25.9	94.9	5.1
128KB	84.9	15.1	81.1	18.9	94.4	5.6
256KB	86.0	14.0	79.4	20.6	93.7	6.3
512KB	78.7	21.3	74.3	25.7	89.0	11.0

Table 1: Utilization of CPU and I/O, which was taken by BootChart.

	Normal	u-readahead	ext2/3optimizer
Volume of files (number, average)	203MB (2248 Av:92KB)		
Volume of requested blocks	127MB		
Volume of required access which includes the coverage of Readahead (average number of access and size of readahead)	208MB (6,379 Av:33KB)	231MB(5,827 Av:41KB)	140MB(2,129 Av:67KB)
LBCAS size	Downloaded size MB (Uncompressed size MB), Occupancy %		
64KB	86.1(247), 51.5%	93.4(272), 46.9%	55.3(144), 88.7%
128KB	96.8(290), 43.9%	104(315), 40.3%	55.3(149), 85.3%
256KB	114(358), 35.5%	123(386), 35.0%	55.6(159), 80.0%
512KB	144(474), 26.9%	153(508), 25.1%	55.6(176), 71.8%

Table 2: Volume transitions at each processing level. The upper table indicates the volume transition on guest OS. The bottom table indicates the volume transition on LBCAS.

Normal	Requests (Av. size 33KB) (R)	Download (D)	Storage Cache (S)	Uncompress (U)=(D)+(S)	Memory Cache (M)	Files per request (R)=(1)+(2)+(3) (U)+(M)=(1)+(2)*2+(3)*3
64K	6,338	3,958	1,663	5,621	3,647	(1) 4,148 (2) 1,450 (3) 740
128K	6,381	2,321	1,729	4,050	3,793	(1) 4,919 (2) 1,462
256K	6,379	1,435	1,748	3,183	3,908	(1) 5,667 (2) 712
512K	6,395	948	1,769	2,717	4,019	(1) 6,054 (2) 341
u-readahead	(Av: size 41KB)					
64K	5,825	4,344	1,172	5,516	3,626	(1) 3,537 (2) 1,259 (3) 1029
128K	5,834	2,526	1,200	3,726	3,761	(1)4,181 (2) 1,653
256K	5,827	1,544	1,179	2,723	3,908	(1) 5,023 (2) 804
512K	5,822	1,015	1,172	2,187	4,023	(1) 5,434 (2) 388
ext2/3opt	(Av: size 67KB)					
64K	2,165	2,296	626	2,922	1,311	(1) 941 (2) 380 (3) 844
128K	2,148	1,189	593	1,782	1,398	(1) 1,116 (2) 1,032
256K	2,129	634	576	1,210	1,409	(1) 1,639 (2) 490
512K	2,132	353	517	870	1,520	(1) 1,874 (2) 258

Table 3: Frequency of function of LBCAS. Upper table shows the normal case. Lower table shows the ext2/3optimizer case. “Requests” indicates the number of I/O issued by guest OS. The rest columns show the frequency of each function of LBCAS. “Files per request” indicates the frequency of downloads for files per a request.

Proceedings of the Linux Symposium

July 13th–17th, 2009
Montreal, Quebec
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*

Programme Committee

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
James Bottomley, *Novell*
Bdale Garbee, *HP*
Dave Jones, *Red Hat*
Dirk Hohndel, *Intel*
Gerrit Huizenga, *IBM*
Alasdair Kergon, *Red Hat*
Matthew Wilson, *rPath*

Proceedings Committee

Robyn Bergeron
Chris Dukes, *workfrog.com*
Jonas Fonseca
John 'Warthog9' Hawley

With thanks to
John W. Lockhart, *Red Hat*