I.J. Information Technology and Computer Science, **2016, 4, 11-18**
Published Online April 2016 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2016.04.02

Virtual Machine Monitor Indigenous Memory Reclamation Technique

Muhammad Shams Ul Haq
Beijing Institute of technology, School of Computer science and Technology, Beijing, 10081, China
E-mail: shams_5@yahoo.com

Lejian Liao and Ma Lerong
Beijing Institute of technology, School of Computer science and Technology, Beijing, 10081, China
E-mail: {liaolj, malerong_bit}@bit.edu.cn

***Abstract*—**Sandboxing is a mechanism to monitor and control the execution of malicious or untrusted program. Memory overhead incurred by sandbox solutions is one of bottleneck for sandboxing most of applications in a system. Memory reclamation techniques proposed for traditional full virtualization do not suit sandbox environment due to lack of full scale guest operating system in sandbox. In this paper, we propose memory reclamation technique for sandboxed applications. The proposed technique indigenously works in virtual machine monitor layer without installing any driver in VMX non root mode and without new communication channel with host kernel. Proposed Page reclamation algorithm is a simple modified form of Least recently used page reclamation and Working set page reclamation algorithms. For efficiently collecting working set of application, we use a hardware virtualization extension, page Modification logging introduced by Intel. We implemented proposed technique with one of open source sandboxes to show effectiveness of proposed memory reclamation method. Experimental results show that proposed technique successfully reclaim up to 11% memory from sandboxed applications with negligible CPU overheads.

***Index Terms*—**Sandbox, Library OS, Virtualization, Memory Management, Memory reclamation.

I. Introduction

Virtualization has become an essential part of operating system design ranging from traditional resource sharing among operating systems to innovative designs for particular problem solving such as process migration, load balancing, process isolation, process sandbox, privilege separation and prototype testing. One of the main reasons, other than semantic isolation, for such prevalence of virtualization in problem solving is facilitation from hardware to efficiently perform virtualization with less or no overheads depending on the nature of task.

Among innovative and nonconventional use of virtualization, recent past witnessed a good research effort for process isolation and application sandboxing. Many sandboxing schemes are proposed by research community [1-6]. In sandboxing/isolation schemes, processes are usually confined in Virtual Machine equivalent execution environments [3-7] so that malicious process cannot effect other processes and over all execution of the system. Such schemes provide each process with required functionality with in its own address space in the form of APIs usually termed as Library OS [6]. This set of APIs encapsulates the system calls as high level library functions and provide narrow and controlled interfacing with host kernel. This provision of Library OS along with Virtual machine Monitor (VMM) make it possible to sandbox unmodified applications. Such sandboxing schemes are not only practically efficient than full virtual machines but also provide theoretically Virtual machine equivalent isolation.

However, a key drawback with these sandboxing schemes is that sandboxed application's execution demands high memory usage than native execution. Sandbox run time needs its own memory allocation for each instance of application execution. As for example, when simple UNIX utility ls (with switch -R) is sandboxed with dune [7], it incurs 124% memory overhead as compared to native execution on UNIX. In the same way gzip consumes 90.1% more memory when it is sandboxed with dune. This overhead leads to thrashing and system unresponsiveness when all or most user applications need to be sandboxed.

Despite the fact that recently prices of memory decrease significantly, still personal user Desktops and laptops are not installed with huge memories. We conducted survey about specifications of computers and laptops used by students in our campus. We got feedback from 1453 students. We found some interesting observations about system specifications and level of user satisfaction. According to survey, 49% students have systems with 4 GB or less RAM. Whereas 53% students prefer one specific freeware antivirus product for reason that antivirus engine claim to use less memory than others. This informal survey provides insights that end user is not enjoying worry free amount of memory in their systems. All software designs, specifically, security solutions must address the problem of memory used by product.

Dynamic memory management techniques defined for virtualization world such as memory overcommit [8], hotplug [9] and KSM [10] does not suit for sandbox systems. These techniques usually install a driver in guest OS like ballooning [8] or help native kernel to take memory back from virtual machines like KSM. In sandboxes, installing driver or allowing other module to share sandboxed memory can result in security problems. In order to fully utilize the benefits of sandboxing, VMM must have indigenous memory reclamation mechanism to reclaim memory from sandboxed applications. VMM based memory reclamation must be independent from native kernel and guest operating system.

This paper describes a novel method to reclaim memory from a process sandboxed by Virtual Machine Monitor (VMM). The proposed method is based on Page modification logging (PML) introduced by hardware vendors to automatically log dirty Guest Physical Addresses (GPAs). This presented work uses PML logs as predictive active working set of the sandboxed process and prioritize pages for eviction in this working set.

Majorly, this paper contributes following:

- Explains the process of transitioning an ELF binary to virtual machine (also termed as process container) by presenting Dune [7] way of transition [section IV]. This section provides good base to understand under the hood details of process confinement.
- Propose memory reclamation algorithm, based on Least Recently Used and Working Set algorithms that works indigenously in VMM [section IV]. Proposed technique reclaimed memory that ranged from 5% to 11% depending on execution nature of application and work load. During experiments, 3 out of 5 times less thrashing occur with this technique under huge memory stress generated by memory stress tool.
- Implement a prototype for the proposed technique as extension to Dune on Intel hardware. [Section V].
- Provides a thorough Experimental Evaluation and comparison with original dune [section VI]. The proposed work incur minor CPU overheads in range of 0.5% to 4.7%

II. Related Work

Even though Dynamic memory management in virtualization draws sufficient attraction of research community, we will discuss two dynamic memory management techniques used by VMWare products and open source VMMs like KVM[14] and Xen[15]: Ballooning or memory overcommit[8,16] and memory hotplug[9]. VMware introduced the concept of memory ballooning to force guest OS to release pages when native server is under memory stress. Ballooning works by adding driver to guest OS to cooperate with VMM. With the help of balloon driver, VMM reclaim memory from guest OS by inflating balloon. While on deflation of balloon, VMM returns memory to guest OS. Although ballooning has been widely used for memory reclamation [17-19] but this technique does not suit for VMM meant for sandboxing. Our proposed technique is quite different from ballooning because we do not add any driver in guest environment. Memory Hotplug is considered most effective and last resort for reducing memory stress on system. Memory hotplug refers to technique which is used to add memory to an active system without any downtime. Hotplug memory needs support from hardware and kernel. The work presented in [9] adds memory hotplug functionality to Linux kernel used by Xen. We believe that memory hotplug does not interfere with our proposed technique in case underlying infrastructure support hotplug. In the same way KVM uses KSM [10] for sharing similar anonymous pages. KSM Linux module de-duplicates same pages used by processes to reduce memory burden on the system. KSM can be useful in sandbox where many instances of same program such as web browsers are running in different sandboxes. But security concerns need to be evaluated for using KSM to reclaim memory from sandbox process.

Sandboxing and process containers represent techniques used to monitor and control the execution of the process. As our proposed memory reclamation technique is used with process sandboxing design so here we are also providing state of the art for process sandboxing. Graphene proposed in [1] is one of such sandbox techniques that confine multi process applications. Graphene uses picoprocess [20] to encapsulate each process with in host process. For multiprocess constructs, graphene defines modified RPC mechanism which make possible for processes of application in same sandbox to communicate. Graphene is different from dune design as Dune sandbox single process applications. LXC [21] is a process container shipped with Ubuntu 14.04 with powerful API to configure process containers. LXC and its variants use kernel features such as seccomp filters, apparmor, chroot and Cgroups to create process sandbox. LXC uses kernel features as opposed to dune that uses hardware virtualization to confine application. Denali [3], Apiary [2], Qube [4] and Virtuozzo containers [5] among other proposed architectures restrict application behavior and isolate application execution either on system level or user level. Dune is different from all other architectures due to user space applications direct execution on privilege hardware.

Other than sandboxing applications, one direction of research check reliability of application such as [25]. This direction is different from sandboxing so we do not include state of the art in software reliability techniques PML has been used for traditional purposes of working set determination in Xen [22] and KVM [14] but NFV Hypervisors-KVM [23] use PML for out of box solution for pacing live migration of virtual machines. We did not witness use of PML for memory reclamation and thrashing avoidance.

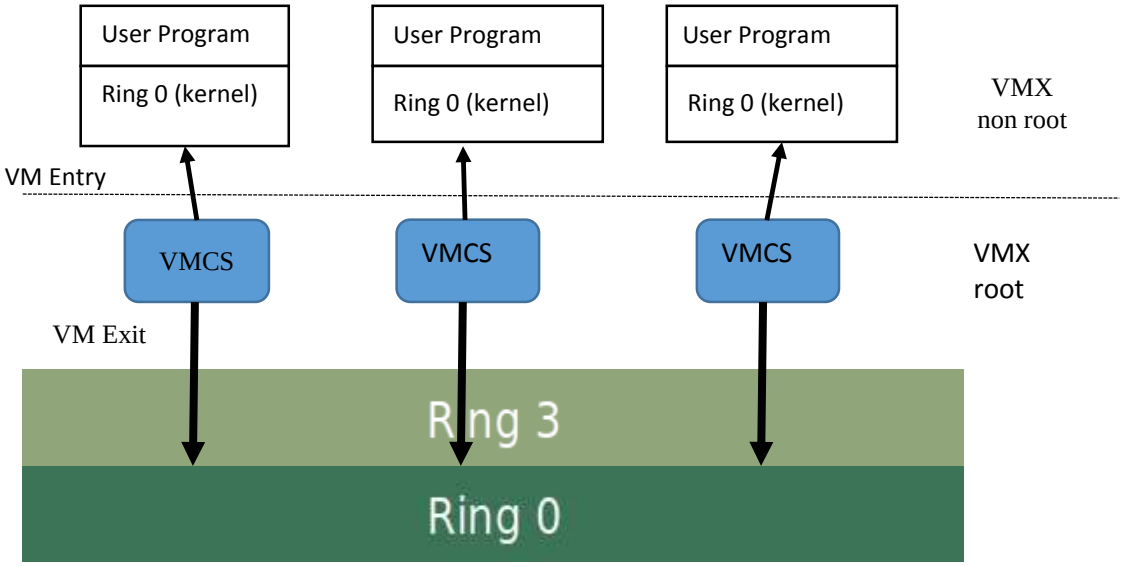

Fig.1. Hardware virtualization operation showing VMX root and VMX Non root modes with transition carried out by VMCS

III. Hardware Virtualization Extensions

This section briefly explains the virtualization extensions provided by Intel. We will discuss only Intel hardware because Dune is implemented on Intel but discussed extensions are conceptually same for AMD.

Intel introduces VT-x technology [12] to ease and simplify the implementation of VMM. VT-x divides the processor into VMX root and VMX non root modes. CPU performs transitions between VMX root and VMX non root mode called VMX transitions. Both modes have traditional four privilege rings. VMX root mode is used for VMM or host operating system whereas VMX non root is used for guest OS. In order to perform transitions between modes, Intel introduces new instruction set. New instructions VMLAUNCH and VMRESUME enters the VMX non root mode and executes the guest OS. This transition from root to non root mode is called VM entry. On events that change system state, Hardware performs VM exit to enter the root mode so that executive software, either host OS or VMM, may take the control of the system.

For facilitation of VMX transitions, Intel provides with memory resident structure called Virtual Machine Control Structure (VMCS) that is a configurable structure to meet the individual requirements of different use cases. VMCS is a CPU readable structure and holds the guest state when CPU runs in root mode whereas holds host state when CPU serve the guest OS. This load and store of machine state is automatically performed by hardware. The VMX transition through VMCS is shown in fig. 1 in which transition is controlled by VMCS in both directions, VM entry and VM Exit. In the same way VMCS contains large amount of configuration options to control VM exits. VMM can configure VMCS to cause VM exit on execution of specific instructions such HLT or on some events such as page faults.

In the same way Intel introduces extensions to virtualize memory Management Unit (MMU) for second level of translations through Extended Page Table (EPT). In Intel based virtualization, every address translation goes through two sets of page tables: guest page tables managed by guest OS and EPT managed by VMM. Guest page tables translate guest virtual address to guest physical address whereas EPT performs translation of guest physical address to host physical address. On VM entry, CPU automatically stores the guest page table root in %CR3 to allow guest OS to manage its local page tables. But actual physical memory (host physical) is only accessible through EPT.

Intel also introduced different technologies such as virtual Processor Identifiers (VPIDs), EPT access and dirty bits (A/D) and page modification loggings to gain efficiency. VPIDs tag TLBs so that there is no need for TLB invalidations on each VM Exit. In the same way, as noted in [24], hardware managed TLB let VMM to cache guest virtual to host physical address translations. This cache escapes the two level address translation and hence provide efficiency.

IV. Dune Design and Implementation

Dune provides a process direct but save access to privilege hardware such as page tables, rings and tagged TLBs and implement three use cases, a privilege separation utility wedge, garbage collector Boehm and process confinement sandbox. Dune sandbox a process by transitioning it into VMX non root ring 3. Dune make use of Intel VMX and EPT to provide VM-equivalent isolation needed to sandbox a process.

The released code base [11] also provides running examples of sandbox, wedge and user firewall. Dune make useof Intel VT-x and EPT for providing applications with direct access to privileged hardware.

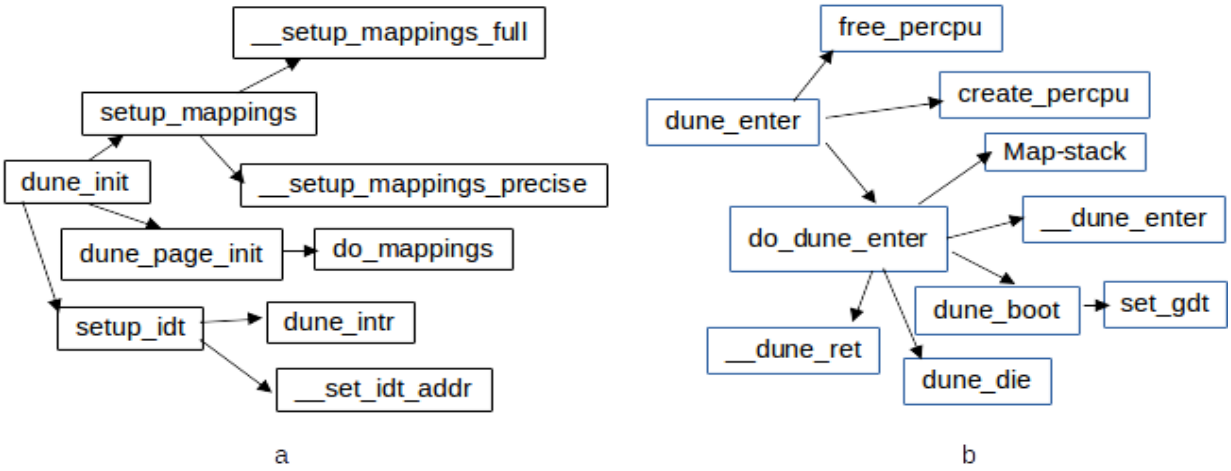

Fig.2. Dune working a) shows function calls to initialize dune mode while b) shows steps for transition of binary into dune mode

In dune implementation there are two parts one is a core kernel module that installs a driver for enabling VT-x to extend kernel for VMM functionality while other part, libdune, provides applications with required functionality such as CPU(virtual), page table, tagged TLBs, IDT and GDT to run in virtual machine (dune mode). In order to confine the application dune transitions the application in VMX non root ring 0. At the moment dune only sandbox single process, single threaded applications. Overall there are three modes for an application to run on system having dune: 1) run as native system 2) run in dune mode 3) run in sandbox. The native application can run as there is no dune involved while in dune mode application runs in VMX non root ring 0 where an application has its own VCPUs and memory.

In order to sandbox a process, dune transitions unmodified ELF binary into VMX non root ring 3. For transition binary in ring 3, two ELF loaders are used; first minimal ELF loader securely loads second ELF loader, ld-linux.so, into untrusted userspace. This second loader loads the untrusted ELF in userspace without possibility to affect the runtime sandbox. All actions from untrusted binary that change privilege state of the system are intercepted by sandbox runtime and handled by call back handler, registered with libdune, to restrict and control the execution behavior.

The implementation of VMX driver is based on KVM but is simple as compared to KVM. Dune does not need many functionalities implemented by KVM such as nested virtualization, I/O virtualization or backward compatibility. The dune module can easily be installed with general UNIX commands insmod or modprobe.

The transition of a process into confined environment starts by reading binary headers and making arrangement for memory layouts. In start, text, data, heap, and stack are provided, along with empty EPT table, for the execution of the program. As application accesses the memory, mapping is generated accordingly on page faults. On need memory layout can be expanded up to full address space.

Dune provides two functions, dune_enter and dune_init, which consolidate dune entry for an application. Fig. 1 shows important function calls made by both functions to transition a process in dune mode. A process must call dune_init before entering dune through dune_enter. The function dune_init provides a process with access to libdune. dune_init setup the environment for Execution of process as independent entity. Fig. 1.a shows that page table initialization, interrupt descriptor table setup and memory mappings are performed through dune_init. Dune uses two types of mappings, precise and full. Precise mapping save memory by mapping only actually used address space while full mappings map complete address space of a process. After environment setup, dune_enter (figure 1.b) transitions the process in dune mode. This function create vcpu, creates GDT and call an assembly routine, __dune_enter, to call dune_enter ioctl to actually transition a process.

V. Proposed System

On basis of limitations of library OSes [6] and problem domain, we highlight following design goals for memory reclamation techniques proposed in VMM based sandboxes

i) Simple: the proposed technique must be simple and does not introduce extra CPU overheads otherwise it will become another source of overheads.

ii) Lightweight: it must be lightweight so that memory gain, in all levels of memory stress, must be greater than memory used by this technique.

iii) Indigenous: it must be independent from host and guest operating systems. In VMM based sandboxes, Library OS is a collapsed form of operating system which provides minimum possible execution environment to process. Under such restricted execution environment, introducing some sort of driver is not a feasible solution. It means it should not install any driver in guest and should not dependent on any channel of communication with host.

In order to devise a technique that meet above established goals, we propose a technique that is modified

form of Working set memory reclamation technique. The proposed technique integrates working set memory reclamation technique with LRU so that those pages from working set of confined process that are least recently used become preferable pages to be evicted. The working set memory replacement depends on observation that most frequently used construct of a program consists of a small set of instructions. As added memory overheads from isolated process is due to sandboxed environment that provides a process with local library OS and execution monitoring, we can confidently establish that keeping most of these pages in memory results in less page faults. But to make technique suitable for each level of system memory stress, we need to further identify the preferable candidate page for eviction. For maintaining this preference we categorize the pages with approximate time of use.

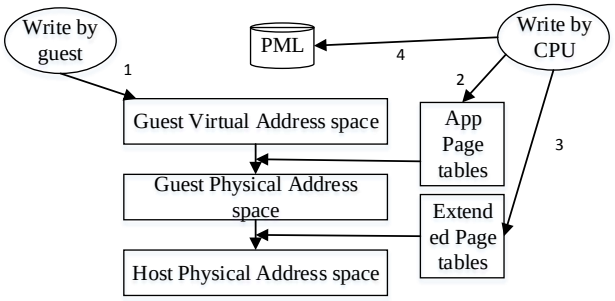

Fig.3. Working Set Acquisition through PML

A. *Acquisition of working set*

In virtualization, working set of a guest mode software is a set of dirty guest physical pages. Generally collecting working set of application is a computationally expensive task due to extra page faults and VMEXITS. But fortunately, Intel recent processors shipped with advanced virtualization extension, Page modification logging [13] that automatically log dirty GPAs. Owing to be hardware automatic facility, PML provides working set with negligible overheads.

Before introducing PML, Intel introduces Accessed and Dirty (A/D) bits for EPT so that memory manager can check which pages are modified or accessed without computationally expensive write protected method. As these bits in EPT are set by processor automatically without knowledge of VMM, collecting working set of guest software become expensive. To reduce overheads for working set collection, Intel added page modification Log extension so that whenever guest OS access or modify a page, CPU writes relevant GPA in buffer. Intel provides in-memory buffer of 4K to store dirty GPAs. Each entry needs 8 bytes, so Max. 512 dirty GPAs can be in buffer. A VMEXIT occurs at the event of buffer full so VMM can use this information on VMEXIT.

To further reduce overheads and to make possible timely availability of working set, we collect logs on each VMEXIT instead of buffer full event. One interesting feature of PML is that CPU treats all type of accesses as write access. Making all references as write gives actual set of memory accessed between recent VM Entry and present VMEXIT.

B. *Memory Reclamation Algorithm*

We implement PML log per VMCS to provide each process with its own PML log. This implementation is straightforward because dune also implements VMCS for each process. The changes made in VMCS are discussed in following paragraphs and implementation section.

In order to meet design goals, we aim to use working set without extensive computation involved as calculating exact page used time is not feasible. What we have in hands is the set of pages which are accessed and modified during execution of the application. As it is not necessary for our paging scheme to evict pages on each VM Exit, LRU principle is followed to sort pages according to time of access so that least recently used pages become favorite for page eviction. Our technique maintain a priority stack (fig. 3) which is updated on each VM exit. Whenever new log is available on VM Exit, paging algorithm rearrange the stack so that recently used page must be on top of the stack with least recently used pages at bottom. On receiving PML log on VMEXIT, the stack is filled with the GPAs that represent working set of process. On next VM exit, stack is manipulated to keep the recently used pages at the top of the stack while least recently used pages at the bottom. The page reclamation is from bottom to top so that least recently used pages should be reclaimed. In this scenario, Re-arrangement of stack on each VMEXIT is acceptable alternative to build slow clock for rearrange time to reduce overheads incurred. This page reclamation algorithm integrated with hardware managed working set collection meets all design goals: simple, Indigenous and lightweight.

Example: let's say first PML log consists of 4 pages {a,b,c,d}. Activation of Memory reclamation method at this moment evicts pages in order of {d,c,b,a}. But let's say 2 new logs are available on next two VM exits with pages (e,f,g) and (d,g,h). Fig. 4 shows the re-arrangement of LRU stack.

VM exit	1	2	3
Working set	a b c d	e f g	d g h
Stack arrangement	a b c d	e f g a b c d	d g h e f a b c

Fig.4. Stack arrangement for page reclamation technique with three working set instances on three VM Exits. VMEXIT 1 has first working set, VMEXIT 2 has all new pages in working set and VM EXIT 3 contains some previous pages.

On availability of new PML log after VM Exit 2, all newly modified pages (e,f,g) are added like FIFO (first In First Out), making pages in working set favorite candidate for eviction. But new PML log containing pages that are already in stack, page {d and g} in VM

Exit 3, requires rearrangement of stack so that newly modified pages must be treated as most recently referenced pages. In case of VM Exit 1 and VM Exit 2 page d is most likely to be paged out while after VM Exit 3, it is least likely to be paged.

Our VMM based paging mechanism has two phases: one phase works in normal low memory condition while other contingency phase works under very low memory condition to avoid thrashing. The low memory and very low memory conditions are assessed on the basis of number of page faults generated by a process. VMM Paging system maintain a counter to determine the number of page faults for each process.

VI. Implementation

We implemented our work on 64 bits Ubuntu 14.04 with kernel version 3.13.0 on 64 bit machines with Intel broadwell processor. We first develop dune bit by bit starting with VT-x driver that extend host kernel with VMM capabilities. After module successful implementation, we implemented libdune a Library OS to facilitate ELF binary to run its own Virtual Machine alike environment. We then incorporated A/D bits in EPT and implemented PML according to the instructions given in [13]. In our implementation we keep PML logging all time active. PML provides 512 entries of dirty GPAs which are aligned to 4KBytes that means PML writes effected page frame number. In order to refresh buffer on writing of 512 frames, A VM Exit is scheduled in VMCS on buffer full event. On each write in buffer, a counter, 16 bit PML index, initialized with 511 is decremented. When PML index reaches zero buffer full event is triggered. We also configure VMCS to flush PML buffer on every VM Exit.

A. Integration with Dune

In modified dune, dune module detects the availability of PML support in processor and enable PML logging for process when detection routine returns true. Otherwise dune fall back to its own execution flow. As dune creates VCPU for each VMCS, we enable PML on creation of each VCPU. In order to create VCPU with PML support, Dune configure the VMCS to add PML index, base address of buffer and turn on PML execution control. As dune already uses VMCALL to perform system calls, we use this interface to start dirty logging. Modified dune now kicks start logging after put_cpu and vmx_run_cpu so a process on entering guest mode (VMX non root mode) by VMLAUNCH or VMRESUME start logging GPN. In order to get PML buffers on VM Exit, modified dune take back all VCPUs assigned to process which automatically flush buffer. We collect Logs in VMM own memory to release VMCS encoded buffer before storing GPAs in list. Modified dune then sort the GPAs according to algorithm defined so that on activation of reclamation process, simple pop operation on list head provide the paging routine with address to be purged out.

VII. Evaluation

We evaluated our work for two factors: what is memory gain and what is the overhead, in terms of CPU time, with respect to original dune sandbox. We used time utility to calculate RSS and execution times. We performed first experiments by sandboxing own built recursive-ls utility used on / directory which recursively visits in each directly to list contents. Table 1 shows the percentage of memory gain achieved when programs are sandboxed with our proposed page eviction scheme. Utility ls sandboxed with original dune uses 124% more memory while sandboxed with our implementation, its overhead reduces to 115%. It means our page reclamation scheme successfully achieve 9.0% gain in memory with only 0.18% CPU overhead. For gzip program we compress and decompress 4.1 GB data consisting of windows 7 installation package. Proposed technique successfully reclaimed 11% memory used by sandboxed gzip.

Table 1. Memory gain in percentage

	Memory gain
ls	9
gcc	11
gzip	7
lighttpd	5

One reason for more gain in gzip execution is due to its more memory usage than ls and lighttpd which reclaims only 5% memory.

Table 2 represents execution overheads of both original dune sandbox (without PML eviction) and enhanced sandbox (with PML eviction) in percentage. Our time overheads are different than original paper. Different configurations can be one reason. All programs ls, gzip and gcc exhibit very less overheads for execution time; ls, gzip and gcc incur execution 0.18%, 0.04% and 0.04% overheads respectively.

Table 2. Execution time overheads in percentage

	Ls -R	gzip	gcc
Original Dune	4.7	0.89	0.43
Modified Dune	4.95	0.93	0.47

We repeated the experiment of sandboxing lightttpd with and without our extension to benchmark its execution by using apache ab benchmark. We used same machine as server and client to get a static web page hosted by Lighttpd. In our experiments, time latency of dune w/o PML is 4.3% instead of 2% as stated in original paper. This difference can be due to some configuration differences in ab, Lighttpd and underlying system; and network stack operational intricacies.

Table 3. lighttpd performance c=1 for latency and c=100 for throughput

	C=1	C=100	%decrease
Linux	5562	18172	
Original Dune	5494	17231	5.1
Modified Dune	5466	17144	0.5

We did not stress on the network stack being on the same machine. Table 3 shows the average requests per second of 20 executions for each category with a gap of 4 minutes between each experiment. Table 3 shows 0.5% reduction in throughput but for the same experiments memory gain is 5%.

In order to check how proposed solution works under huge memory pressure of more than 90% memory usage, we tweak with memory stress tool to generate desired load. Our method successfully avoided thrashing 3 out of 5 times where original dune failed to avoid. This experiment also highlights rationale for proposing VMM based memory reclamation where possible.

Discussion

The amount of memory successfully reclaimed from isolated process suggests some useful insights. The amount of memory reclaimed directly depends on the footprint of process in memory. This direct relationship is due to fact that Logs generated during execution will mark more addresses for a process with high memory footprint. As for example, memory reclaimed from gzip is highest as its memory usage is high. The results of gcc are interesting as its memory usage is high but reclaimed memory is low with 7% gain. One of reason for such low reclamation can be due to temporal locality and no of page faults produced by application. In our experiments, gcc has less temporal locality and high number of major page faults under high memory stress. This irregular behavior is worst case scenario for our proposed technique. The applications that have regular pattern of memory access (high temporal and spatial locality) get more benefits from VMM based indigenous techniques.

In order to avoid thrashing, Linux uses swap management in which some processes are removed from memory and are stored on hard disk. But this thrashing management, through swapping processes on disk, in environment where processes are sandboxed by VMM is not adequate. This inadequacy is due to semantic gap between guest view of execution and native host OS. This semantic gap results in eviction of pages blindly and results in more page faults. For example in case of multi process application, host may swap one process while other process remains in the memory requesting services from swapped process. Our scheme starts working even before aggressive swapping start by host OS. VMM based paging routine start eviction of pages more aggressively but maintains reasonable eviction rate to keep number of page faults under certain threshold.

VIII. Conclusion

In this paper, we try to open a new research direction in process sandboxing by explaining that one of major problems for not using sandboxing at large scale is memory inefficiency of proposed techniques. We proposed page reclamation technique which is based on Intel hardware virtualization extension called Page Modification logging. The proposed technique does not require any driver in guest or any other new interface with guest and host software to reclaim pages. Our proposed technique efficiently reclaimed memory up to 11%. We believe that reclaimed memory through PML logs can be increased by devising more sophisticated page reclamation algorithm. One possible future direction, to select page for eviction, can be assigning weights to addresses as function of presence of addresses in number of VM Exits and keeping history of page faults by process.

References

[1] Chia-Che Tsai , Kumar Saurabh Arora , Nehal Bandi , Bhushan Jain , William Jannen , Jitin John , Harry A. Kalodner , Vrushali Kulkarni , Daniela Oliveira , Donald E. Porter, Cooperation and security isolation of library OSes for multi-process applications, Proceedings of the Ninth European Conference on Computer Systems, April 14-16, 2014, Amsterdam, The Netherlands

[2] Potter S, Nieh J. Apiary: easy-to-use desktop application fault containment on commodity operating systems. In: USENIX Annual Technical Conference. Boston, MA, USA: USENIX Association; 2010.

[3] Whitaker A, Shaw M, Gribble SD. Denali: lightweight virtual machines for distributed and networked applications. In: 5th USENIX Symposium on Operating Systems Design and Implementation. Boston, MA, USA: USENIX Association; 2002. p. 195-209

[4] Rutkowska J, Wojtczuk R. Invisible things lab, technical report: Version 0.3 Qubes OS architecture; 2010

[5] Parallels Inc. Virtuozzo containers. http://www.parallels.com/au/products/pvc46/

[6] Avi Kivity, Dor Laor, Glauber Costa, Pekka Enberg, Nadav Har'El, Don Marti, and Vlad Zolotarov, Sv—optimizing the operating system for virtual machines. In Proc. USENIX Annual Technical Conference (ATC) (Philadelphia, PA, June 2014), USENIX Association, pp. 61–72.

[7] A. Belay, A. Bittau, A. Mashtizadeh, D. Terei, D. Mazi`eres, and C. Kozyrakis. Dune: safe user-level access to privileged CPU features. In OSDI, pages 335–348, 2012

[8] Carl A. Waldspurger, Memory resource management in VMware ESX server, ACM SIGOPS Operating Systems Review, v.36 n.SI, Winter 2002 [doi>10.1145/844128.844146]

[9] D. Hansen, M. Kravetz, B. Christiansen, and M. Tolentino, "Hotplug Memory and the Linux VM," in Proc. Linux Symp., July 2004, pp. 278–294.

[10] A. Arcangeli, I. Eidus, and C. Wright. Increasing memory density by using ksm. In Proc. of Linux Symposium, July 2009

[11] http://dune.scs.stanford.edu/ (time accessed: 13th Jan 2016).

[12] R. Uhlig, G. Neiger, D. Rodgers, A. Santoni, F. Martins, A. Anderson,S. Bennett, A. Kagi, F. Leung, and L. Smith. Intel Virtualization Technology. Computer, 38(5):48 – 56, May 2005.

[13] http://www.intel.co.uk/content/dam/www/public/us/en/documents/white-papers/page-modification-logging-vmm-white-paper.pdf. (Time accessed: 13th Jan., 2016)

[14] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the Linux virtual machine monitor. In OLS '07: The 2007 Ottawa Linux Symposium, pages 225--230, July 2007.

[15] Paul Barham , Boris Dragovic , Keir Fraser , Steven

Hand , Tim Harris , Alex Ho , Rolf Neugebauer, Ian Pratt , Andrew Warfield, Xen and the art of virtualization, Proceedings of the nineteenth ACM symposium on Operating systems principles, October 19-22, 2003, Bolton Landing,NY,USA [doi>10.1145/945445.945462]

[16] Magenheimer, "Memory overcommit. . . without the commitment," in Proc. Xen Summit, Jun. 2008, pp. 1–3.

[17] A. Gordon, M. R Hines, D. D Silva, M. Ben-Yehuda, M. Silva, and G. Lizarraga, "Ginkgo: Automated, application-driven memory overcommitment for cloud computing," in Proc. RESoLVE: Runtime Environm./Syst., Layering, Virtualized Environ. Workshop, 2011.

[18] J. Heo, X. Zhu, P. Padala, and Z. Wang, "Memory overbooking and dynamic control of Xen virtual machines in consolidated environments," in Proc. IFIP/IEEE Symp. Integr. Manage, Jun. 2009, pp. 630–637.

[19] W. Zhao and Z. Wang, "Dynamic memory balancing for virtual machines," in Proc. ACM Int. Conf. Virtual Execution Environ., Mar. 2009, pp. 21–30.

[20] J. R. Douceur, J. Elson, J. Howell, and J. R. Lorch. Leveraging legacy code to deploy desktop applications on the web. In OSDI, 2008.

[21] https://linuxcontainers.org/lxc/introduction/ (time accessed: 13th Jan., 2016)

[22] Memory hotplug for Xen, 2011, [Online]. Available: https://lkml.org/lkml/2011/3/28/108 (time accessed 13th jan., 2016)

[23] https://wiki.opnfv.org/nfv_hypervisors-kvm (time accessed 13th jan., 2016)

[24] N. Khameesy and H. A. Mohamed "A Proposed Virtualization Technique to Enhance IT Services" in *I.J. Information Technology and Computer Science. 2012,12,21-30*

[25] M. Anjum, M.. A. Haque, N. Ahmad "Analysis and Ranking of Software Reliability Models Based on Weighted Criteria Value" in I.J. Information Technology and Computer Science,2013,02,1-14

Authors' Profiles

Muhammad Shams Ul Haq was born in Pakistan in 1980. He got his Masters and Bachelor degrees in Computer Science from International Islamic University Islamabad Pakistan in 2007 and 2004 respectively. Since 2012, he has been pursuing his PhD degree in School of Computer Science and Technology, BIT, Beijing, China.

His research interests include virtualization, computer security and dynamic memory management

Lejian Liao a professor in School of Computer Science and Technology, Beijing Institute of Technology. He acquired his PhD degree in 1994 and Master degree in 1988 in Chinese Academy of Sciences. His main research areas include distributed Artificial Intelligence and Web Intelligence. He published, as author or co-author, more than 80 papers in international conferences and journals, including some top conferences such as IJCAI, AAAI,and SIGIR..

Ma Lerong is an Associate professor in Yan'an University, China. Currently, he is pursuing his PhD in school of computer Science and Technology, BIT, Beijing, China. His Research interests are in Machine learning, data mining, and information retrieval.

How to cite this paper: Muhammad Shams Ul Haq, Lejian Liao, Ma Lerong,"Virtual Machine Monitor Indigenous Memory Reclamation Technique", International Journal of Information Technology and Computer Science(IJITCS), Vol.8, No.4, pp.11-18, 2016. DOI: 10.5815/ijitcs.2016.04.02