2012 41st International Conference on Parallel Processing Workshops

MemHole: An Efficient Black-Box Approach to Consolidate Memory in Virtualization Platform

Pengfei Zhang, Rui Chu, Huaimin Wang
National Laboratory for Parallel and Distributed Processing, National University of Defense Technology
e-mail: {zpfalpc23, ruichu}@gmail.com, whm_w@163.com

***Abstract*—As an important aspect of the hardware resource consolidation in virtualization environment, memory consolidation and over-commitment has been motivated by the increasing elastic computing cloud platform. The most popular consolidation technology, Memory Balloon [2], might introduce serious performance penalty with thrashing when guest memory usage changes dramatically. In order to overcome the drawback of Memory Balloon and guarantee the system performance, we propose MemHole, which provides more guest physical memory than really allocated and makes the corresponding host physical mapping undetermined until accessed, to reduce the thrashing of guest memory paging. The prototype of MemHole has been implemented in Xen [3] platform and the efficiency has been shown in the preliminary evaluation.**

Keywords- Virtualization Platform; Memory Consolidation; Balloon; Black-box

I. INTRODUCTION

The trend of IaaS cloud platform intensively calls for elastic resource allocation. As an important solution, the consolidation and over-commitment of hardware resources in virtualization platform has been thoroughly studied. Specially, memory consolidation is a classic problem, and there were many mechanisms to dynamically adjust the memory allocation among virtual machines (VMs) in a virtualization platform, such as Content-based Page Sharing [1], Demand swapping [2], as well as one of the most widely used technology, Memory Balloon.

Memory Balloon is a gray-box approach. There exists a driver program named balloon driver running in guest operating system. The driver can dynamically allocate/reclaim heap memory to decrease/increase the memory footprint of this VM. The balloon approach has been implemented in most current virtualization platforms, such as VMware ESX Server [2] and Xen [3]. However, it also has obvious drawbacks. For example, in our previous practice, there exists a scheduler program running in the privileged management VM (called Domain0 in Xen), to collect the memory usage and demand of each VM and decides whether reclaim memory pages by the balloon driver to schedule the physical memory allocation. If the memory usage of a VM and the scheduler cannot proceed rapidly enough to meet the VM's demand changes dramatically, the memory paging mechanism in guest operating system will be triggered to swap out and swap in memory pages frequently. Such thrashing usually results in serious performance penalty. In fact, even though the scheduler can allocate enough memory to the VM, the VM have no awareness because of the semantic gap between guest operating system and hosted hypervisor.

We propose MemHole to compensate the thrashing of such unnecessary memory paging in Balloon approach. MemHole is a black-box mechanism, in which the VMs might have the view of more memory than really assigned to. In another words, some part of the memory pages are allocated but not committed. With MemHole, system performance can be guaranteed in the worst situation because of extended free memory view in guest VM.

We have implemented the prototype of MemHole in Xen virtualization platform. The preliminary performance evaluation shows that MemHole can consolidate and adjust the memory footprint of guest VMs and effectively guarantee the system performance even in situation when memory usage increases fast.

II. DESIGN

In most virtualization platforms, a memory page might have three kinds of address as follows:

- Guest Virtual Address (GVA): the virtual address at the view of VMs and it can be accessed by processes.
- Guest Physical Address (GPA): the physical address at the view of guest operating system kernel.
- Host Physical Address (HPA): the physical address at the view of the host, and equals to the real hardware address of a memory page.

A virtualization platform without Memory Balloon has two kinds of memory page, denoted as *used* and *available*, which indicate the memory has been used and available for the processes in VM. Balloon memory represents the memory occupied by the balloon driver in guest but not committed by host. The associated GVA, GPA and HPA of the memory types are illustrated in Table I.

TABLE I. MEMORY ADDRESSES FOR DIFFERENT MEMORY TYPES

	used	**available**	**balloon**	**hollow**
GVA	Yes	No	Yes	No
GPA	Yes	Yes	Yes	Yes
HPA	Yes	Yes	No	No

Different from the types of *used*, *available* and *balloon*, another kind of memory page in MemHole approach, named *hollow memory*, to introduce the memory pages which has GPA, but neither GVA nor HPA**,** as depicted in Table I. Even though hollow memory has not been committed in the hosted hypervisor, it could be readable/writable by the corresponding VM, because of the associated GPA makes

1530-2016/12 $26.00 © 2012 IEEE
DOI 10.1109/ICPPW.2012.92

the guest operating system still identify the memory as available. When the hollow memory is to be allocated from the guest operating system to a process, the operating system will create GVA for it. This action will be trapped to VMM, and as a result, VMM will commit this hollow memory as needed. Based on this on-demand commitment property, MemHole mechanism has the capability to present more memory to VM than really assigned, to restrain the heavy but unnecessary memory paging in guest operating system.

In order to implement the mechanism of hollow memory as described above, we exploit the virtualization support and multi-level memory addressing mechanism, VT-x and Extended Page Table (EPT), built in Intel CPUs. When a guest operating system is intending to use the hollow memory, the action of creating page table entry for the page will be trapped by a VMExit, which guarantees that we have the chance to allocate physical page for that in VMM, including building the mapping in the EPT, granting the access authority, and transforming the memory type to available. After that, CPU resumes to VM context by a VMEntry and re-executes the instruction.

The hollow memory introduces the benefits as follow:

- **Rapid response**: Different from *Balloon*, the hollow memory is committed at the time when it is actually used by VM for the first time. As long as the host has sufficient free memory, the commitment can always be completed successfully for the VM demands.
- **Synchronous:** the CPU's virtualization features guarantee VM be suspended and every hollow memory access will be synchronously taken over by host. After the execution recovers, the context returns to VM with no awareness of the pause.

As analyzed above, the size of hollow memory will gradually decrease during the VM running since the usage of hollow memory area will fill up the holes. Lower hollow memory size in each VM could use up the free memory in host, on the other hand, too much hollow memory might also bring extra overhead because of the frequent VMExit and the corresponding procedure to allocate HPA in VMM. In MemHole, we also design a monitor process in Domain0 named *coordinator*, to dynamically balance the hollow memory of each guest. The coordinator constantly watches on free memory and hollow memory of each guest VM, as well as remaining physical memory in host. Then, the coordinator uses the information to decide whether to increases/decrease each VM's hollow memory and also the memory size to be transformed.

The coordinator is designed to adjust the amount of hollow memory pages in three follow situations:

- **Global Enough**: This situation means that the free memory size of all VMs is larger than a preset value.
- **Global Shortage**: This situation happens when the free memory size of all VMs is smaller than a threshold memory size.
- **Local Enough:** If the free memory of a particular VM is larger than another preset value, coordinator will reclaim some amount of available memory and convert it to hollow memory for this VM.

III. Implementation and Evaluation

We have implemented MemHole in Xen. The prototype leverages the inter-domain memory sharing component [4] to simplify the implementation of hollow memory, and also take the program localization feature into consideration to reduce VMExit overhead on the hollow memory access route.

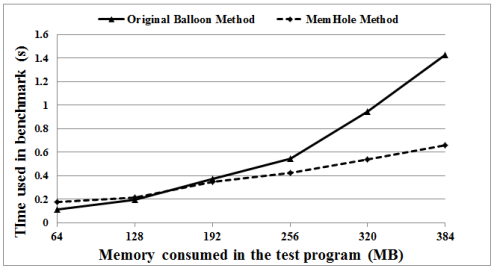

Figure 1. Execution time of test program with different memory usage compared with orginal balloon method

To evaluate the performance of MemHole, we have implemented a test program which can consume memory in VM and show the performance indicator under different memory usage pressures. The result depicted in Figure I shows that in the implemented prototype, the MemHole Method does improve the performance under most of the pressures, compared with the original Balloon method.

IV. Conclusion

The study shows that MemHole can efficiently consolidate memory footprint in virtualization platform, avoiding memory thrashing introduced by Balloon approach. This work is still in progress now. We will test the effectiveness using more benchmark suites such as DaCapo [5], optimize the implementation for further performance and improve the robustness and fault tolerance in our further work.

Acknowledgment

The work was supported by the National Natural Science Foundation of China under Grant No.61003076， the National Basic Research Program of China (973) under Grant No.2011CB302601 and the National High-tech R&D Program of China (863) under Grant No.2011AA01A202.

References

[1] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat. Difference engine: Harnessing memory redundancy in virtual machines.In 8th USENIX symposium on Operating System Design and Implementation, 2008.

[2] C.A.Waldspurger, Memory Resource Management in VMware ESX Server, In OSDI' 02: Proceedings of the 5th symposium on Operating systems design and implementation, pages 181 – 194, 2002.

[3] 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. In Proceedings of the 19th ACM Symposium on Operating Systems Principles, 2003.

[4] J. F. Kloster, J. Kristensen and A. Mejlholm, "Efficient Memory Sharing in the Xen Virtual Machine Monitor", Technical Report. Department of Computer Science, Aalborg University, Jan 2006.

[5] DaCapo Benchmarks Home Page. http://dacapobench.org/