IMPLEMENTING PAGE FUSION IN VIRTUALBOX FOR LINUX GUESTS

Sandeep Koppala, Snehal Saraf, and Ujwala Tulshigiri

ABSTRACT

Page Fusion or RAM deduplication is a feature present in VirtualBox that avoids the memory duplication across different virtual machines having similar operating systems. In a server environment, it is usual for multiple virtual machines running same operating system to run on the same host. In this case, lots of memory pages are identical (e.g. kernel code, shared libraries etc.). De-duplicating such memory pages in a copy-on-write fashion allows efficient utilization of host memory. Instead of scanning the entire guest memory using some daemon, maintaining the hash for each page and comparing them, Page Fusion uses the knowledge from within the guest to decide possible candidates for fusion which saves tremendous amount of time. Apart, this feature is fully transparent to the virtual machine.

PageFusion has been designed to work with x64 hosts and Windows guests (Win 2000 and later). Our project aims at implementing PageFusion for x64 Linux guests.

Keywords— ***PageFusion, RAM Deduplication, Page Sharing, VirtualBox,***

I. INTRODUCTION

Memory over-commitment is a very important feature of many modern hypervisors. This feature allows the virtual machines to be provided with more memory than what the host machine has. Virtual Machines running on the same host often have memory pages that are exactly identical. These memory pages, if identified, can be stored only once instead of multiple times. This is sometimes also referred to as memory de-duplication or same-page merging.

Traditional approaches to memory de-duplication are often slow and tedious. They involve no prior knowledge about the guest machine. The memory pages of the virtual machines are hashed and stored. Once the hash values are available, the hypervisor tries to find duplicates of hash values across different virtual machines. A duplicate hash means that it is very likely that the contents of the actual memory pages are same. This is verified before merging the different copies of the page together as a single copy-on-write page.

PageFusion is a feature offered by VirtualBox that allows memory over-commitment. However, PageFusion is different from the traditional approach, since it depends on information provided by the guest OS to selectively attempt to de-duplicate pages instead of a blind comparison. This means that PageFusion cannot completely extract all duplicate pages. But it offers advantages of being fast, and not affecting the performance of the VMM.

For PageFusion to work, the guest OS needs to have VirtualBox Guest Additions drivers installed. It is through these drivers that guest attempts to inform the VMM about the memory pages which are good candidates for de-duplication, and for long term. Examples of such pages can be kernel code, read-only pages, other executable code, etc. Presently, PageFusion is supported only for Windows guests. This means that the Guest Additions drivers that perform PageFusion are specifically written for Windows guests.

The goal of this project is to write Guest Additions drivers for Linux guests that will communicate relevant information about the guest memory to the VMM, so that the VMM can perform memory de-duplication across different virtual machines.

The rest of the paper is organized as follows. In section III, we explain the current implementation of PageFusion for Windows guests. We briefly explain the overall architecture, and the important functions that carry out PageFusion for Windows guests. In section IV, we propose a design for Linux guests based on the findings in section III.

In section V, we explain the implementation of the design along with some code snippets. In section VI, the results and metrics are presented. Section VII enumerates some major challenges and roadblocks that we faced during the course of the project.

II. PAGE FUSION FOR WINDOWS GUESTS – CURRENT DESIGN AND IMPLEMENTATION

PageFusion relies on the Guest OS's ability to provide information to the VMM about memory pages which can be good candidates for de-duplication. This communication happens through the Guest Additions driver (or VBox Driver) as depicted in Fig.1. There is a service called *VBoxService* which runs in the guest as a part of the Guest Additions. This service is responsible for carrying out a host of tasks, one of which is PageFusion. So, we focus specifically on PageFusion.

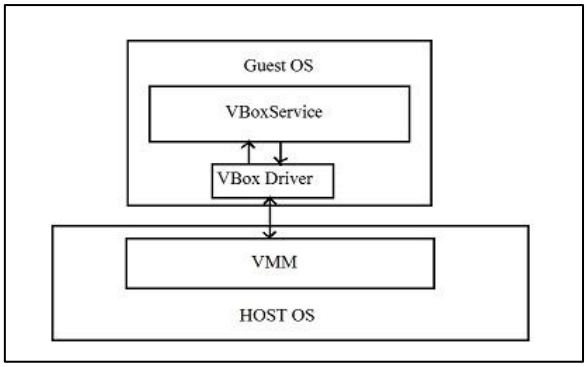

Figure 1: VirtualBox – Guest Host Interaction

When *VBoxService* is invoked with the option '*--enable-pagefusion'*, it forks another instance of the *VBoxService* to handle page fusion exclusively. This new instance performs the following tasks:

A. Inspecting the guest:

1. This function inspects the guest system to find all the processes running on the system.

2. For each process, it then inspects all the modules which are loaded by the process.

3. Thereafter, it attempts to load each module.

4. For each loaded module, try to insert it to the AVL tree. The AVL tree is used to keep a track of which modules have already been registered, and hence duplicate attempts can be avoided.

5. Finally it tries to register each loaded module with the VMM.

6. In addition, it also enumerates the system modules which are expected to be common and hence shareable amongst the various guests.

B. Registering the modules:

1. The code queries the file version of the module using *GetFileVersionInfoSize*, *GetFileVersionInfo* and VerQueryValue api's. The Win32 API's provide the file version information of the module to the *VBoxService*.

2. It then populates the aregion[]. This array contains the regions of the module to be shared which are virtually contiguous pages and have the same set of attributes. The attributes verified include whether all the pages are committed and have the same set of permissions. < eg : if the module is 75KB, and this module has the following contiguous set of pages { 10KB, 5 KB, 35KB, 25 KB } such that each set of pages have the same permissions and other attributes, the we create an array aregions consisting of 4 members describing each of the above 4 regions. The VirtualQuery() API is used for this purpose.

3. This function then invokes a function call *VbglR3RegisterSharedModule()* passing it the module name, the module version, the base address of the module, the no of regions in the module and the aregions[] array created in step 2 above.

4. The above call results in a call to the Ring0 which then ensures that the modules are registered appropriately.

C. Checking Shared Module:

The function *VbglR3CheckSharedModules()* is invoked to ensure that the registered modules are shared. This function call eventually triggers a VMM request to perform actual page sharing.

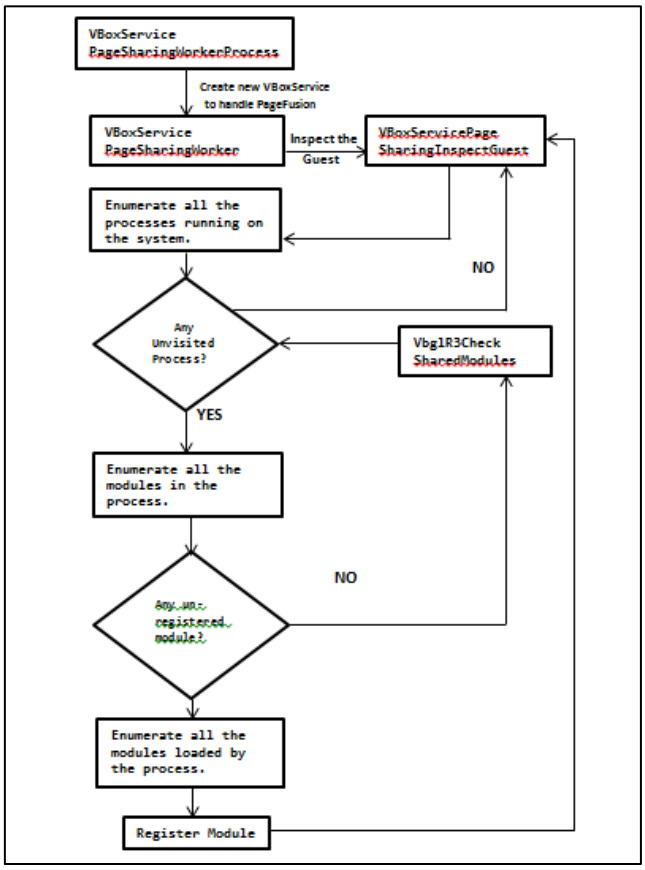

Figure 2.1 Logical code flow for windows Page Fusion

III. PROPOSED PAGE FUSION DESIGN FOR LINUX GUESTS

A. *Analysis of Linux Guests:*

Common sources for Page Fusion are Zeroed out memory pages, kernel read-only pages, specifically initialized memory regions (e.g. heap), statically or

dynamically loaded libraries. Linux OSes have shared libraries like libc, read-only system call table, kernel modules for drivers, file system and network which are common across multiple Linux guests. Apart there are some GUI applications like web browsers, document editors, web-servers like Apache Tomcat, which might contribute to memory duplication. If we use could fuse all the read-only / read-execute memory pages for these applications common across various Linux guests, we could save substantial amount of memory.

B. *Sources for Memory Duplication*

1. Shared modules common across the guests

As most if the drivers (sound, apic, usb, parport) and fs kernel modules (nfsd) are common across the guests, we can share the memory pages which hold the content of these modules. The file */proc/modules* provides the list of kernel modules with its name, memory size of the module in bytes, load count (instances of modules that are currently loaded), dependencies, load state and current kernel memory address of it. Having the load count greater than zero indicates that the module is loaded. This information can be used for registering the modules with the VMM.

2. Read-only kernel pages

By checking the permission bits for kernel pages, we can determine whether page is read only. Those pages are the one of the possible candidates for deduplication.

3. Shared Libraries

The examples of shared libraries are all .so files like libpthread.so, libc.so, libdb libdaemon.so.0 and libz. For given process, using *lsof* utility, we get the list of open files. Using Linux utility objdump, we can get the information of program and section headers, start address of .so. If two guests have exactly the same version, then only it we can send it's start address to VMM so that it can try to deduplicate the library across guests.

4. User Level Applications

By reading */proc/[pid]/maps*, we get the currently mapped memory regions and their access permissions. It provides the start and end address for each memory region.

C. *Proposed Design*

From the source code, it can be seen that the PageFusion driver code is currently implemented only for Windows Platform. We propose to write a function *VBoxServicePageSharingInspectGuest()* for the Linux platform that performs similar actions as described in section III(a). There are two major challenges with this approach:

1. Integrating the code changes so that the recompiled *VBoxService* invokes the *VbglR3RegisterSharedModule()* with appropriate parameters / information.

2. Figuring out the information as described above, so as to ensure that the number of pages being fused is as high as possible.

The first challenge is more of an implementation problem. However, the second challenge is more of a design concern. We have attempted to get the information regarding shareable modules in Linux by querying "proc" file system.

The */proc/modules* file in the Linux file system contains a list of all the modules that are present on the system with other information like base address of the module, size of the module, reference count, etc. We attempt to use this information in the *VBoxServicePageSharingInspectGuest()* function to register shareable modules with the VMM, which will eventually attempt to page-fuse them.

IV. PAGE FUSION IMPLEMENTATION FOR LINUX GUESTS

We have implemented the function called as *VBoxServicePageSharingInspectGuest()* as described in the previous section. This function opens the file /proc/modules and parses its contents to build an array of structures that contain the module information as obtained from the file.

For each module thus found, we invoke another function, *VBoxServicePageSharingRegisterModule()*. This function also mimics the behavior of its namesake for Windows guests. For Windows, this function can be invoked in two modes – the *fValidateMemory* flag set to either True or False.

For Linux guests, we set this flag to False. *VBoxServicePageSharingRegisterModule()* function treats each module as one region, and makes a call to the *VbglR3RegisterSharedModule()* function, passing appropriate parameters.

Once all the modules have been registered with the VMM as explained above, *VbglR3CheckSharedModules()* is invoked. This makes sure that the VMM R0 calls are triggered that are supposed to check all the registered modules so far and perform page sharing wherever possible.

Pseudo codes for the two main functions are given below:

I) module VBoxServicePageSharingInspectGuest()

1. Open the file */proc/modules* in read only mode

2. for each line read from the file:

(a) Split the line into tokens based on the delimiter ' '.

(b) Push all the tokens into 'C' structure of type LinuxModule and insert it into an array of Linux struct modules.

3. for each structure in the array:

(a) Call *VBoxServicePageSharingRegisterModule()*

(b) Call *VbglR3CheckSharedModules()*

II) module VBoxServicePageSharingRegisterModule()

1. Build aRegions[] array with the module address and module size
2. Call VbglR3RegisterSharedModule with appropriate paramters

V. EXPERIMENTS AND RESULTS :

We have been successfully able to integrate the code changes, and compile the Guest Additions successfully for Linux (x64) guests. We use the VBoxManage metrics feature on the host machine to display the data for the metric RAM/VMM/Shared. This data shows the exact memory that is being shared across VMs.

We have also verified that the modules that we are registering from the guest OS are actually getting registered with the VMM successfully. We verified this from the VM logs that are dumped on the host machine. With the data obtained from /proc/modules, we have not been able to see any statistics for PageFusion for two Linux guests.

To analyze this issue, we traced and studied the entire call stack in the ring0 (hypervisor). The relevant functions that perform page sharing are defined in GMMR0.cpp. We tried putting traces and enabling the log statements using the Virtualbox logging infrastructure. However, we were unable to obtain the same with release as well as debug builds.

In order to investigate without logs, we performed the following experiments -

Experiment I:

To verify the assumption that the host side page-fusion code is agnostic of the guest OS, we conducted an experiment by running processes in the guest machines that allocate identical number of pages and write the same data into these pages. The base address of this chunk is then dumped into a file - /tmp/modules. The guest additions code is recompiled to pick up data from /tmp/modules instead of /proc/modules.

In this experiment, we have seen only one page being fused across the two Linux guests.

Experiment II:

This experiment is identical to the first experiment but instead of allocating pages in the user mode, we wrote a linux kernel module (LKM) that allocated a bunch of pages. We then filled these pages with specific bit pattern and recorded the base address and other details in /tmp/modules.

In this experiment too, we have seen only one page being fused across the two Linux guests.

Experiment III:

In this experiment, we used the information from /proc/<pid>/maps for a specific user application [We ran identical version of the VLC media player on two linux guests]. This file contains the currently mapped memory regions for the application and the dependent shared libraries. It also specifies the permission bits.

Thereafter, we filtered out a few "Read-Only", "Shared" and "Read-Execute" regions from the pool. The information about these pages were inserted into the /tmp/modules and the same procedure described above was repeated.

In this experiment too, we have seen only one page being fused across the two Linux guests.

VI. CHALLENGES AND ROADBLOCKS IN PAGE FUSION IMPLEMENTATION

We faced many challenges during the course of this project so far.

A. Building VirtualBox on Windows x64

Building VirtualBox on Windows x64 hosts is a complex process with lots of dependencies. We had to investigate this entire process to get our code compiled on Windows x64 hosts.

B. *Enabling logs on VirtualBox*

Logging infrastructure in VirtualBox is complicated. Even after setting up the environment as per the instructions on the wiki, we were unable to set up the logging properly. We had to resort to indirect techniques to get a view of the code flow. We could not get the Ring0 logs.

C. *Using gdb with VirtualBox*

We have not been able to debug the VirtualBox host code with a debugger (like gdb). We encountered host OS crashes on attempting to run the page fusion code with debug builds.

VII. CONCLUSION AND FUTURE WORK

We were successful in registering the modules to be shared with the VMM. We were also able to share one page across the Linux guests. As a part of immediate future work, we need to scale up this shared page count. We need to also investigate and implement other alternatives identified in this paper to determine the candidate pages for page sharing.

VIII. REFERENCES

[1] https://forums.virtualbox.org/

[2] A Comparison of Software and Hardware Techniques for x86 Virtualization (VMware) - Keith Adams, Ole Agesen

[3] Memory Resource Management in VMware ESX Server - Carl A. Waldspurger

[4] Kernel Same Page Merging
http://www.ibm.com/developerworks/linux/library/l-kernel-shared-memory/index.html?ca=dgr-lnxw01LX-KSMdth-LX

[5] Linux Kernel Module Programming Guide
http://www.tldp.org/LDP/lkmpg/2.6/html/

[6] Runtime Benefits of Memory Deduplication
http://os.ibds.kit.edu/downloads/da_2012_rittinghaus-marc_memory-deduplication.pdf