Domain Level Page Sharing in Xen Virtual Machine Systems*

Myeongjae Jeon, Euiseong Seo, Junghyun Kim†, and Joonwon Lee

CS Division, Korea Advanced Institute of Science and Technology
{mjjeon,ses,joon}@calab.kaist.ac.kr
Samsung Electronics Co., Ltd.†
junghyunx2.kim@samsung.com

Abstract. The memory size limits the scalability of virtual machine systems. There have been some researches about sharing identical pages among guest systems to reduce memory usage. However, they require memory overcommitment feature through swap mechanism which some virtual machines including Xen do not have. In this paper a new approach is proposed to share identical pages with designated sharing area. This approach reduces the memory usage as well as redundant I/O operations. Moreover, understanding the characteristics of certain shared pages becomes easier. The conceptional design was evaluated by simulation based on real-world applications.

Key words: virtual machine, parallelism, memory management

1 Introduction

The virtual machine systems, of which the heyday has been thought to be ended in 1970s, are now being resurrected due to the rapid improvement of hardware performance and storage capacity.

However, the scalability of virtual machines is limited by the hardware resources. The resources are able to be categorized into two groups; one is time sharing resources and the other one is space sharing ones. The shortage of time sharing resource is somewhat tolerable because it only induces some latency of processing time. The shortage of space sharing resources, however, such as memory and storage sometimes prohibits the addition of guest systems.

To overcome the scalability limitation from memory sharing, some approaches have been introduced following a way that a guest system shares existing identical pages which were allocated to other guest systems. However, these solutions are beneficial only with memory overcommitment feature. The overcommitment of memory in virtual machine systems means that the sum of the memory sizes that are allocated to guest systems is allowed to exceed the real hardware memory size. The most intuitive method to support memory overcommitment is

* This work was supported by KOSEF grant funded by the Korea government(MOST) (No. R01-2006-000-10724-0) and also partially funded by the MIC, Korea, under the ITRC support program supervised by the IITA.

using swap mechanism as in traditional operating systems. However, Xen [1] which is the representative of massive virtual machine system does not employ swap mechanism because of its design philosophy. Thus, the overcommitment of memory in Xen is restricted. As a result, the described solutions are not able to be directly adopted in Xen architecture.

This paper suggests page sharing scheme that the shared area is placed not in the guest operating systems but in the special designated area. This approach will aid the implementation of memory overcommitment in Xen because it helps to understand the property of shared contents well and to manage the size of shared memory area easily. There are permanent shared regions like operating system kernel and essential libraries. Thus, by checking the size of permanent shared area and shared counts, Xen can decide how much memory will be overcommitted safely.

The rest of this paper is organized as follows: Section 2 reviews existing research on page sharing among guest systems and Xen architecture also. Section 3 describes our approach for domain level page sharing scheme to support memory overcommitment in Xen. Section 4 presents the evaluation results. The new design concept for memory overcommitment without swap mechanism by using our approach is proposed in Section 5. Finally, section 6 summarizes our conclusions.

2 Back Ground

2.1 Xen Architecture

Guest systems in Xen are called *domains*. Xen virtual machine manager which is also called as *hypervisor* is located between hardware and domains and it distributes various resources to each domain. There is a special domain which is called *Domain-0*. It acts as control center that starts, stops and terminates other domains. The other domains which are ordinary guest systems are called *Domain-U*.

In virtual systems real memory can not be directly provided to each guest system because it may hinder the isolation among guest systems. Thus, Xen provides some real memory pages to each guest system as much as the virtual physical memory size of the corresponding guest system. To distinguish clearly, in terms of Xen, real memory is called *machine memory* and the virtual physical memory for each domain is called *pseudo-physical memory*. As in native environment, operating system in each domain distributes pseudo-physical memory to user-level tasks.

I/O requests from each guest system are actually handled in the Linux kernel located at Domain-0. Xen forwards I/O requests of all Domain-U to Domain-0. The forwarded requests will be handled with standard kernel functions in Domain-0. This model makes Xen have simple design and provide full standard and powerful interfaces to guest systems.

As a medium of delivering the I/O request, a structure called I/O ring lies in between Domain-0 and Domain-U. I/O ring consists of request and response

queues. Each queue is a circular queue and follows producer/consumer style. Domain-U produces I/O requests and put them to a queue in I/O ring. And then Domain-0 takes the requests at the other end of the queue and processes them. On the contrary, the processed results are put into I/O ring by Domain-0 and Domain-U gets the results for the requests they made from I/O ring.

Xeno Linux which is the modified Linux kernel for Xen employs front-end block device driver instead of standard block device driver to handles I/O requests for block devices such as hard disk and CD-ROM. Front-end block device driver puts the I/O requests of Domain-U into I/O ring. And it is responsible to get the results from I/O ring and to deliver them to the corresponding user level task in the domain.

Analogously Domain-0 requires a part to take request from I/O ring and to put the result into I/O ring. For this Domain-0 has back-end device driver. Back-end device driver gets a request from I/O ring and processes it with the existing kernel functions of Xeno Linux in Domain-0. After the processed result is queued in I/O ring, back-end device drivers a notification to the domain which made the request.

Xen itself can be light-weighted because I/O processing is done through existing operating system in Domain-0. However, it makes also Xen hardly have swap mechanism at machine memory level. Thus, the sum of memory which is allocated to all guest systems should be always less than machine memory capacity. Hot-plug memory and ballooning [2], with which guest systems give and take machine memory to and from other guest systems on demand, were proposed for more efficient memory use. It is, however, just a temporary solution because the actual allocated memory size of all guest systems is always less than that of machine memory after all.

2.2 Page Sharing among Guest Systems

VMware ESX server[3] was a pioneer product that aimed to run massive guest systems. Accordingly, reducing memory consumption was directly related to its performance. VMware ESX server achieved much reduction of memory usage by *Content-based page sharing* scheme[4].

After a guest system gets a newly allocated page which is loaded with a certain disk block, Content-based page sharing compares the newly allocated and loaded machine page to all existing pages with MD5 hash function [5]. If there is a page with the identical content, then the mapping table in the guest system is modified so that the corresponding pseudo-physical page in the guest system points to the existing real hardware page instead of newly allocated one. This method needs not alteration of guest operating systems. It can be implemented with modification of only virtual machine manager.

However, executable files or data files stored in disks which are the primary target of Content-based page sharing do not change frequently. Thus, processing read requests for the already loaded pages to be shared is waste of resource. Moreover, a page which is located at the same disk block of a read-only filesystem have same contents naturally and does not require MD5 hash function to

compare the newly allocated page with the existing one of which target block is the same with that of the newly allocated one.

Although it was proposed before VMware ESX server, page sharing in Disco [6] is more efficient than VMware ESX in terms of processing redundant disk read requests. Disco captures DMA requests from guest systems and compares them to the recorded requests that were processed earlier. It becomes possible because Disco dominates all the hardware directly. If there is an existing page containing the requested block, the DMA operation for the request will not be actually started and the guest system which generates the request will share the existing page. This is fairly effective for not only memory saving but reducing read operations also. Intercepting DMA requests requires that virtual machine managers directly handle both hardware and I/O requests from guest systems by itself. Thus, this approach is hard to be straightly adopted in Xen, which handles I/O requests through the proxy guest system.

To achieve the improvement of scalability from all mentioned page sharing schemes, memory overcommitment should be supported by the virtual machine manager. Thus, they are not suitable to Xen which does not provide swapping of physical memory, which is an essential feature for the memory overcommitment, for the virtual systems.

3 Page Sharing for Xen

3.1 Design Overview

In most cases, Domain-U uses a small set of well-known operating systems such as Linux, FreeBSD and Microsoft Windows.In such environment many domains share read-only filesystems that contain operating system and frequently used program files and libraries. Each domain has their own writable filesystems for storing data and temporary files.

In this configuration, multiple pages scattered in different domains mostly happen to contain same disk block. The aim of this paper is to reduce memory usage and disk block read operations through the sharing of the multiple identical pages. This should be done with little overhead and fit to Xen architecture.

Before back-end block device driver processes it, our approach checks the read request whether there exists a machine page containing the target disk block or not. If it can not be found, the read request will be processed in the original path. On the contrary, if there exists the page in machine memory, the read request will not be processed and the page will be returned to front-end block device driver at Domain-U as the result for the request. Without distinction of the requesting domain, machine memory which was allocated to Domain-0 will be used as the page for the read request.

The suggested method quickly and precisely identifies page contents without requiring complex hash functions. It also reduces redundant read operations. The I/O handling in a virtual machine system is much slower than those in native systems [7]. Thus, the suggested scheme is expected to improve system performance as well as save memory usage.

3.2 Work Flow

The original processing flow of a disk read request in Xen is described as follows with Figure 1:

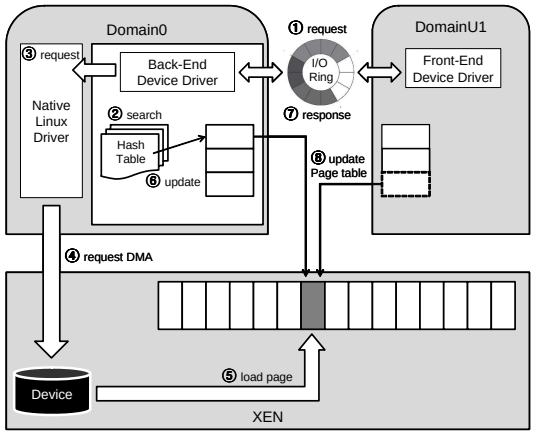

Fig. 1. Procedure for processing I/O requests in the suggested page sharing scheme

1. ①. The requester domain prepares an empty pseudo-physical page and gives access grant and address of the page to I/O ring. The I/O request is also put into I/O ring at this time.
2. ③. Back-end block device driver at Domain-0 maps the machine page mapped from the pseudo-physical page given from the requester domain to a pseudo-physical page in Domain-0. And it calls native Linux block device driver to load the target block into the mapped page.
3. ④. The native Linux driver initiates DMA operation for the request.
4. ⑤. DMA (in real hardware) reads the target block from disk and put it into the target page.
5. ⑦. After the finish notification from DMA, back-end block device driver delivers the result message to I/O ring and signal the requester that the operation is completed.

In our scheme the requesting domain does not give its machine address to Domain-0. On the contrary, Domain-0 provides its machine pages to the domain which requests block read from read-only filesystem. By manipulating mapping table the machine page which originally allocated to Domain-0 can be seen and used by the requester. Domain-0 can change the mapping states of machine pages to pseudo-physical pages by calling *hyper-call*. Hyper-call is function interfaces that connect Domain-0 and Xen hypervisor. In the suggested scheme, the flow of processing read request which is not in the memory is described as follows with Figure 1:

1. ①. ditto
2. ②. Domain-0 analyzes the read request. It searches block number table of existing pages. If it can not find a corresponding page, it allocates a new machine page for the request and maps the machine page address into a pseudo-physical page in Domain-0.
3. ③. ditto
4. ④. ditto
5. ⑤. ditto
6. ⑥. Upon completion of DMA operation, back-end block device driver add the device ID, block number and corresponding machine page address to the block number table.
7. ⑦. ditto
8. ⑧. Front-end block device driver maps the target pseudo-physical page address in ① to the machine page address.

In the suggested scheme, as ⑥ in Figure 1 a table to stores the device ID and block number of loaded pages is added and managed. In case there occurs a read request and the search result for the table returns a page that have the corresponding block, the processing of the read request is done only with manipulating corresponding mapping information. It is described as follows with Figure 1.

1. ①. ditto
2. ②. Domain-0 analyzes the read request. It searches block number table of existing pages. If it can find a corresponding page then it gets the machine address of that page from the table.
3. ⑦. ditto
4. ⑧. ditto

In the suggested scheme constructing of loaded block table, searching the table and allocating machine page for the newly loaded block are all done in Domain-0. Domain-0 is the best for the roles because Domain-0 actually processes all the I/O requests from all domains and controls all domains.

3.3 Data Structures

For the search of existing page contents we designed a data structure as illustrated in Figure 2. In Domain-0, there is a linked list of which element is each device ID. In other words, a device has an element in the list. A minor device [8] in Linux kernel is used for the unit of a device here. In a hard disk a minor device means not the disk itself but the partition in the disk. As a result, each of the elements corresponds to a mounted filesystem respectively. This is called device list.

A linked list is attached to each element of the device list. This is block list. An element in a block list has a block ID and also a machine page address of which a page currently containing the block. All the block elements in a certain

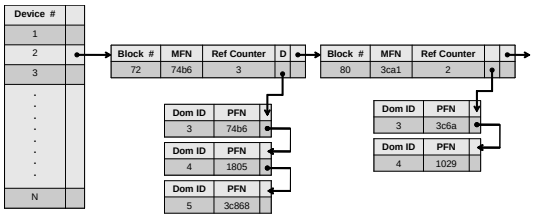

Fig. 2. Data structure for page sharing information

device are gathered in the list which is attached to the device element in device list.

Each element of the block list also has a linked list respectively which is to store sharing information of the block. This is called allocation list. An element in this list has mapping information of a machine page, the pseudo-physical page which logically contains the block content and the domain which participates in the sharing of the machine page. If a domain which participates in the sharing frees a pseudo-physical page which references a machine page, the element for the domain will be removed from the allocation list. If all elements are removed, the corresponding machine page will be freed from Domain-0 also and it can be served for another disk block. A reference counter in an allocation list records the current number of sharing domains for a machine page.

Domain-0 works in with Xen hypervisor to allocate a new machine page, make a share of an existing machine page for other domains and deallocate an existing page without domains in use. On the contrary to data structure managed by Domain-0 in Figure 2, mapping tables are only accessible and writable in Xen hypervisor. Thus, Xen provides hyper-calls, which are analogous to system calls in traditional operating system, to Domain-0 for manipulating the mapping. The suggested method also uses these hyper-calls to manipulate the mapping tables. This approach makes actual sharing operations easier which are just modifying mapping tables.

4 Evaluation

Unfortunately, the suggested method has not been fully implemented yet since the structure of Xen related to the page management is rather complex and we lack of information on it. Thus, we evaluated the suggested scheme by simulation based on the real-world implementation of the described data structures and the functions. The simulation environment is described in Table 1. The target system was Xen 3.0 and Xeno Linux for it.

4.1 I/O Reduction and Page sharing

Actually the amount of saved memory differs not from the sharing mechanism but from the property of workload and number of active domains. Thus, in this

Table 1. Evaluation environment

Virtual Machine Monitor	Xen 3.0
Guest OS	Linux Kernel 2.6.14 (Xeno Linux)
Memory for Sharing in Domain 0	128 MBytes
Memory for each Domain U	64 MBytes
Shared File System	executables and libraries

paper, the evaluation was not for comparison with existing page sharing schemes but for showing example benefits solely from the suggested page sharing.

The workload for the evaluation is comprised of widely-used applications in the real-world. They are randomly started and finished in each domain. The applications for the evaluation are *Open-Office* (Officeware), *Firefox* (Web-browser), *Gimp* (Image editor) and *VLC* (Video player) and the executable file sizes of those applications are respectively 153.7 Mbytes, 27.2 Mbytes, 9.8 Mbytes and 9.1 Mbytes.

Each Domain-U has individual 64 Mbytes memory respectively. Domain-0 has 128 Mbytes memory. All the memory in Domain-0 except the portion for operating system and system programs is used for shared area.

The result was compared with the result under the original Xen system for each evaluation. The amount of shared pages is hard to be expressed easily because it changes continuously as time flows. Thus, the number of sharing which is also same as the number of reduced read operations is used for the comparison.

After an hour of execution, the results in Figure 3 were obtained. Each bar means the actual read requests processed in each environment.

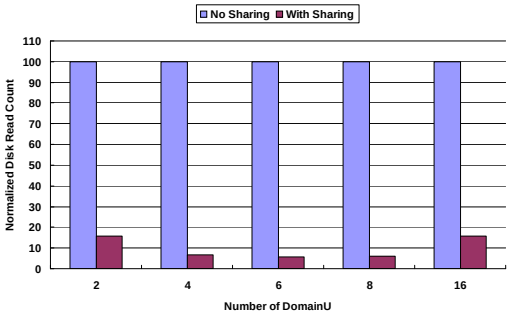

Fig. 3. Number of processed read requests under the suggested scheme

The use of shared area has the effects similar to buffer cache increment which prepares the frequent contents in advance by other domains. Each program was executed multiple times in the evaluation. Thus, the addition of shared area

reduces much read requests. Comparing with the results under no sharing, only about 10% of total read requests are actually processed.

The ratio of actually processed requests tends to decrease as the number of domains in the sharing increases. this tendency, however, changes to the opposite when there exist too many participating domains. With 16 domains, a little increase of the ratio happens. This is due to the fact that the working space exceeds the shared area when it contains many domains. Because each of them runs many applications concurrently and opens various files for the applications, the working set size for the shared filesystem increases as the number of running domains increases. Thus, adequate amount of shared area to the expected working set should be prepared for the best result.

4.2 Analysis of Overhead

To find a identical page in 128 Mbytes of shared area, we searched 32768 elements. If the search is done with well known B+ Tree, 32-order 4-depth tree is enough and the computational overhead for searching the tree is trivial.

To analyze the spatial overhead, we tracked the magnitude of it while the number of shared pages increases. The results show that the size of management structure grows linearly to the increase of shared area. For 128 MBytes of shared area shared with 4 domains, about 700 KBytes of memory is used for the management overhead. This is 0.005% of shared area capacity.

5 Concept of non-Swap Overcommitment

Since Xen does not allow memory overcommitment, the actual improvement of scalability will be not gained, even if the implementation will be completed. To achieve the scalability improvement with the suggested method, Xen should have also memory overcommitment feature in it.

The most intuitive approach for memory overcommitment in virtual machine hypervisors is to employ swap mechanism. To support swap mechanism in virtual machine hypervisors, they should have device drivers for swap devices and filesystem drivers for swap filesystems. However, this policy does not fit the lightweight virtual machine hypervisors.

As a result, to increase the number of running domains without using swap mechanism, the feature is needed which allows the newly starting domain to use the surplus memory from the suggested page sharing method.

However, a blind overcommitment of the surplus memory may cause critical problems. When a pseudo-physical page which shares a machine page is freed, a new machine page should be mapped to the pseudo-physical page. If there is no available machine page due to the over-commitment at this time, the domain which owns the pseudo-physical page will be unable to proceed any more.

Thus, we propose a notion of *permanent share*. In read-only shared filesystem, there are files which should be always loaded in memory during the run-time such

as operating system kernels and core libraries. Sharing for these files is named permanent share.

A permanent share is not being freed until the sharing domain is terminated. Thus, it is safe to provide surplus machine page, which is produced from permanent shares, to newly starting domains. This approach is expected to improve the scalability of virtual machine systems significantly, because modern operating systems and core libraries have quite large executable file sizes.

Categorizing files with the permanent share property can be done transparently by analyzing the using patterns. However, when the categorizing fails, the domain with freed share may be unable to proceed its execution. Thus, accurate judgment of permanent share characteristics is a remaining issue.

6 Conclusion

Consolidating virtual servers into few virtual machine systems enables flexible configuration of cluster or distributed systems. The scalability of the server consolidation, however, is primarily limited by memory size.

This paper suggested a new page sharing design for Xen. It analyzes the block read requests that were forwarded from guest systems to controlling guest system and shares existing pages by modifying mapping tables when there exist identical pages. This scheme was verified with simulation using mock-up implementation.

As a further work, we also suggested a concept model of memory overcommitment without swap by using the suggested page sharing scheme based on permanent sharing notion where we are currently working on.

References

1. Barham, C.P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of vritualization. In: Proceedings of the 19th ACM symposium on Operating systems principles. (2003) 164–177
2. Schopp, J.H., Fraser, K., Silbermann, M.J.: Resizing memory with balloons and hotplug. In: Proceedings of the Linux Symposium. Volume 2. (2006) 313–319
3. Rosenblum, M.: Vmware's virtual platform: A virtual machine monitor for commodity pcs. In: Proceedings of the 11th Hotchips Conference. (1999)
4. Waldspurger, C.A.: Memory resource management in vmware esx server. In: Proceedings of the 5th Symposium on Operating Systems Design and Implementatioe. (December 2002)
5. Rivest, R.L.: The MD5 message-digest algorithm. RFC 1321, MIT, RSA Data Security (April 1992)
6. Bugnion, E., Devine, S., Govil, K., Rosenblum, M.: Disco: Running commodity operating systems on scalable multiprocessors. ACM Transactions on Computer Systems **15**(4) (1997) 412–447
7. Cherkasova, L., Gardner, R.: Measuring cpu overhead for i/o processing in the xen virtual machine monitor. In: Proceedings of USENIX 2005 Annual Technical Confernce. (2005) 387–390
8. Bovet, D.P., Cesati, M.: 13. In: Understanding the Linux Kernel. 3rd edn. O'Reilly (2006) 536–537