Study Of Intel PML Effectiveness For Virtual Machine Working Set Size Estimation

Stella Bitchebe*
Nice Sophia Antipolis, I3S
stella.bitchebe@unice.fr

Alain Tchana†
Nice Sophia Antipolis, I3S
alain.tchana@unice.fr

Laurent Réveillère‡
Bordeaux University, LaBRI
laurent.reveillere@u-bordeaux.fr

Abstract

Since 2016, Intel started to release processors with a novel virtualization technology feature referred to as *Page-Modification Logging* (PML for short). The latter allows the hypervisor to efficiently track memory pages that a virtual machine (VM) modifies during its execution. PML is intended to improve both VM live migration, VM checkpointing, and VM working set. The two former operations have successfully been experimented using PML. This paper studies for the first time the utilization of PML to efficiently estimate VM working set. Our conclusion is that the current PML design makes it not efficient (not accurate and introduced a significant overhead) for working set estimation. The paper presents a novel design called PRL, for Page Reference Logging, which addresses all PML limitations.

1 Introduction

Virtualization is a technology that uses a hypervisor or virtual-machine monitor (VMM) to manage the underlying hardware resources and provides an abstraction for one or several virtual machines (VMs). Each VM can run a complete operating system (OS) and its applications. Over the last decade, virtualization has become the foundation of datacenters as it allows optimal resource utilization (thus energy saving) by VM packing. However, the observed gain in production datacenters is very far (low) from what is expected [23]. Indeed, VM owners use to over-estimate resources [10, 16] for their tasks. For instance, Jyothi et al. [16] analyzed resource reservation for a 50k nodes production datacenter and found that 75% of jobs were over-provisioned (even at their peak), with 20% of them over 10× over-provisioned.

Several academic researchers and hypervisor vendors have proposed on-demand allocation, also called resource overcommitment [25], as a solution to this issue. It consists in dynamically adjusting the amount of resources allocated to a VM according to its actual needs. This solution requires the datacenter operator to permanently estimate the actual needs of each VM. A wrong estimation would impact user applications, especially when it concerns the memory. The latter is the most critical resource type [18, 23, 26] because its lack can lead to application crashing. This paper focuses on memory allocation on-demand.

The actual memory needs of a VM is called the *working set size* [11], noted WSS. Existing WSS estimation solutions [8, 14, 17, 19, 21, 25], which are all software-based (presented in Section 5), include several drawbacks [21], the most important being: inaccuracy (over/under-estimation), overhead (impact on uver VMs), and intrusivity (requires an important collaboration with the VM owner). For all these reasons, memory overcommitment is not currently used in production datacenters (e.g., Amazon AWS [1]).

Since 2016 [3], in collaboration with VMware, Intel started to release processors (e.g., Broadwell Xeons) equipped with a novel virtualization technology referred to as *Page-Modification Logging* (PML for short) [5] (described in Section 2). When enabled, PML logs the guest physical address (GPA) of memory pages which are modified by a VM during its execution. The ultimate goal of PML is well sumarized by Richard Brunner, the VMware product's chief platform architect [3]:

"*PML ... provides a faster mechanism for a hypervisor to monitor all the memory pages that a guest VM modifies, in order to improve working page set analysis, checkpointing, and even VM migration.*"

Up to the time this paper is written, PML has successfully been used (by Xen, starting from version xen-4.6.0 [7]) to improve VM live migration and checkpointing. From far of our knowledge, no research work has studied the effectiveness of PML for VM WSS estimation. In order words, no one can say at this moment if the current PML design allows the estimation of a VM WSS without the drawbacks of existing (software-based) solutions. This paper strips away the veils. We show that the current PML design is not suitable for WSS estimation because the latter includes five main limitations, presented in Section 3. We propose *Page Reference Logging* (PRL for short), a new design of PML which tackles the identified limitations, presented in Section 4. Section 6 concludes the paper.

2 Background

PML [5] is an Intel's hardware-assisted virtualization feature for monitoring a VM memory activity. It relies on the Extended Page Table (EPT) [24] memory virtualization solution. This section first presents EPT before diving into PML.

*PhD Student
†Professor
‡Professor

EuroDW'19, March 2019, Dresden, Germany
.

2.1 Extended Page Table (EPT)

EPT (also called Nested Page Table) is a hardware-assisted memory virtualization solution (proposed by many chip vendors such as Intel and AMD) which uses a two layer page table (PT). The first layer PT resides in the guest address space and is exclusively managed by its OS, at the rate of one PT per process. This first layer PT thus contains guest physical addresses (GPAs). Each process context switch triggers the setting (by the guest OS) of the CR3 register with the GPA of the scheduled-in process PT. The second PT layer resides in the hypervisor address space and managed by the hypervisor, at the rate of one PT per VM. Every vCPU context switch triggers the setting (by the hypervisor) of an additional CR3 register (noted nCR3 for nested CR3) with the host physical address (HPA) of the scheduled-in VM PT. On TLB miss, the hardware page table walker (HPTW) translates a GVA *va* to the corresponding HPA *hpa* by performing a 2-dimensional page walk as summarized in Fig. 1, interpreted as follows. The HPTW retrieves the address of the current guest's process PT (noted GPT) from the CR3 register (*gpa0*) ($step_1$), and by walking the EPT, it finds the corresponding HPA (*hpa0*), which allows to obtain the physical location of the process PT page. By combining *hpa0* with the offset of *va*, the HPTW obtains the GPA (*gpa1*) of the GPT's page upper directory (PUD). ($Step_1$) is repeated until the HPTW obtains the GPA (*gpa4*) within a page table entry of the GPT. *gpa4* is then translated into a HPA by another EPT walk, to obtain the final result *hpa4-4* in Fig. 1.

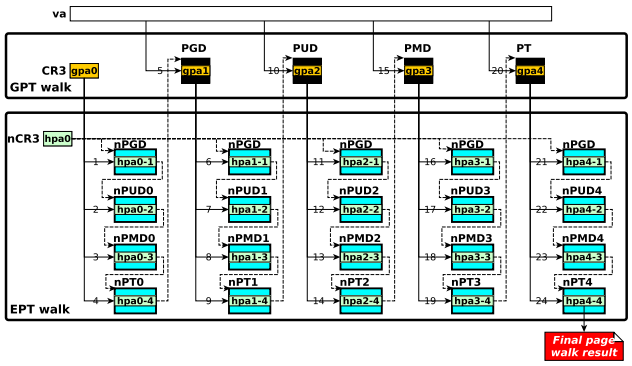

Figure 1. EPT memory virtualization solution for a 4-radix PT.

2.2 PML functioning

PML requires specific changes in the virtual machine control structure (VMCS). It is enabled by setting bit 17 of the *Secondary Processor-Based VM-Execution Controls* of the VMCS. A new 64-bit VM-execution control field called the *PML address* is introduced. *PML address* points to a 4KB aligned physical memory page called *PML log*. The latter is organized in 512 64-bit entries which will stored logged GPAs (see bellow). A new 16-bit guest-state field called the *PML index* is also introduced. The *PML index* is the logical index of the next entry in the *PML log*. Because the *PML log* comprises 512 entries, the *PML index* is typically a value in the range 0-511 (starting from 511). When PML is enabled (bit 17 is set and EPT is enabled) each write that sets a dirty flag in EPT triggers the logging in the *PML Log* of the GPA at the origin of this modification. The *PML index* is decremented after each log. On *PML log* full, the processor raises a VMExit so that the hypervisor comes into play. The logging process will restart when the *PML index* will be reset to 511.

The next section shows how PML can be roughly integrated into the whole process of several virtualization operation improvement (VM live migration, checkpointing and WSS estimation).

3 Motivations

This section presents the five main limitations which make PML ineffective for WSS estimation. For each limitation, we explain why it does not mean one for live migration and checkpointing.

(Limitation L_1) VMExit on logging buffer full should not be handled by the CPU of the VM which WSS is estimated

On log full, the CPU raises a VMExit that stops the execution of the VM which actually uses the CPU, that VM is the one which WSS is computed. A context switch is then performed so that the hypervisor can handle the log full event. VMExit transitions are known to degrade VM performance [12]. Therefore, the introduction of additional VMExits on a VM for estimating its WSS is not acceptable for the VM owner because this virtualization operation is only beneficial for the datacenter operator (she wants to earn more money). Furthermore, the time taken by the hypervisor to handle the *PM log* full event is to increase that impact. In summary, the CPU time stolen from the VM is

$$T_x + T_e + T_h \quad (1)$$

where T_x, T_e and T_h are respectively the time needed for performing a VMExit, VMEnter and log full handler. To illustrate this issue, we ran SPEC CPU2006 [6] applications and the Linux `echo {1..100,000,000}` command (up to ten time) while calculating the number of VMExits and VMEnter related to PML. We also measured the time taken by a simple PML full handler, consisting in copying the content of the log to a bitmap (shared with the privileged VM) for further analysis. The tested VM is configured with a single vCPU which is pined to a dedicated core. Thus, the VM is supposed to have booked 100% of the core. Table 1 presents the results. We can see that the proportion of VMExits related to PML is not zero, up to 2.14%. This number is not negligible because the time of the handler is quite significant, about **257μsec** on average, knowing that the average duration for other VMExit reasons is **32μsec**. This could ecxacerbate tail latencies, which are

Applications	#VMExits	Proportion (%)
astar	972	0.03
bzip2	1214	0.20
echo	10096	2.14
gobmk	244	0.03
h264ref	358	0.05
libquantum	240	0.04
mcf	1840	0.32
omnettp	360	0.06
sjeng	312	0.04

Table 1. Number of VMExits due to PML logging buffer full.

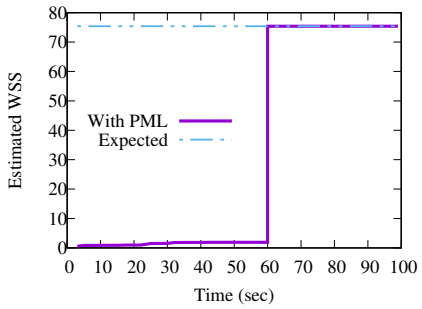

Figure 2. Using PML, WSS estimation is not effective when the workload performs read operations (see the first execution part).

known to be very important [4, 13]. Notice that this impact increases with the complexity of the log full handler.

This L_1 limitation does not mean one neither in live migration nor checkpointing because it has been proven that reducing the CPU time used by a VM during these two operations accelarates them [22].

(Limitation L_2) The log should be more larger than 4KB The size of the log is fixed to 4KB, holding at most 512 GPAs. This size is too small regarding the activity of the majority of today's applications. The consequence is the significant proportion of VMExit related to log full events, exacerbating L_1.

For the same reason as for L_1, a small log is not an issue neither for live migration nor checkpointing improvement.

(Limitation L_3) PML should not log GPAs which point to guest paging structures (called GPT GPAs) Let us consider a write instruction to a guest virtual address *va* from an application address space. Let us consider a TLB miss during the execution of that instruction. During the 2D page walk (described in Section 2.1), every access to a GPT paging structure entry sets the accessed flag (if 0) of that entry (*gpa1-gpa4* entries in Fig. 1). Since the latter actions modify the memory page of the paging structure, they trigger the setting of the dirty flag of the EPT paging structure entries which contain the HPA associated to modified pages. This corresponds to *gpa0-gpa3* in Fig. 1. In respect with this PML function, the initial write to *va* leads to the logging of five GPAs: the first four GPAs belong to the guest paging structure (*gpa0-gpa3*) while the last GPA is the one associated to *va* (*gpa4* in Fig. 1). We claim that the systematic logging of GPT GPAs is not necessary for two reasons. First their logging increases the rate of log full events, thus exacerbating limitation L_1. Second, GPT GPAs can be estimated using an alternative strategy (will be described in the full paper version).

This limitation does not mean one neither for live migration nor checkpointing because for these operations, every updated pages (including GPT pages) should be recorded.

(Limitation L_4) PML should log regardless the value of the dirty flag The WSS counts "hot" pages. A page is said to be hot if it is logged several times (according to a configured threshold) during a short period of time. In respect with the current PML design, a page is logged only once, when its dirty flag is set for the first time. Thus, subsequent write to that page are not logged until the system software switches the dirty flag to zero. Using this PML design, it is likely to include "cold" pages in the WSS, resulting to the over-estimation of the VM memory size, thus resource waste.

The current PML design is sufficient for live migration and checkpointing because for these operations the system software is interested by the tracking of any page which is modified. Thus, counting the number of modification is not necessary.

(Limitation L_5) PML should also log all accessed pages, not only dirty ones Currently, PML only logs GPAs which are at the origin of the dirty flag setting. This means that only modified pages are recorded. However, the WSS of a VM should include both read and write pages. Not considering read pages will lead to under-estimation, thus performance degradation for the VM. Fig. 2 shows the WSS estimation of a synthetic workload whose WSS is constant, 75MB. The workload performs read operations during the first execution part and write operations during the second part. We can see that using PML we are not able to estimate the WSS during the first execution part.

This limitation does not mean one neither for live migration nor checkpointing because these operations are interested to tracking only modified pages.

4 Page Reference Logging

This section presents *Page Reference Logging* (PRL for short), a new PML design which works for both VM WSS estimation, live migration and checkpointing.

Fig. 3 presents the design of PRL, interpreted as follows. PRL allows the processor to function in two exclusive modes including: PRL_{PML} and PRL_{PAML} (PAML stands for Page Access and Modification Logging). To enable PRL_{PML}, the system software should configure the VMCS as follows: (1) enable accessed and dirty flags for EPT, (2) set bit 17 of the *Secondary Processor-Based VM-Execution Controls*, and (3) clear bit 26 of the *Secondary Processor-Based VM-Execution Controls*. The two former actions were already necessary for enabling PML in the original design while the latter action is new. It allows to differentiate the two modes. In fact, PRL_{PAML} is enabled in the same way as PRL_{PML} with the sole difference that the system software should set bit 26 of the *Secondary Processor-Based VM-Execution Controls*. We also give to the system software the possibility to control whether GPT GPAs are logged or not. PRL_{PAML} logs GPT GPAs if bit 27 of the *Secondary Processor-Based VM-Execution Controls* is set. (Notice that we are exploiting the fact that bits 26 to 31 of the *Secondary Processor-Based VM-Execution Controls* are not used in the current PML design.) In PRL_{PML}, the processor works in the same way as the current PML design (see Fig. 3 left), thus allowing PML to satisfy live migration and checkpointing requirements. This section focuses on the description of PRL_{PAML} (see Fig. 3 right), which tackles all requirements related to VM WSS estimation.

First, a VMCS 64-bit (instead of 16-bit as in the original PML design) guest-state field is used for the PRL index, which is the logical index of the next entry in the PRL log. Only bits 0-59 are used while the 4 remaining bits allow to trigger log full events (see bellow). By allowing a large range for the PRL index (from 0 to $2^{60}-1$), the PRL log can have a tremendous amount of entries, thus addressing limitation L_2. Notice that the PRL_{PML} mode can also take benefit of this feature.

Second, a new 16-bit host-state field called "*log full handler CPU*" indicates the index of the CPU to which an interrupt is sent when the PRL log buffer is full. A new 8-bit host-state field called "log full vector" indicates the interrupt vector which will be executed by the target CPU on log full interrupt reception. The system software should indicate the same destination CPU in all VM VMCS. This destination CPU should belong to the privileged VM (noted pVM) so that the latter services as the execution room of the WSS estimation system (see Fig. 3 right, actions (2-5)). By this way, PRL_{PAML} avoids to schedule out the VM which WSS is actually estimated, handling limitation L_1. Recall that impacting the pVM is not an issue because it belongs to the datacenter operator. It already hosts all datacenter management applications (such as monitoring tools) and sometimes device drivers. However, it is not because pVM is used for performing WSS estimation that it should be interrupted unexpectedly. As mentioned above, it hosts datacenter management services which SLA should also be respected. This is why minimizing its interruption still an objective. The utilization of a large PRL log buffer as we do helps achieving this goal.

Third, for every *gpa* which is input of the PRL process (which starts after an EPT walk), the following algorithm takes place:

1. If at least one of the first four bits of *PRL index* (bits 63-60) is not zero, the PRL process ends.
2. If *gpa* equals the GPA within the guest CR3 register and bit 27 of the of the *Secondary Processor-Based VM-Execution Controls* is set (we are facing a GPT GPA, which are configured to not be logged), the PRL process ends.
3. If bit 9 of the GPT entry which contains *gpa* is set and bit 27 of the of the *Secondary Processor-Based VM-Execution Controls* is set (we are facing a GPT GPA, which are configured to not be logged), the PRL process ends. This action and the previous one allows to handle limitation L_2 (not logging of superfluous GPAs).
4. If all bits of *PRL index* are zero, it means that this is the first time that the PRL log buffer has been detected to be full. Then *PRL index* is decremented and an interrupt is sent to the pVM's CPU which is responsible for handling log full events. The PRL process ends without the interruption of the VM which WSS is actually estimated. The processor will start logging again upon the reset of *PRL index* by the system software, especially the log full event handler.
5. Else, *PRL index* is decremented and *gpa* is logged. This is done regardless the value of the dirty flag. By this way, PRL_{PAML} can log both accessed and dirty pages, thus handling limitation L_5. In addition, PRL_{PAML} can log the same page several times, handling limitation L_4.

5 Related work

Several studies have investigated VM WSS estimation.

Committed_AS-based techniques [2] entirely rely on the VM. It considers that the WSS of the VM is given by the `Committed_AS` [2] kernel statistic (`cat /proc/meminfo`). This stat is not accurate for two reasons. First, *Committed_AS* does not take into account the page cache, and thus may cause substantial performance degradation for disk I/O intensive applications [9]. Second, this technique could lead to resource waste since the committed memory is most of the times greater than the actively used memory. **Zballoond** [9] consists in gradually decreasing the VM's memory size until swap-in and refault events start to become non-zero. The VM WSS is the lowest memory size which leads the VM to zero swap-in and refault events. *Zballoond* is very active

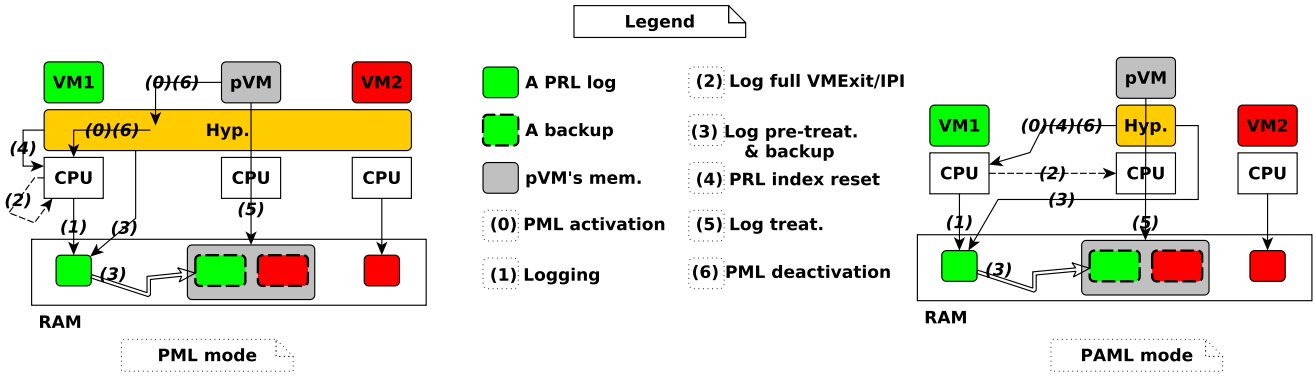

Figure 3. Page Reference Logging (PRL), the new design that we propose. It can can work in two exclusive modes: PML (left) and PAML (right).

in the sense that it performs memory pressure on the VM. **Geiger** [15, 20] monitor the evictions and subsequent reloads from the guest OS buffer cache to the swap device. The VM WSS is computed using an imaginary memory buffer which extends the VM's physical memory (noted m_{cur}). The size of this buffer (noted m_{ghost}) represents the amount of extra memory which would prevent the VM from swapping-out. Knowing the ghost buffer size, one can compute the VM's WSS using the following formula: $WSS = m_{cur} + m_{ghost}$ if $m_{ghost} > 0$. This solution is only able to estimate the WSS when the size of the ghost buffer is greater than zero (the VM is in a swapping state). **VMware** [25] periodically and randomly selects 100 pages from the VM's memory and invalidates them. By so doing, the next access to these pages trap in the hypervisor. The latter counts the number of pages (noted f) among the selected ones which were subject to a non present fault during the previous time interval. The VM WSS is the proportion of trapped pages. This techniques is unable to estimate WSSs greater than the current allocated memory.

6 Conclusion

This paper studied Intel Page Modification Logging (PML), a new feature introduced by Intel in 2016 for improving VM checkpointing, live migration and WSS estimation. Although the effectiveness of PML for the two former operations has been demonstrated by Xen developers, we showed in this paper that the current PML's design is limited for WSS estimation. We have identified five main limitations and presented a new PML design (called Page Reference Logging, PRL for short) that handles these limitations. We are currently implementing PRL in gem5. In addition, we are implementing a complete WSS estimation system which leverages PRL.

References

[1] [n. d.]. AWS re:Invent 2016: Deep Dive on Amazon EC2 Instances, Featuring Performance Optimization Best Practices (CMP301). https://fr.slideshare.net/AmazonWebServices/aws-reinvent-2016-deep-dive-on-amazon-ec2-instances-featuring-performance-optimization-best-practices-cmp301.

[2] [n. d.]. Committed_AS. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/s2-proc-meminfo.html.

[3] [n. d.]. 'Dirty Page Logs' coming to future vSphere release. https://www.theregister.co.uk/2016/05/11/dirty_page_logs_coming_to_future_vsphere_release/.

[4] [n. d.]. Google: Taming the Long Latency Tail - When More Machines Equals Worse Results. http://highscalability.com/blog/2012/3/12/google-taming-the-long-latency-tail-when-more-machines-equal.html.

[5] [n. d.]. Page-Modification Logging for Virtual-Machine Monitor. https://www.intel.com/content/www/us/en/processors/page-modification-logging-vmm-white-paper.html.

[6] [n. d.]. SPEC CPU 2006. https://www.spec.org/cpu2006/ Visited on December 2018.

[7] [n. d.]. [Xen-devel] PML (Page Modification Logging) design for Xen. https://lists.xenproject.org/archives/html/xen-devel/2015-02/msg01305.html Visited on December 2018.

[8] Jui-Hao Chiang, Han-Lin Li, and Tzi cker Chiueh. 2013. Working Set-based Physical Memory Ballooning. (2013), 95–99. https://www.usenix.org/conference/icac13/technical-sessions/presentation/chiang

[9] Jui-Hao Chiang, Han-Lin Li, and Tzi cker Chiueh. 2013. Working Set-based Physical Memory Ballooning. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*. USENIX, San Jose, CA, 95–99. https://www.usenix.org/conference/icac13/technical-sessions/presentation/chiang

[10] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-efficient and QoS-aware Cluster Management. (2014), 127–144. https://doi.org/10.1145/2541940.2541941

[11] Peter J. Denning. 1968. The Working Set Model for Program Behavior. *Commun. ACM* 11, 5 (May 1968), 323–333. https://doi.org/10.1145/363095.363141

[12] Jayneel Gandhi, Mark D. Hill, and Michael M. Swift. 2016. Agile Paging: Exceeding the Best of Nested and Shadow Paging. (2016), 707–718. https://doi.org/10.1109/ISCA.2016.67

[13] Md E. Haque, Yuxiong He, Sameh Elnikety, Thu D. Nguyen, Ricardo Bianchini, and Kathryn S. McKinley. 2017. Exploiting Heterogeneity for Tail Latency and Energy Efficiency. (2017), 625–638. https://doi.org/10.1145/3123939.3123956

[14] Stephen T. Jones, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2006. Geiger: Monitoring the Buffer Cache in a Virtual Machine Environment. (2006), 14–24. https://doi.org/10.1145/1168857.1168861

[15] Stephen T. Jones, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2006. Geiger: Monitoring the Buffer Cache in a Virtual Machine Environment. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XII)*. ACM, New York, NY, USA, 14–24. https://doi.org/10.1145/1168857.1168861

[16] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayanamurthy, Alexey Tumanov, Jonathan Yaniv, Ruslan Mavlyutov, Íñigo Goiri, Subru Krishnan, Janardhan Kulkarni, and Sriram Rao. 2016. Morpheus: Towards Automated SLOs for Enterprise Clusters. (2016), 117–134. http://dl.acm.org/citation.cfm?id=3026877.3026887

[17] Jinchun Kim, Viacheslav Fedorov, Paul V. Gratz, and A. L. Narasimha Reddy. 2015. Dynamic Memory Pressure Aware Ballooning. (2015), 103–112. https://doi.org/10.1145/2818950.2818967

[18] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, and Thomas F. Wenisch. 2009. Disaggregated Memory for Expansion and Sharing in Blade Servers. (2009), 267–278. https://doi.org/10.1145/1555754.1555789

[19] Pin Lu and Kai Shen. 2007. Virtual Machine Memory Access Tracing with Hypervisor Exclusive Cache. , Article 3 (2007), 15 pages. http://dl.acm.org/citation.cfm?id=1364385.1364388

[20] Pin Lu and Kai Shen. 2007. Virtual Machine Memory Access Tracing with Hypervisor Exclusive Cache. In *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference (ATC'07)*. USENIX Association, Berkeley, CA, USA, Article 3, 15 pages. http://dl.acm.org/citation.cfm?id=1364385.1364388

[21] Vlad Nitu, Aram Kocharyan, Hannas Yaya, Alain Tchana, Daniel Hagimont, and Hrachya Astsatryan. 2018. Working Set Size Estimation Techniques in Virtualized Environments: One Size Does Not Fit All. *Proc. ACM Meas. Anal. Comput. Syst.* 2, 1, Article 19 (April 2018), 22 pages. https://doi.org/10.1145/3179422

[22] Vlad Nitu, Pierre Olivier, Alain Tchana, Daniel Chiba, Antonio Barbalace, Daniel Hagimont, and Binoy Ravindran. 2017. Swift Birth and Quick Death: Enabling Fast Parallel Guest Boot and Destruction in the Xen Hypervisor. (2017), 1–14. https://doi.org/10.1145/3050748.3050758

[23] Vlad Nitu, Boris Teabe, Alain Tchana, Canturk Isci, and Daniel Hagimont. 2018. Welcome to Zombieland: Practical and Energy-efficient Memory Disaggregation in a Datacenter. , Article 16 (2018), 12 pages. https://doi.org/10.1145/3190508.3190537

[24] Rich Uhlig, Gil Neiger, Dion Rodgers, Amy L. Santoni, Fernando C. M. Martins, Andrew V. Anderson, Steven M. Bennett, Alain Kagi, Felix H. Leung, and Larry Smith. 2005. Intel Virtualization Technology. *Computer* 38, 5 (May 2005), 48–56. https://doi.org/10.1109/MC.2005.163

[25] Carl A. Waldspurger. 2002. Memory Resource Management in VMware ESX Server. *SIGOPS Oper. Syst. Rev.* 36, SI (Dec. 2002), 181–194. https://doi.org/10.1145/844128.844146

[26] Wm. A. Wulf and Sally A. McKee. 1995. Hitting the Memory Wall: Implications of the Obvious. *SIGARCH Comput. Archit. News* 23, 1 (March 1995), 20–24. https://doi.org/10.1145/216585.216588