Resource Redundancy Elimination by Bridging the Semantic Gap in Virtualized Systems

Inhyeok Kim[#1], Changwoo Min[#2], Young Ik Eom[#3]

[#]*School of Information and Communication Eng., Sungkyunkwan University*
300 Cheoncheon-dong, Jangan-gu, Suwon, Gyeonggi-do 440-746, Korea

[1]kkojiband@ece.skku.ac.kr

[2]multics69@ece.skku.ac.kr

[3]yieom@ece.skku.ac.kr

***Abstract*— Traditional virtualization technologies have been used to provide multiple virtual environments on a physical environment by running multiple guest operating systems. In virtualized systems, deep semantic gaps which cause space and access redundancies of virtual resources exist between the guest and the host. We propose a novel host-based file system that eliminates memory and storage redundancy on the virtualized environments by bridging the semantic gap between the guest and the host. It eliminates the redundant resources in guest and host through host file system sharing. Our experiments show how much the redundancy is eliminated by using the host-based file system. As a result, the performance of the guest file system is improved due to pass-through guest block device layer.**

***Keywords*— Virtualization, Semantic gap, Resource redundancy**

I. INTRODUCTION

The virtualization technologies, such as emulation, para- and full-virtualization, have deep semantic gap between the guest and the host due to the layered approach in the virtualized systems. Such semantic gap may cause several kinds of resource redundancies. Especially, memory and storage redundancies which degrade the utilization that the resources of same contents coexist on the multiple guests limit the scalability of the virtualized systems [1, 2]. In order to reduce memory redundancy, previous studies have proposed the content-based page sharing [3] and the sharing-aware block device [4]. However, they used whole memory scanning or the location on the virtual block device file, not using the guest's logical information because of the semantic gap. We propose a *host-based file system* that eliminates memory and storage redundancies by bridging the semantic gap. As we will show in our experiments, our proposed file system supports efficient page sharing and pass-through virtual block devices. The rest of the paper is organized as follows. In Section 2, we define resource redundancy. Related work is discussed in Section 3. Section 4 describes the proposed scheme in detail. Section 5 presents the evaluation results. Finally, Section 6 concludes the paper.

II. RESOURCE REDUNDANCY & PREEMPTION

The traditional virtualization technologies have supported unmodified guest which recognizes the virtual machine as the real machine like as all resources only are owned by self. These approaches make deep semantic gaps which cause several resource redundancies and preemption. At first, after the guest accesses some type of virtual resources, the host should access the physical resources corresponding to the virtual resources. This situation is called by access redundancy have problem with overlapping accesses to one physical resource. And next, the time-shared resources such as CPU and network device have no space redundancy because the guest uses exclusively those resources normally based on scheduling by usage time. However the space-shared resources which are collaboratively allocated to the guest have some space redundancy by the semantic gap occurring by no communication between the guests. And in similar situation, the resource preemption can be occurred that the guest who has no needs or low priority owns the resource.

III. RELATED WORK

VMware described page sharing technique employed in VMware ESX server [3]. It periodically scans the physical memory of guests to detect page sharing opportunities. However, because of high scanning cost, the scanning interval should be long enough to keep the runtime overhead low. In the other hand, short-live page sharing opportunities cannot be detected due to coarse-grained scanning.

Satori [4] is a sharing-aware block device for efficient page sharing in virtualized environment. It can detect short-live sharing opportunities with low cost. However, in Satori, the efficient page cache sharing can be supported only when guests share the same file system image, because Satori checks sharing opportunities using block location on the shared file system image. So if a block is written once using the copy-on-write technique, the block lose cheap sharing opportunity and is shared using the content-based page sharing scheme since then.

VAMOS [5] optimizes I/O virtualization overhead, which runs some part of middleware on the host to reduce the guest-host switches. It provided new interface between guest and host for the database. It was applied to MySQL. Like this approach, the higher level interface can cut down the number of communication between components, but the interface is specific to the middleware. Therefore, this approach accompanies high development and maintenance cost.

IV. HOST-BASED FILE SYSTEM

We proposed the host-based file system, named as *MyFS* to reduce memory and storage space redundancies and storage access redundancies. Key idea is to eliminate the semantic gap between the guest and the host by using pass-through virtual

block device on the guest. In previous work, the host can only see the raw disk image of guest's virtual block device. However in our proposed approach the host can see same logical structure of the file system used in guest. So, the host-based file system can support file-level page cache sharing using the logical information without expensive memory scanning by the host.

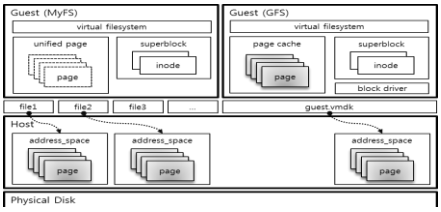

Fig. 1 Host-based file system architecture

As Fig. 1 shows, in the case of the general file system, the guest should mount the file system on virtual block device, but the guest using the host-based file system, do not mount the file system on the guest, instead the guest directly accesses the part of the file system mounted on the host. To achieve this approach, we implemented some specific functions and data structures for using the host file system information to handle file related operations such as file open/close and page cache mapping which are required to the file system by the Linux virtual file system on the guest. The one of important thing which is required by the Linux virtual file system is the inode number which is unique identifier for each file. The general file system on virtual block device has its own inode table, but the host-based file system has no inode table, instead uses file descriptors of the guest process on the host which are acquired to open the file by the requirement of the host-based file system. And next, the page caches are mapped directly to the host page caches using the inode number which is the file descriptor in the host. Through this approach, we can easily eliminate the page caches and storage redundancies between the guest and host, and also the host can support page cache and storage sharing between the guests.

V. Evaluation

We first evaluated the performance improvement through the access redundancy elimination of the host-based file system. The experiment used five real workloads, such as tar and md5sum. The Fig. 2 presents the result that the host-based file system offers about 5~30% better performance than the general file system. Then we measured the memory and IO usage of both cases through reading a 300 MB file. In Fig. 3, the cached size means the host-level page cache size for the virtual block device file and the shared files, and the buffer size means the host-level anonymous page for the virtual RAM of the guest. As you can see, the cached size of both cases was incremented by almost same with 300 MB, but in the case of the buffer size, the general file system used more than the host-based file system. Because the host-based file system does not request the guest-level page cache, instead shares the host-level page cache. However the general file system should allocate the guest-level page cache using own

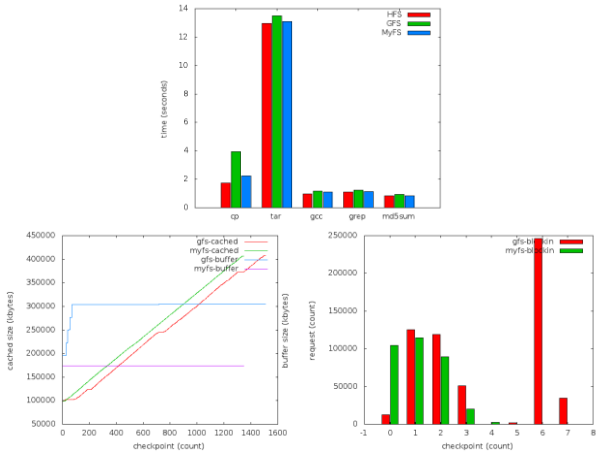

Fig. 2 (a) HFS vs GFS vs MyFS real benchmark
(b) GFS vs MyFS memory usage (c) GFS vs MyFS io usage

virtual RAM. The Fig. 4 shows the result of two consecutive experiments to read a 300 MB file. This experiment shows the host-based file system makes IO requests only at first period, but the general file system makes IO requests at both periods. Because the general file system led to flush many page caches due to memory space redundancy occurred by the guest-level page caches.

VI. Conclusions

We proposed the host-based file system to eliminate memory and storage redundancy on virtualized environments. Our scheme eliminated the semantic gap between the host and guest. So the host and guest share logical file system information, therefore the host can make a decision for resource sharing easier than the guest file system using virtual block device. We showed that the host-based file system have good performance and low memory usage through three experiments.

Acknowledgment

This research was supported by Future-based Technology Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (2010-0020730).

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," Proc. Of 9th ACM Symposium on Operating Systems Principles, 2003.

[2] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Geiger: Monitoring the buffer cache in a virtual machine environment," Proc. Of 12th Architectural Support for Programming Languages and Operating Systems, 2006.

[3] C. A. Waldspurger, "Memory resource management in VMware ESX server," Proc. Of 5th USENIX symposium on Operating System Design and Implementation, 2002.

[4] G. Miło´s, D. G. Murray, S. Hand, and M. A. Fetterman, "Satori: Enlightened page sharing," USENIX Annual Technical Conference, 2009.

[5] A. Gordon, M. Ben-Yehuda, D. Filimonov, and M. Dahan, "VAMOS: Virtualization aware middleware," 3rd Workshop on I/O Virtualization, 2011.