World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:6, No:10, 2012

Evaluating the Effectiveness of Memory Overcommit Techniques on KVM-based Hosting Platform

Chin-Hung Li

***Abstract*—**Determining how many virtual machines a Linux host could run can be a challenge. One of tough missions is to find the balance among performance, density and usability. Now KVM hypervisor has become the most popular open source full virtualization solution. It supports several ways of running guests with more memory than host really has. Due to large differences between minimum and maximum guest memory requirements, this paper presents initial results on same-page merging, ballooning and live migration techniques that aims at optimum memory usage on KVM-based cloud platform. Given the design of initial experiments, the results data is worth reference for system administrators. The results from these experiments concluded that each method offers different reliability tradeoff.

***Keywords*—**Kernel-based Virtual Machine, Overcommit, Virtualization.

I. Introduction

ALTHOUGH virtualization brings flexibility to OS arrangement, the continual need to balance physical resources to workload demand that makes capacity planning more important. One of the biggest challenges about resource overcommit [1] is the potential performance decrease within guests or hosts.

However, the on-line memory consumption of a system is largely application-dependent. The challenge of driving higher system utilization by consolidating workloads is managing the complexity of the consolidation. Generally speaking, Linux user guide would suggest that swapping is supposed to be the last resort. Technologies such as Kernel Same-page Merging (KSM) [2] and memory ballooning [3] are different ways to maneuver memory overcommit. Both same-page sharing and ballooning outperform swapping.

In this paper we have arranged three different scenarios to utilize guest's memory. The experiments in this paper were to explore the current approaches to memory management in cloud platform. In the last section, we discussed the effect and affect of present initial results.

II. Fundamental

A. Server Virtualization

Most of virtualization projects are focusing on server consolidation [4]. That is because server consolidation brings impressive financial benefits (e.g., reduction of maintenance costs, power consumption savings and floor space savings). Although there are many good reasons to create a virtual infrastructure, the degree of savings depends on the configuration to overcommit host resources, such as CPU, memory, disk and network bandwidth. Many best practice guides [5] suggest, for example, using sparsely disk image files for overcommitting storage, preparing enough host swap space for overcommitting memory, etc.

As of this writing, Linux KVM hypervisor shows efficiently processor resource sharing [6] among guests. However, memory overcommitment is still a challenge for virtual machine rental service providers who deployed KVM. Unlike a regular process, each virtual machine runs an operating system on it. Thus virtual machine has greater chance requesting more memory rather than an application.

B. KVM Hypervisor

KVM stands for Kernel-based Virtual Machine [7] which means a full virtualization solution. KVM is for Linux which is working on processors that have capabilities related to virtualization. KVM virtual machine implementation has two loadable kernel modules. One is "kvm.ko" that handles the main virtualization function and another is the processor specific module, "kvm_intel.ko" or "kvm_amd.ko". In addition, it also needs a modified QEMU to emulate all peripherals for virtual machines.

The philosophy of KVM designation is re-use the original Linux source code as much as possible. Therefore, KVM brings a very light-weight integration and driver compatibility. Now it has become a very popular cloud hosting solution. Fig. 1 presents the components of a typical KVM installation. As you can see that each KVM VM can be treated as a user space process in Linux, all the memory optimization settings apply.

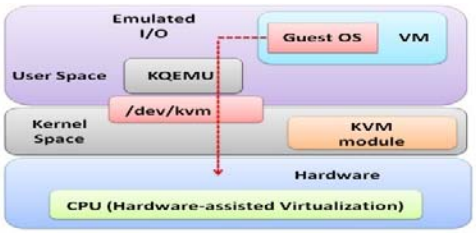

Fig. 1 Linux KVM architecture

C. Memory Ballooning

The simplest way to define the total free memory one KVM guest can use is to allocate fix size amount in template file. But fix memory allotment for VMs is not an efficient way to use limited memory. When the host is running low on memory but

C. H. Li is with the National Center for High-Performance Computing, Tainan 74147 Taiwan (phone: +886-6-505-0940; fax: +886-6-505-5909; e-mail: OscarLi@nchc.narl.org.tw).

VM is not running out its memory, there should be a better way for host to allocate unused memory of guests.

Ballooning technique is about memory reclamation of a virtual machine as if it had been configured with less memory. When host needs extra free memory, its hypervisor will send a request to the guest OS to return some memory back to the underlying host. With this technique, the system administrator can instruct the target VM to release some of its memory so that it can be used for other purpose.

Ballooning essentially is a cooperative operation between the guest driver and the hypervisor. KVM ballooning is achieved via one of para-virtualized device driver (virtio_balloon). As shown in Fig. 2, it depicts how ballooning works. The size of guest's balloon part memory is controlled by the virtio_balloon driver. These pinned memory pages will no longer available to guest but only host can reclaim it then. Once balloon memory becomes available again, the hypervisor will return memory and instruct its guest OS to deflate the balloon part. Finally relinquish the guest memory pressure.

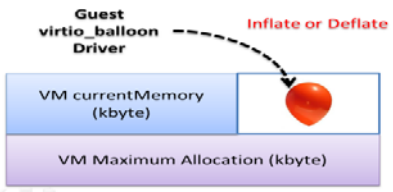

Fig. 2 Memory ballooning concept

D.Kernel Same-page merging

Kernel Same-page Merging (KSM) is a new Linux kernel feature [8]. Essentially, Linux KSM is sort of virtual memory de-duplication. This concept is represented on Fig. 3. KSM is initiated by host ksmd service which looks for host pages that contain identical content. When KSM finds identical pages, those pages will be marked sharable and merged as read-only. If later a process requests to modify one of these pages from registered memory, KSM will create a copy-on-write page.

Since bit-by-bit memory pages comparison is a CPU-intensive task. Memory scanning frequency need to meet the workload demand, otherwise it will result in high CPU load.

According to Linux kernel documents, a high ratio of pages_sharing to pages_shared indicates good effectiveness. If mission is running VMs as more as possible, keep ksmd and ksmtuned services up and running is a way to save memory. Despite its advantages, KSM has page size restriction. However, for systems that run the same applications on a lot of homogeneous guests, KSM is theoretically helpful for memory saving.

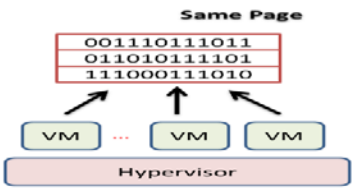

Fig. 3 KSM concept

E. Live Migration

An easy way to system load balancing is to move out some processes to another host. Generally speaking, migration would be considered a preferable way to leverage whole system's workload. Technical glitches are bound to happen in a server room. Services go down all the time. Migration can help those system administrators to avoid the down time of their critical services. Fig. 4 illustrates the basic architecture ready for VM live migration.

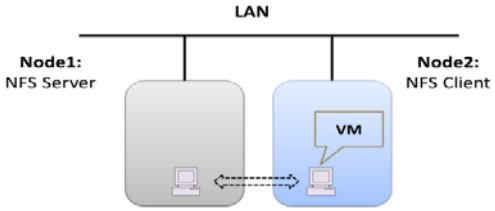

Fig. 4 Scenario of virtual machine live migration

Many private cloud middleware come with a dedicated scheduler for virtual machine placement to meet different goals. Like OpenNebula's built-in scheduler [9]-[10], mm_sched, it can be configured using packing or stripping policy for placing its VMs in host pool. While adopting the packing policy, administrators need to use possible optimization approaches to maximize the virtual machine density.

III. Experiment

In this section, we try to evaluate the suitability of these memory overcommit techniques for deploying KVM-based virtual machine. To understand the impact, we deployed the above optimization mechanisms on real hosts and ran real world workloads on guests to evaluate the potential advantages and disadvantages.

A. Environment

Experimental tests were conducted on 2 hosts. Each host has 4 Intel i7 CPU cores with Intel VT-x support and 12GB of DDR3 RAM. Since KSM function now is directly supported by latest Linux kernel, we installed 64-bit CentOS 6.0 as host operating system.

We evaluated the effectiveness of current memory optimization techniques (ballooning, same-page merging and migration) by booting guests and running virtualized workloads. And we also evaluated the potential side effect while simultaneously using ballooning and same-page merging.

In order to keep track of the memory variations, we recorded the hosts and guests memory usage every two seconds. In addition to Linux guest, we also arranged Windows guest to compare effectiveness during same-page merging test. These guests were default assigned 2048 MB memory.

TABLE I
HOST MEMORY STATE AFTER GUEST BOOT

Host Used Memory (MB)	1VM	2VM	3VM	4VM	5VM	6VM
Linux Guest	981	1360	1957	2448	2930	3414
Windows Guest	2785	4859	6940	9017	11094	11751

Guest Memory Current:2048 MB; *Max:2048 MB*

Table I shows the initial host memory usage after guests OS boot. These values are host memory usage while not running user space program on guests. Notice that after booting 6 Windows guests, this 12 GB memory geared host started using its swap memory.

B. Virtualized Workload

In order to know the impact of running user space applications on guests while deploying memory optimization mechanisms. Here, we had prepared a simple code which will statically allocate 4 GB memory during execution. Fig. 5 shows this code snippet. We used it as a guest application in the following experiments.

```
C
C     This program is for test large memory (> 4GB)
C     Created by Chau-Yi Chou on 20 May 2011.
C
      parameter(n=1000, nn=n*n*n)
C     static locate around 8 GB RAM
C      real*8 a(nn)
C     static locate around 4 GB RAM
      real*8 a(nn/2)
      do 100 i=1, nn
        a(i)=12
100   continue
      write(*,*) "a(100)= 1st step!", a(100)
      do 200 i=1, nn
        a(i)=a(i)+i
200   continue
      write(*,*) "a(100)= 2st step!", a(100)
      end
```

Fig. 5 User space program code snippet

C. Deployment

In this paper, we decided to arrange three kinds of experiments to evaluate the pros and cons while deploying memory overcommit techniques.

Experiment (1)

Both Linux and Windows are popular operating systems now. Cloud providers might have lots of chances putting Linux and Windows guests together on a host. Therefore, we arranged three tests to check how much memory KSM could actually save on a host. The first step is to verify the effect of ksmd service. We also deployed homogeneous and heterogeneous guests as separate tests to see which type of guest OS is in favor of same-page merging technique.

Experiment (2)

To check out the possible factor interfering KSM performance, we arranged guests with different memory capacity on a host for testing. And then we simultaneously executed our Fortran program once on each guest. Since the program will consume about 4 GB memory at a time, it is convenient to monitor the host swap change to verify the feasibility of KSM. In this test, we also add ballooning mechanism while guest user space program running.

Experiment (3)

Shared network storage is pre-requisite for virtual machine live migration. We prepared two hosts: one's role is NFS server and another is NFS client. The NFS server node placed a Linux guest in the beginning. We use this test to see the response time of memory variation during live migration.

IV. RESULTS AND ANALYSIS

A. Performance Evaluation of Virtualization Workloads

TABLE II
EXPERIMENT 1-1
KSM SERVICE EFFECTIVENESS

Host Used Memory (MB)	0VM	1VM	2VM	3VM	4VM	5VM	6VM
KSM disabled	677	1180	1665	2150	2637	3119	3606
KSM enabled	684	1163	1647	2135	2618	3100	3582

KSM_SLEEP_MSEC=10

TABLE III
EXPERIMENT 1-2
HOMOGENEOUS GUESTS PERFORMANCE

	Host Pages	1VM	2VM	3VM	4VM	5VM	6VM
Linux Guest	sharing	42	124	208	292	376	458
	shared	23	44	44	44	44	44
	unshared	24	6	9	12	15	18
Windows Guest	sharing	30	105	180	256	330	*424394
	shared	14	45	45	45	46	46
	unshared	34	6	9	12	15	*85

KSM_SLEEP_MSEC=10

TABLE IV
EXPERIMENT 1-3
HETEROGENEOUS GUESTS PERFORMANCE

	Host Pages	2VM	4VM	6VM
Half Linux Guest and half Windows Guest	sharing	103	253	*398794
	shared	45	45	45

KSM_SLEEP_MSEC=10

The first test results listed in Table II manifests that activating KSM service did not guarantee huge memory saving. Table III shows the performance comparison between launching Linux guests and Windows guests under kernel same-page merging technique. The ratio of pages_sharing/pages_shared steadily gets higher when placing more similar guests. On the other hand, increasing similar guests results in a proportional increase in host unshared pages.

From Table III, we found that running Linux guests on a host will increase more memory saving than running Windows guests. And KSM works well while host not running out of free memory. After launching 6 Windows guests, there is a shortage of free memory for host. Consequently, a very a weird

pages_sharing value calculated when the host began swapping out pages.

Comparing Table III and Table IV, it seeks to capture the fact that KSM efficiency drops while mixing Linux guest with Windows guest.

TABLE V
EXPERIMENT 2-1
KSM WORKING WITH DIFFERENT MEMORY CAPACITY GUEST

Guest Memory	Host Pages	1VM	2VM	2VM App	Run
Current:6144 MB; *Max:6144 MB*	sharing	40	124	290515	
	shared	23	44	64	

Guest Memory	Host Pages	1VM	2VM	2VM App	Run
Current:4096 MB; *Max:6144 MB*	sharing	40	19694	18839	
	shared	23	1097	414	

Guest Memory	Host Pages	1VM	2VM	2VM App	Run
Current:3072 MB; *Max:6144 MB*	sharing	40	81347	80462	
	shared	23	1215	590	

From Table V, KSM efficiency shows as expected when all guests' memory usage did not exceed their physical host's free capacity. Unfortunately once guests start executing memory eating programs, the host will be forced to page out some of its memory. The same thing happened again. The number of host's sharing memory pages sudden rises and then the number of host's used shared memory pages plummets by half.

TABLE VI
EXPERIMENT 2-2
KSM AND BALLOONING WORKING TOGETHER

Guest Memory	Host Pages	1VM	2VM	2VM Run App	2VM Run App with Ballooning
Current:5120 MB; *Max:6144 MB*	sharing	40	20690	22731	37674
	shared	23	1111	554	1543

From Table VI, it proves that ballooning and KSM should not deploy at the same time. Likewise KSM still works well as long as host did not do any memory swapping. Running more identical guest application on similar guests increases KSM efficiency. But when balloon driver starts shrinking guest's total memory, even the host did not start swapping, the ratio of pages_sharing/pages_shared declines.

TABLE VII
LIVE MIGRATION EFFICIENCY

2 Nodes Connected over Gigabit Ethernet	Before Migration	After Migration
Host-1 (with 1 Guest launched)	Total 12038 MB	Total 12038 MB
	Used 1359 MB	Used 1213 MB
	Free 10678 MB	Free 10824 MB
Host-2	Total 12038 MB	Total 12038 MB
	Used 651 MB	Used 786 MB
	Free 11386 MB	Free 11251 MB

Host OS: CentOS 6 x86_64
Guest OS: Fedora Core 4 x86_64

It is a comfort to see that hosts' memory variation become explicit during live migration. According to Table VII, host will quickly release both CPU load and memory pressure while manually moving one virtual machine to another host.

V. CONCLUSION AND FUTURE WORK

The experiment results showed that KSM mechanism works well only under certain conditions. During experimental processes, we found that KSM is prone to be interrupted and becomes invalid while directly running any memory consuming application on host. For example, copying a huge file or enforcing hibernation on host, these memory-related manipulations will not be encouraged for memory optimization.

And it is worth to notice that ballooning will meddle in the operation of KSM. Mixed using these two methods shows no good results. Memory ballooning might be an immediate way to squeeze memory from guests. But only the guest itself knows correctly which pages are shared and which are not. The ability to more granular control of ballooning will require advanced communication channel between the host and its guests.

The final experiment showed that live migration method has positive effect to alleviate host's memory load. Live migration would be a reliable method to leverage the virtual workloads in the cloud.

Server virtualization can deliver significant tangible benefits. But resource overcommitting degree is crucial to leverage the benefits. This paper explains how we evaluated the current memory overcommit mechanism Linux provided. All the results showed that KSM yields slight improvement. Once the host memory is not sufficient, KSM does not scale out and function normally. Given high capacity hosts (multi-cores, big memory and high bandwidth) would be a more pragmatic way when facing shortage. Migration virtual machine to another free host currently may affect some guest's functionality. For example, if you're running a VM hosting web or database server. Live migration has more issues need to be considered. Besides, a host pool is essential for on-premises deployment.

While current results helps system administrators manage VMs' memory allocation, it needs to be refined to meet different goals for different users. Tests on latest Linux resource management features such as Cgroups [11] will be also our future work.

REFERENCES

[1] Manage resources on overcommitted KVM hosts, http://www.ibm.com/developerworks/linux/library/l-overcommit-kvm-resources/.
[2] A. Arcangeli, I. Eidus and C. Wright, "Increasing memory density by using KSM," Available at: http://www.kernel.org/doc/ols/2009/ols2009-pages-19-28.pdf.
[3] J. H. Schopp, K. Fraser and M. J. Silbermann, "Resizing Memory With Balloons and Hotplug," Available at: http://www.kernel.org/doc/ols/2006/ols2006v2-pages-313-320.pdf.
[4] R. Rose, "Survey of System Virtualization Techniques," Available at: http://www. robertwrose.com/vita/rose-virtualization.pdf, 2004.
[5] Kernel Virtual Machine (KVM) Best practice for KVM, Available at: http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaat/liaatbestpractices_pdf.pdf.

[6] L. Nussbaum, O. Mornard, F. Anhalt, J. P. Gelas, "Linux-based virtualization for HPC clusters," Available at: http://www.loria.fr/~lnussbau/files/linux-virtualization-mls09.pdf
[7] Kernel-based Virtual Machine, http://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine/
[8] Neil Smyth, *Red Hat Enterprise Linux 6 Essentials*, 2010, ch. 21.
[9] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Capacity Leasing in Cloud System using the OpenNebula Engine," *Cloud Computing and Applications, 2008*. Chicago, Illinois, USA.
[10] P. Sempolinski and D. Thain, "A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus," in *Proc. CloudCom'*2010, pp.417–426.
[11] S. Seyfried, "Resource Management in Linux with Control Groups," Available at: http://www.linux-kongress.org/2010/slides/seyfried-cgroups-linux-kongress-2010-presentation.pdf.