Kernel Same-Page Merging in KVM

Deepak Maniyani
UFID: 9399-4924
CISE Department
deepak.maniyani@ufl.edu

***Abstract*—One of the most popular applications of virtualization technologies is its ability to consolidate multiple virtual servers inside a single physical hardware. When it comes to increasing the number of virtual machines inside a server, main memory becomes the primary bottleneck. With the widely spread usage of virtualization technologies, various research efforts have addressed these issue and it has resulted in interesting techniques that have in fact made virtualization more robust and efficient. Memory deduplication is one such solutions, where in we try to remove multiple copies of the data across the memory and reduce the effective memory usage.**

Kernel based virtual machine (KVM) is an open source virtualization technology. It was originally developed only for machines running linux on a x86 hardware. Kernel Same Page Merging (KSM) is used by KVM hypervisor to share the identical pages in memory among the guest virtual machines to avoid memory duplication. Although KSM was created only for handling memory duplication in KVMs, later it was found effective in most of the general purpose linux operating system usage, so it was merged into the linux mainline

***Index Terms*—KVM, KSM, Memory Deduplication, Benchmarks, Memory Usage, CPU Utilization**

I. Problem Statement

In this paper, we discuss the core idea behind the Content Based Sharing technique and the work cycle of KSM. We also discuss the major sources of overhead caused by KSM. The primary goal of the project is to capture the KSM metrics in multiple scenarios like virtual machines with and without same operating system kernel and evaluate and discuss the results. As Arcangeli et al. pointed out [1] the major negative impact of having KSM is the higher CPU utilization. We also capture CPU utilization metrics during the experiments to analyse the overhead costed by the memory duplication algorithm.

II. Introduction

KVM is a linux subsystem that allows user to create multiple virtual machines inside the linux operating system. KVM is implemented using the hardware extensions provided by the x86 processors for virtualization. So it can be enabled only in processors with virtualization extension support.

A. KVM Architecture

KVM creates virtual machine inside the linux kernel by simply opening a device node in */dev/kvm*. This nodes will assign separate memory addresses space for each guest machines. KVM introduces a new mode called guest mode in addition to the existing user mode and kernel mode in linux. When the guest operating sytem runs, it runs in user mode and instruct the kernel in kernel mode to run the instructions on its behalf. Kernel instructs the hardware to change the virtualization mode and wait for interrupts and signals to switch back. All the x86 architectures have the concept of virtual memory where the actual physical address is hidden by a virtual address. The translation is carried out using hardware unit called MMU. This a single layer translation of memory. But in case of the virtual machines we require a two layer of address translation: guest virtual address $\rightarrow$ guest physical address and guest physical address $\rightarrow$ host physical address. There may or may not be hardware support for this two layered translation by extensions. KVM makes no assumption of hardware prerequisite for this. So KVM makes use of the shadow page table. Initially the the shadow page table will be empty. When guest access memory the shadow page table gets filled. But the guest memory writes has to be trapped in order keep the guest memory mapping and the shadow page table synchronized. KVM adds two specific module to the linux. 'kvm.ko' module provides the virtualization infrastructure. Unlike the core processor architecture Intel and AMD didn't really follow any specific cross industry standard for designing hardware virtualization extension. So there is one more module which is specific to the processors, either 'kvm-intel.ko' or 'kvm-amd.ko'.

B. QEMU

It is basically a software based hardware-emulator, capable of emulating cross architecture CPU instruction at software level. Hence it also come with the bottleneck that It can only execute instructions line by line. KVM by itself can not create guest machines. It utilizes QEMU to create virtual machine. Since not all the instruction from the guest machine need to be emulated. Some of them can directly be executed on CPU, even in VMX non-root mode. So KVM has the intelligence to interpret which instruction needs be emulated and which need not. In another way we can observe KVM as a accelerator for QEMU hypervisor. QEMU is basically a user process. As shown in the Figure 1., it communicates with KVM via the directory */dev/kvm*. A seperate QEMU thread is created for each of the guest machines.

C. Content Based Sharing

Memory over-committing is one of the prominent techniques used to consolidate more number of virtual machines in a hardware. Memory over-committing can be implemented with the help of 'demand paging', where in we swap the

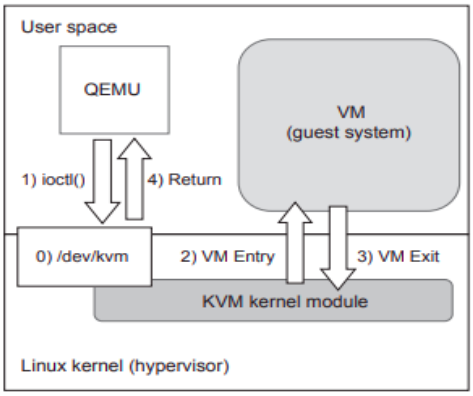

Figure 1: KVM/QEMU architecture, source:[5]

main memory pages to disk in a anonymous way with respect to the guest machines. A naive demand paging can result in pathological memory traffic in the system. Double paging is example for one such anomaly. In double paging, the VMM swaps a page to the disk anonymously to guest operating system. Now say, guest operating system memory policy triggers a page swap for that particular memory, the page from disk has to be brought to the memory just to send it back to the disk. To overcome such anomalies, we use techniques like ballooning, which forces the guest kernel to swap certain amount of memory to disk and free up some space for VMM's usage. Ballooning has been a effective solution in minimizing the negative impacts posed by demand paging.

Here we discuss one more technique that has been proved valuable in tool for over committing memory by increasing the memory density in the system. If there are multiple pages with the same content, then all the applications can refer to a single page instead of multiple pages. This is the core idea behind memory deduplication. The implementations of memory deduplication technique can be categorized as in-band and out of band methods. In band methods scans the memory content in the disk traffic, in in contrast to out of band, in which the pages are periodically scanned in memory to detect identical content.

III. Background

A. KSM

Kernel Same Page Merging or Kernel Shared Memory is memory deduplication application used in linux kernel. The primary target of the KSM is to increase the memory density in the RAM. KSM scans through the memory to find memory pages with identical content and share them across to free up more space. Free space can be used to run more applications or in case of a VMM more more virtual machines can be run. KSM is a out of band method. Once KSM identifies set of pages with the same contents, it retain only one of them and deletes and frees up memory in all the other spaces. Now all the application point to a single page. KSM will mark the page as read only. If any of the application tries to write something to pages, it create a copy of the pages then proceeds with the writing. This techniques is called as Copy on Write.

As we discussed above one of the functionality of the KSM is to scan through the memory pages. But if KSM scans each and every page in the memory it would be detrimental. Not only in terms of the CPU utilization, but also the amount of book keeping needed, will cover a significant amount of the main memory. So KSM has to be intelligent enough to scan the areas where the probability of finding page which can be shared with another process is more.

Once the KSM scans the page it compute hash for the page and store it a metadata page in main memory. If in the future scans if we get a matching hash the new will compute hash for the original page again to check whether the content is still same or changed. If it is changed then it is marked as a volatile page and will be skipped in the following scans.

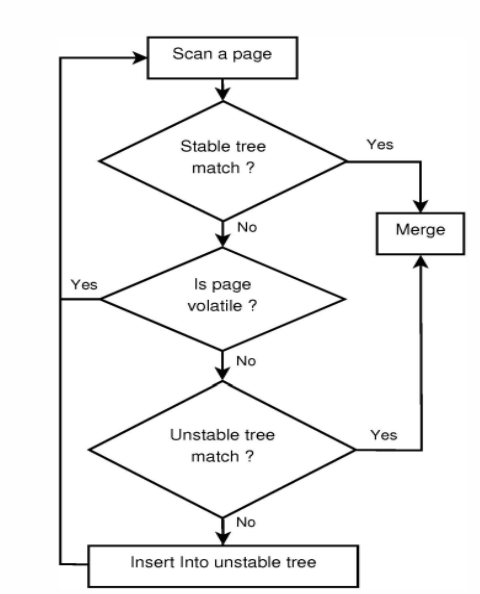

Figure 2: KSM scan cycle, source:[2]

In order to help KSM to determine the virtual ares in the memory which are more likely to have shareable contents, it exposes a set of APIs, which can be accessed by processes to register pages as candidates for memory scan.

B. KSM tuning via sysfs

KSM's operation can be controlled via sysfs at */sys/kernel/mm/ksm*. Following are the file system pages available under the directory:

- run: Writing '1' will start the KSM process, '0' will stop the KSM process, but it will not duplicate the pages which are already merged. Writing '2' will terminate the KSM thread and duplicate the shared pages
- sleep: This specifies the time interval in s between the sleep wake cycle of the KSM thread
- pages_to_scan: This specifies the number of pages to be scanned after the thread wakes up from a sleep cycle.

Lesser the sleep time and more the pages_to_scan, it will increase amount of page scan rate of the KSM. Generally aggressive scanning will result in increased CPU utilization[2].

C. KSM Operation

KSM is a module in linux which when enabled create a thread which scans a specified range of virtual addresses for content based sharing. The address range is provided to the thread by the processes which registers themselves to the KSM. KSM can look for shared pages within a guest virtual machine or among multiple guest machines as well.

KSM algorithm makes use of two red-black trees called as stable and unstable trees[1]. Stable pages contains list of references for the pages which are currently being shared and the unstable tree contains list of pages which has the potential to be shared but currently no match is found. Every time KSM scans a new page, it checks whether it has match in the stable tree, if match found then the reference is updated and the duplicate page is freed. Otherwise it is marked as volatile page and gets added to the unstable tree. When a match is found KSM will calculate the hash again and if the page content is found to be changed, then also it is marked as volatile.

KSM is definitely does a essential memory deduplication and increase the memory density. But this process adds its own overhead to the system. The overhead caused by the KSM can be analyzed in three different levels as follows[2]:

1) Scan and Merge Phase: To update the stable and unstable tree KSM perform a scan of the memory which involves computing of the hashes, comparison of the existing hashes etc. This overhead is directly proportional to the range addresses registered to KSM thread buy the processors. But the merge overhead depends on how many pages are actually can be shared among processes. Merge requires updation of Page table Entry which adds to the overhead
2) MMU Notification: In virtual machines monitors, various processes subscribe to receive an notification from MMU for any change in page table. This is because for complete memory virtualization, a software emulation might be needed, where a shadow page table is constructed from emulating the changes in the Page Table. So Whenever merge happens, it can potentially trigger a chain of events causing more overheads
3) Copy On Write Breaks: As we discussed in the earlier section, KSM marks page as read only, when it is shared among multiple processes. But when a process tries to modify the page's content, it has to make duplicate first before it can write to the page. These step is called Break of CoW. This also adds potential overhead to the system

IV. Tools and Setup

A. Hardware and Software

All experiments were performed on a Intel®Core^{TM} i7-8565U CPU@1.80GHz, 8 CPU x86_64 Arch, 16GB RAM. Ubuntu 18.04 Operating system. KVM, libvirt and QEMU were used to create virtual machines.

B. Configuration

Following table shows the sysfs configuration parameters of KSM that was used for performing the experiments:

Parameters	Values
sleep_millisecs	200
pages_to_scan	100
max_page_sharing	256

Table 1: KSM configurations

C. Workloads:

A tool called 'stress' was used to create memory pressure inside the guest machines. It is a binary package available for most of the POSIX compliant systems, which generally used to generate configurable amount of CPU, Memory, I/O and disk pressures on the system. It is widely used by kernel programmers to create heavy workloads to test their applications. For virtual machines of 1 GB RAM a memory allocation/de-allocation load of 256MB is created for the experiments.

To analyse the KSM in action, we conducted the experiments in the following setups:

a) Single virtual machine having the same kernel as the host machine: An Ubuntu 16.04 machine with 1GB RAM
b) Two virtual machine with different kernels: An Ubuntu 16.04 machine and a FreeBSD 12.01 machine with 1GB RAM each
c) Two virtual machines with same kernel: 2 FreeBSD 12.01 machines with 1GB RAM each
d) Three virtual machines with same kernel: 3 FreeBSD 12.01 machines with 1GB RAM each

Metrics collected:

a) KSM sysfs metrics:
   - pages_shared: number of memory pages shared bu the KSM
   - volatile_pages: number pages marked as volatile by KSM
   - pages_unshared: number of page that was unshared due to CoW breaks
b) CPU utilization:top command was used get the CPU utilization metrics
c) Memory: vmstat command was used to get the memory metrics like currently used memory, free memory etc.

We collected the matrix every 10 seconds using a shell script and stored in a csv file which later was used by a python application to generate graphs and average statistics.

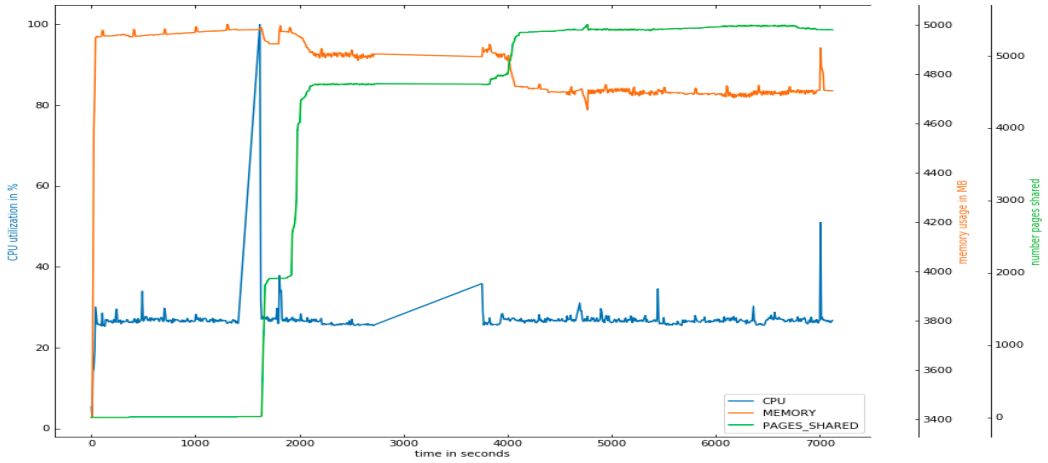

Figure 3: CPU utilization, memory usage and number of shared pages plotted for a setup with 2 VMs: (Ubuntu 16.04, 1GB RAM and FreeBSD 12.1, 1GB RAM) with KSM

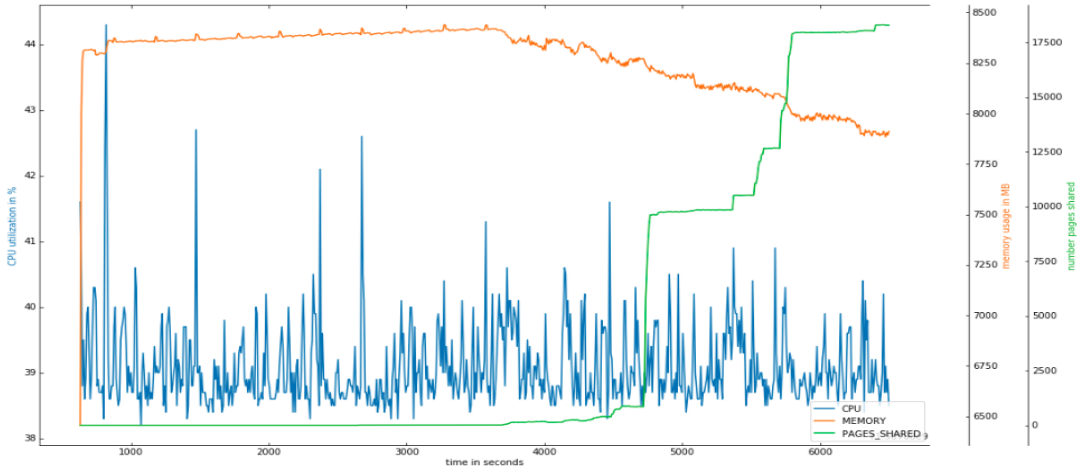

Figure 4: CPU utilization, memory usage and number of shared pages plotted for a set with 3 VMs (FreeBSD 12.1, 1GB RAM) with KSM

Scenario	pages_shared	volatile_pages
1 Virtual Machine with Ubuntu 16.04, 1GB RAM	4894	66413
2 Virtual Machines with FreeBSD 12.1,Ubuntu 16.04 ,1 GB RAM Each	5361	510248
2 Virtual Machines with FreeBSD 12.1, 1 GB RAM	18440	885621
3 Virtual Machines with FreeBSD 12.1, 1 GB RAM	18320	1348844

Table 2: KSM share pages across different scenarios

V. Results and Analysis

Table (2) has the list of KSM metrics observed for all the scenarios that was discussed in the previous section. The experiment was run for about 1.5 hours for each set up. As expected, we can observe that the number pages shared is comparatively less for virtual machines with different kernels. Kernel library contribute the maximum range of addresses for KSM's page scan. This observation is coming from the fact that the number of volatile pages doesn't show the same amount of drastic difference when we have guest machines with same and different operating system kernel compared to pages_shared count.

Figure 3 and Figure 4 are combined graphs containing KSM metrics and system metrics together to analyse KSM's efficiency. An important observation from this result is that KSM takes a substantial amount of time with current configuration for freeing up any significant amount memory. A minimum of 20 minutes, it takes to cross the boundary of 2000 pages. We can not observe any pattern in the CPU utilization with the 'pages_shared' metrics. It indicates that the current configuration doesn't have any significant impact on the CPU utilization. We can clearly make KSM, a bit more aggressive, if we have a need for more memory deduplication.

Scenario	Metrics	With KSM	Without KSM
1 Virtual Machine with Ubuntu 16.04, 1GB RAM	CPU	14.1%	13.5%
	Memory	2875 MB	2932 MB
2 Virtual Machines with FreeBSD 12.1, 1 GB RAM	CPU	26.5%	26%
	Memory	5791 MB	6505 MB

Table 3: System metrics comparison with and without KSM

Table 3 contrast the system metrics in the experiments with KSM enabled setup and KSM disabled setup. As we have discussed in the above observations the CPU utilization impact of KSM is almost negligible with the current configuration. In the first scenario, since only single guest machine is present. The amount memory reduced by the KSM is comparatively very less. But in the second case about 720MB of memory is found to be freed up by KSM. Which is definitely impressive, considering the fact that both machines are having only 1GB RAM each. The increase in the CPU utilization in both the cases are almost equal, proving the fact that the KSM is more suitable when there are multiple virtual machines with same kernel.

VI. Related Works

Kernel Same Page merging has been a proven module for memory deduplication in linux system. There are many papers which discusses the efficiency and side effects of KSM. Rachamalla et al. [2] have done a empirical analysis of KSM under a wide variety of workloads. The main focus of the work was to analyse the memory deduplication and the overheads under different configuration parameters of KSMs and find out a optimal set up for workload specific scenarios. They have shown that there is no globally optimum configuration KSM which attains maximum number of 'pages_shared' with minimal negative impact.They also have shown that for application having high memory updates, a more aggressive configuration of KSM will help in capturing the potentially short-lived same pages. And similarly for application with lesser memory updates a slow scanning rate of KSM would suffice.

VII. Conclusion

In this paper we have discussed the working of KSM and analyzed KSM's performance in multi-vm environment. We have seen that KSM is highly suitable when we have multiple guest machines having the same kernel. We also have observed that, KSM with default configuration parameters take at least 20 minutes to start having any substantial memory free ups. So it is definitely not suitable for virtual machines with shorter life span. CPU utilization impact is considerably less under the default KSM configuration, especially for system with more virtual machines. For infrastructures requiring higher degree of consolidation can configure KSM parameters to have more aggressive page scans to get better memory deduplication.

References

[1] Arcangeli, Andrea Eidus, Izik Wright, Chris., "Increasing memory density by using KSM," In Proceedings of the linux symposium. Citeseer, 19–28, 2009.

[2] Rachamalla, Shashank, Debadatta Mishra, and Purushottam Kulkarni, "Share-o-meter: An empirical analysis of KSM based memory sharing in virtualized systems.," 20th Annual International Conference on High Performance Computing. IEEE, 2013.

[3] W. Lin, C. Tu, C. Yeh and S. Hung, "GPU acceleration for Kernel Samepage Merging," 2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Hsinchu, 2017, pp. 1-6.

[4] Lublin, Uri, Yaniv Kamay, Dor Laor, and Anthony Liguori, "KVM: the Linux virtual machine monitor,", Ottawa Linux Symposium, Ottawa, 2007.

[5] Goto, Yasunori, "Kernel-based virtual machine technology,", Fujitsu Scientific and Technical Journal 47.3, 2011, 362-368.