Extending Intel PML for Hardware-Assisted Working Set Size Estimation of VMs

Stella Bitchebe
ENS Lyon, LIP
Lyon, France
celestine-stella.ndonga-bitchebe@ens-lyon.fr

Djob Mvondo
Univ. Grenoble Alpes, LIG
Grenoble, France
mvondodb@univ-grenoble-alpes.fr

Alain Tchana
ENS Lyon, LIP
Lyon, France
alain.tchana@ens-lyon.fr

Laurent Réveillère
Univ. Bordeaux, LaBRI
Bordeaux, France
laurent.reveillere@u-bordeaux.fr

Noël de Palma
Univ. Grenoble Alpes, LIG
Grenoble, France
noel.depalma@univ-grenoble-alpes.fr

Abstract

Intel page modification logging (PML) is a hardware feature introduced in 2015 for tracking modified memory pages of virtual machines (VMs). Although initially designed to improve VMs checkpointing and live migration, we present in this paper how we can take advantage of this virtualization technology to efficiently estimate the working set size (WSS) of a VM.

To this end, we first conduct a study of PML with the Xen hypervisor to investigate its performance impact on VMs and the accuracy of a WSS estimation system that relies on the current version of PML. Our three main findings are as follows. (1) PML reduces by up to 10.18% the time of both VM live migration and checkpointing. (2) PML slightly reduces the negative impact of live migration on application performance by up to 0.95%. (3) A WSS estimation system based on the current version of PML provides inaccurate results. Moreover, our experiments show that write-intensive applications are negatively impacted, with up to 34.9% of performance degradation, when using PML to estimate the WSS of a VM that runs these applications.

Based on the aforementioned findings, we introduce page reference logging (PRL), an extended version of PML that allows both read and write memory accesses to be tracked without impacting user VMs, thus more suitable for WSS estimation. We propose a WSS estimation system that leverages PRL and show how it can be used in a data center exploiting memory overcommitment. We implement PRL and the underlying WSS estimation system under Gem5, a popular open-source computer architecture simulator. Evaluation results validate the accuracy of the WSS estimation system and show that PRL does not incur more performance degradation on user's VMs.

CCS Concepts: • **Software and its engineering → Virtual machines**; • **Hardware → Simulation and emulation**;

Keywords: Virtualization, Virtualization Technology, Page Modification Logging, Virtual Machine Control Structure, Working Set Size

1 Introduction

Virtualization has become the foundation of data centers because it allows resource mutualization between several clients while ensuring isolation. Virtualization also provides adequate support for administration, including resource management. In this context, memory page tracking is a key mechanism that traces memory access for virtual machines, so that the hypervisor (or virtual machine monitor) can improve memory management for the various services it implements. Memory page tracking is at the heart of several essential tasks such as checkpointing [62] for recovery after failure, live migration [28] for maintenance and dynamic packing, and working set size (WSS)[1] estimation [31] for memory overcommitment [12] and fast restore [62].

WSS estimation is an essential task for data center operators because it allows, among other things, memory overcommitment [50, 52] (by periodically adapting the memory size of the virtual machine (VM) according to its real needs), fast restore [62] and efficient processor cache partitioning [9]. Regarding memory overcommitment, for example, its implementation and adoption are necessary for the following reasons. First, VM owners use to over-estimate resources [30, 36] for their tasks. Jyothi et al. [36] analyzed the resource reservation for a 50k-node production data center and found that 75% of jobs were over-provisioned (even at their peak), with 20% of them over 10 times over-provisioned. E. Cortez et al. [29] made similar observations in recent traces of the Microsoft Azure cloud. Second, some types of cloud-native workloads are fundamentally based on the dynamic management of overcommitted VMs. In particular, this is the case for serverless or function as a service (FaaS) systems,

[1]The working set refers to the set of memory pages a process/OS/VM is using at a given time[31].

whose design and pricing model are intrinsically linked to an aggressive packing of hundreds or thousands of micro-VMs on the same physical machine [24, 59]. Third, memory is a finite resource whose evolution does not follow that of other resources (especially the CPU), so researchers talk about the problem of the memory wall [43, 54].

The most widely used approach for WSS estimation through memory page tracking relies on present bit invalidation. Such an approach may lead to severe performance degradation caused by the generated page faults. It is particularly true when a significant amount of pages have to be tracked, as in WSS estimation [25, 34, 39, 44, 50, 58]. We assess this issue using a synthetic application that parses an array. The present bit is invalidated every second for all memory pages of the VM. We measure up to 96.22% of performance degradation for a VM with 1GB of memory size.

We can, however, take advantage of hardware virtualization features available for memory page tracking, specifically Intel page modification logging (PML). In 2015, Intel started to release processors equipped with this virtualization technology feature that allows the hypervisor to track all memory pages modified by a VM during its execution. When enabled, the memory management unit (MMU) logs in a 4KB page (called the PML logging buffer) in RAM, all guest physical addresses (GPAs) that led to the setting of the dirty bit in the extended page table (EPT) during page walks.

Contributions. We conduct a comprehensive study to assess the effectiveness of PML, and its impact on the performance of user applications. Our main findings are summarized as follows. First, PML reduces by up to 10.18% the time of both VM live migration and checkpointing. Second, PML slightly reduces the negative impact of live migration and checkpointing on application performance by up to 0.95%, which is not negligible for tail latencies [37]. Third, PML can be used for WSS estimation, but needs to be improved to provide an accurate estimation, summarily because read accesses are not tracked and hot pages cannot be identified.

In fact, the current design of PML focuses on write workloads. It only logs the GPA of a page once, even if the page is accessed several times. Therefore, cold pages are likely to be counted in the working set, over-estimating the latter, which leads to memory waste. In addition, PML would incur an unacceptable overhead for the VM whose WSS is estimated. Indeed, when the PML logging buffer is full (after 512 GPAs logged), the CPUs of the VM whose WSS is computed trigger a VMExit. The handler of that VMExit consumes CPU time which is taken from the VM's CPU quota. We measure up to 34.9% of performance degradation.This would not be acceptable for cloud users because they are not the beneficiaries of the WSS estimation which is executed for the needs of the data center operator.

We introduce page reference logging (PRL), an extended version of PML to track both read and write working set without impacting VM performances. PRL can be used in two exclusive modes: PRL_{PML} and PRL_{PAML}. PRL_{PML} is similar to the current PML functioning, making PRL effective for live migration and checkpointing. In contrast, PRL_{PAML} focuses on WSS estimation. In PRL_{PAML} mode, read accesses are taken into account, and several recordings of the same page are also possible, allowing hot pages tracking. In addition, PRL_{PAML} avoids performance overhead on user VMs for the following reason. VMExits related to PRL_{PAML} are redirected to the *dom0*[2] CPUs. *Dom0* is responsible for running VM administration services. Therefore, it makes sense to use it to host WSS estimation computations since the data center operator is the main beneficiary of this task. Technically, when the PRL_{PAML} logging buffer is full, the actual CPU (which is running the user's VM whose WSS is computed) sends an inter-processor interrupt (IPI) to one of the *dom0*'s CPU, thus raising a VMExit on it. By redirecting VMExits related to PRL_{PAML} to the *dom0*, the user VM can continue its execution while handling these VMExits (unlike in the current PML design), thus avoiding the negative impact on user VMs. The handler of this IPI identifies hot pages and makes them available to a WSS estimation system. We describe an implementation of PRL in Gem5 [23], a popular computer architecture simulator.

We present a prototype implementation of a WSS estimation system that uses PRL in the context of the Xen hypervisor. Using both real (HPL Linpack [8], BigDataBench [7]) and synthetic applications, we evaluate and compare our solution with an implementation of a software-based solution, precisely the VMware's WSS estimation solution described in [58][3], following the same evaluation methodology as the work of Nitu et al in SIGEMETRICS 2018 [50]. The evaluation results confirm that: (1) our solution is accurate, (2) our solution has no impact on user VMs, (3) our solution is not intrusive (no modification of the guest OS is required), unlike most state-of-the-art solutions [25, 34, 39, 44, 50, 58].

Our complete prototype, experimentation scripts, and workloads will be made available open source with the non anonym- ized version of this paper.

Outline. The paper is organized as follows. We describe the results of the comprehensive study of PML in Section 2. We introduce PRL and a WSS estimation system which leverages PRL in Section 3. We evaluate the resulting system in Section 4, present related work in Section 5 and conclude in Section 6.

[2] *dom0* is the privileged VM (noted pVM) in Xen [19]. In fact most virtualization systems rely on a pVM, *host OS* in KVM [1], *parent partition* in Hyper-V [2] and *Service Console* in VMware [4, 5].

[3] To the best of our knowledge, [58] is the only publication made by VMware concerning their WSS estimation.

2 Understanding PML

PML is an Intel Virtualization Technology feature that extends the capability of a hypervisor, allowing it to efficiently track or monitor the guest physical memory pages modified by a VM during its execution. PML relies on Extended Page Table (EPT) [57] hardware support for MMU virtualization, and requires specific changes in the virtual machine control structure (VMCS)[4] [3]. A new 64-bit VM-execution control field called *PML address* is introduced. The *PML address* points to a 4KB aligned physical memory page called *PML logging buffer*. This buffer is organized into 512 64-bit entries that store the logged GPAs. A new 16-bit guest-state field called *PML index* is also introduced. The *PML index* is the logical index of the next entry in the logging buffer. Because the buffer includes 512 entries, the *PML index* has values ranging from 0 to 511, starting at 511. When PML is enabled, each write instruction that sets a dirty flag in the EPT during a page walk triggers the logging of the corresponding GPA (concnerned by the dirty flag entry). The *PML index* is then decremented by 1. Whenever the *PML logging buffer* is full, the processor triggers a VMExit and the hypervisor comes into play. The logging process restarts after the *PML index* is reset to 511. The actions taken by the hypervisor in response to this VMExit depend on the objective (*e.g.*, VM live migration).

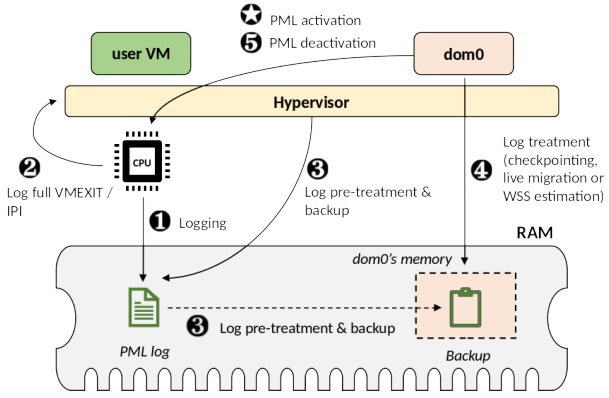


Figure 1. Basic use of PML to improve a virtualization operation (*e.g.*, live migration, WSS estimation).

2.1 PML functioning

Figure 1 illustrates the general workflow of PML when used to improve a virtualization operation (*e.g.*, live migration). The figure shows on the one side the user's VM (in green) that is the target of the virtualization operation and on the other side the *dom0*, which runs the system that implements this virtualization operation. The execution of that system generally begins with the activation of PML for the target user's VM ✪. Then, the CPU of that VM can start logging GPAs ❶. When the *PML logging buffer* is full, the CPU triggers a VMExit trapped into the hypervisor ❷. The handler of that VMExit performs a certain task (*e.g.*, copy the content of the *PML logging buffer* to a larger buffer that is shared with the *dom0* ❸). Then, the *PML index* is reset to 511 and the VM resumes (VMEnter). The system implementing the virtualization operation (in the dom0) operates periodically on the results generated by the log full handler ❹. This is done as part of the virtualization operation, *e.g.*, remigrate dirty pages in case of live migration. When the virtualization operation is complete, PML is disabled ❺.

[4]A logical processor uses VMCSs [15] for its VMX (Virtual Machine Extension) operations. The hypervisor uses a different VMCS for each vCPU of the VMs that it supports or manages.

2.2 PML Study

To better understand PML, we conduct a study to investigate both its effectiveness and its performance impact on user applications. We target live migration and WSS estimation operations. We do not elaborate on checkpointing because live checkpointing, which could take benefit of PML, is not implemented in current hypervisors. We carried out the experiments on a laptop with the following characteristics: Single socket Intel(R) core (TM) i7-3768, 16GB memory, 500GB SSD, 4-way 64 TLB entries. We used Xen 4.7 as the hypervisor and Linux 4.15.0 for the guest kernel. For the applications that run inside the virtual machine, we used HPL Linpack [8], BigDataBench [7] (read, write and sort applications, 10GB data set size), and a synthetic application for which the code structure is shown in Listing 1. The synthetic application consists in parsing an array several times during a period. Each array entry points to a 4KB data structure (size of a memory page). The type of operation (read or write) performed on an array entry is decided according to a write intensity parameter (wi) which represents the proportion of write operations. Unless otherwise indicated, the array uses 400MB of memory and the VM has one vCPU and 1GB of memory for the synthetic application, and four vCPUs and 12GB of memory for the macro benchmarks.

```
1  /* Main variables description
2   * wi: proportion of write operations
3   * tab: array to be processed
4   * PERIOD: duration of each iteration
5   * SIZE: number of tab memory pages
6   * nbOps: # of read and write operations computed
7   * throughput: mean number of operations per nanoseconds
8   * ns: duration in milliseconds
9   */
10 #define PERIOD ...
11 #define SIZE ...//SIZE*size_page=size_tab
12 struct page{
13   unsigned long entry[512];
14 };
15 void synthetic_workload(int wi){ //write intensity
```

```
16  /*
17   * declare & initialise variables
18   * nbOps, throughput, ns, ...
19   */
20  void *tab;
21  struct page *temp;
22  struct timespec start, end;
23  clock_gettime(..., &start);
24  do{
25    posix_memalign(&tab, ...);
26    for(i = 0; i < SIZE; i++){
27      temp = (tab + size_of_page*i);
28      op = rand() % 100;
29      if(op < wi)
30        temp->entry[...] = ...;//write operation
31      else
32        read = temp->entry[...];//read operation
33      nbOps++;
34    }
35    clock_gettime(..., &end);
36    //convert duration (from start to end)
37    //in nanoseconds
38    ns = ...;
39    throughput = (nbOps-nbOps_prev)/(ns-ns_prev);
40    ns_prev = ns;
41    nbOps_prev = nbOps;
42  }while(ns < PERIOD);
43  free(tab);
44 }
```

Listing 1. Synthetic application skeleton. Its performance metric is the number of operations per nanosecond.

2.3 PML-based VM migration

In Xen, the heart of live migration is mainly implemented through the function `int save()` in the file `tools / libxc / xc_sr_save.c`). The use of PML is thus limited to this memory saving phase. We compare the use of PML with the classical memory page tracking approach which consists of write protecting memory pages so that next writes lead to page faults. We use the synthetic application as a baseline for this evaluation because its behavior is predictable compared to the macro-benchmark.

We consider two metrics: *performance* the performance of the user application during migration and *duration* the duration of the `save()` method execution. *performance* checks whether PML reduces or increases the negative impact of these operations on the application while *duration* indicates whether PML accelerates migration or not. Two successive live migration operations are performed while running the application during a certain period, with different proportions of write operations.

Figure 2 and table 1 present the results for *performance*. We observe that live migration anyway negatively impacts the performance of the application, as illustrated by the two descending peaks in all curves. However, PML slightly minimizes this impact by 0.06% to 0.95%. The magnitude of the reduction depends on the write intensity of the workload. Indeed, although the use of PML for a read-intensive workload reduces very slightly the performance of the application, its advantages are more important for the write-intensive workload.

Figure 3 presents the results for *duration*. We observe that PML reduces the execution time of method `save()` during live migration by 0.98% to 10.18%. In particular, read-intensive applications migrate much faster when PML is used. This is due to two main reasons. First, when using PML if a page has not been logged (the GPA of the page is not present in the *pml logging buffer*) this means it has not been modified and then can immediately be migrated. The hypervisor does no longer need to invalidate it first. Second, as the workload performs less write operations there are fewer logged GPAs, thus less *logging buffer* walk operations by the hypervisor and then a faster migration. Note that it is very important to speed up live migration because it allows you to quickly free a machine for maintenance, quarantine a corrupted VM, *etc.*

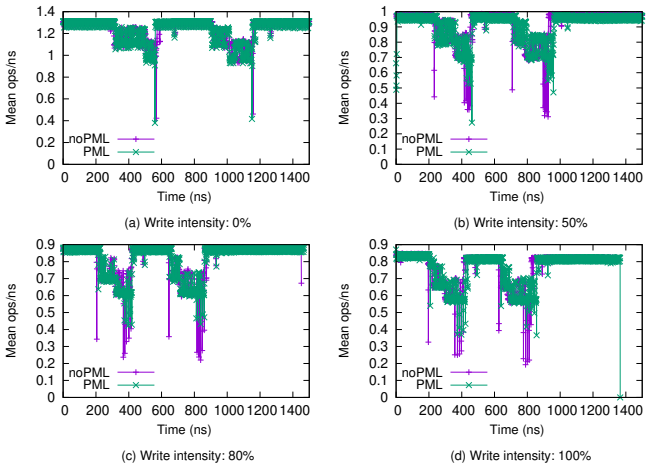


Figure 2. Operations per nanosecond while two live migrations are performed. We run this experiment with different write intensity values (for the synthetic application): 0%, 50%, 80%, and 100%.

Write intensity (%)	0	50	80	100
Improvement (%)	6×10^{-2}	0.14	0.39	0.95

Table 1. PML benefits during live migration. We compare the performance degradation of the running application during live migration, with and withouth PML.

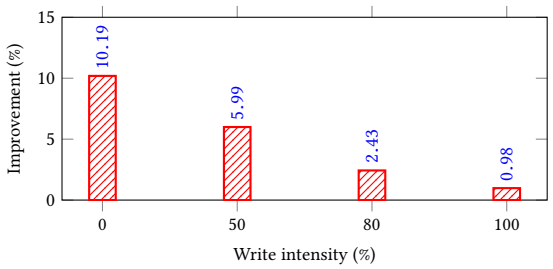


Figure 3. Execution time improvement of the method `save()` using PML, compared to the version without PML.

2.4 PML-based WSS estimation

A WSS estimation system based on PML would work as follows. Once PML is enabled for the VM, the system collects GPAs until no new GPAs are visible in the logging buffer.The estimated WSS would then be the total number of different GPAs that have been collected. We implemented this system in the Xen hypervisor. However, such a system fails to provide an accurate WSS estimation. First, because PML only tracks write WSS (hence the name *page **modification** logging*). Second, PML does not track hot pages (even write ones). A page is said *hot* if it is referenced several times over a short period. According to the current PML design, an accessed page is, however, logged only once. Using this design for WSS estimation, it is not possible to distinguish between *hot* and *cold* pages, which leads to an overestimation of the actual memory requirements of the virtual machine. To evaluate this limitation, we modified the synthetic application by adding a *for* loop at the beginning which modifies all the entries in the array (which is xMB). The remaining application code works on a small portion of the array (noted y, with $y < x$). Although the correct value of the WSS estimation is y, the system reports x, thus wasting memory. Third, PML degrades the performance of the VM whose WSS is estimated. In fact, the handling of PML logging buffer full events should not be done by the CPU of the VM whose WSS is estimated. Indeed, depriving the user's VM of its CPU quota is unfair because the WSS estimation is only beneficial for the data center operator. One could legitimately say that this limitation is also true for live migration. However, it has been proven [51] that a slight reduction in CPU time used by the migrated VM accelerates live migration (as we previously observed in figure 3). We measured the overhead of the current PML design to estimate the WSS of applications from BigDataBench [7] (read, write and sort applications) and HPL Linpack [8][5]. For BigDataBench applications, the input dataset is 10GB. We run each application with and without PML and calculate the overhead, as shown in Figure 4. Read-intensive applications are not affected by the use of PML because it only tracks page modifications. However, other workloads such as HPL Linpack are significantly impacted by the use of PML, with a performance degradation of up to 34.9%.

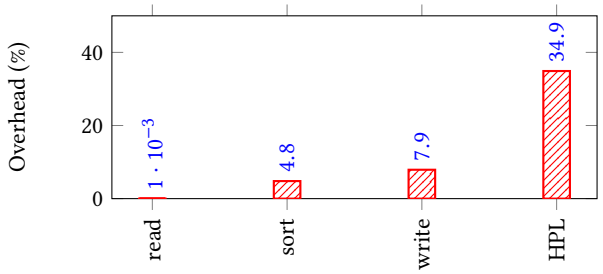


Figure 4. Impact of PML when used for WSS estimation. We plot the overhead for applications in terms of FLOps (FLoating point Operations per second).

We summarize the main conclusions of this study on PML as follows. First, PML reduces VM live migration time. Second, PML slightly reduces the application performance overhead during live migration. Third, we cannot use PML with its current design for efficient WSS estimation, because a WSS estimation system based on it will fail to accurately estimate the whole working set of a VM.

3 Page Reference Logging

We introduce page reference logging (PRL), an extended version of PML to facilitate efficient WSS estimation.

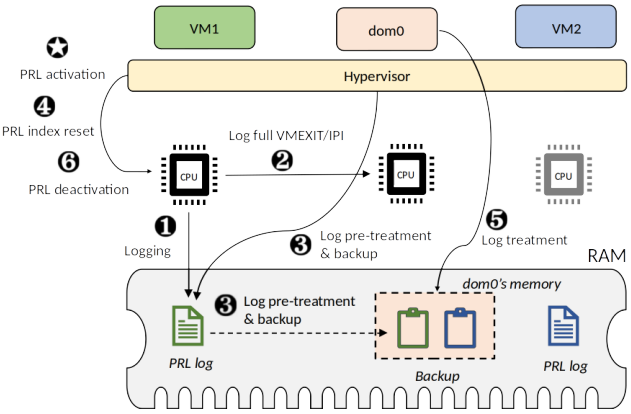


Figure 5. Overview of PRL functioning.

3.1 PRL functioning

Figure 5 illustrates the general functioning of PRL. Let's consider a user's VM ($VM1$ in green) that is the target of the WSS estimation operation, and the *dom0* which runs the WSS estimation system. The execution of that system generally begins with the activation of PRL for the target user's

[5]HPL is a High-Performance benchmark implementation whose code solves a uniformly random system of linear equations

VM ✪ so that the CPU of that VM can start logging GPAs ❶. When the *PRL log buffer* is full, the VM's CPU sends an IPI to a dedicated *dom0*'s CPU (which will be responsible for computing the working set of the VM) ❷. Then the hypervisor copies the content of the *PRL log buffer* to a larger buffer that is shared with the *dom0* ❸ (while the VM continues its execution with no interruption), and the *PRL index* is reset to 511. After what the logging process restarts ❹ and in the meantime the WSS estimation system operates on the results generated by the log full handler ❺.

3.2 PRL architectural design

A processor supporting PRL can be used in two exclusive modes: PRL_{PML} and PRL_{PAML}. The first mode mimics the current PML functioning (Section 2.1), making PRL effective for live migration and checkpointing, as it is for PML. In contrast, PRL_{PAML} focuses on WSS estimation.

As for PML specific changes are needed in VMX to support PRL. To activate PRL_{PAML}, the system software must set a new bit of the *Secondary Processor-Based VM-Execution Controls* called *PRL enable* (similarly the bit 17 of the *Secondary Processor-Based VM-Execution Controls* must be set to activate PML). 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 that will be executed by the target CPU upon reception of a *log full* interrupt. This destination CPU must belong to the dom0, which serves as the execution room of the WSS estimation system (Fig. 5, ❷ - ❺). In this way, PRL_{PAML} avoids scheduling out the VM whose WSS is estimated. Remember that the dom0 belongs to the data center operator, so using it for WSS estimation makes sense. For each GPA which is the input of the PRL process (which begins after an EPT step, on TLB miss), the following algorithm takes place:

1. *PRL index* is initialized to 511.
2. If the *PRL index* value is 0, this means that the PRL log buffer has been detected as full. In such a case, the *PRL index* is decremented and an interrupt is sent to the processor of the dom0 which is responsible for handling log full events. The PRL process ends without interrupting the VM whose WSS is estimated. The processor restarts the logging when *PRL index* is reset by the system software.
3. If the *PRL index* is negative, this means that the log full event handler is running. The GPA is therefore not logged and the PRL process ends. Some may argue that the PRL process will miss some GPAs when processing the event handler, we claim that it does not affect the WSS estimation. Indeed, if a missed GPA belongs to the working set, it is likely to be seen in the near future (after reactivation of the PRL mechanism) because it is hot. Otherwise, the GPA is cold and its loss does not change the WSS estimation. The results of the evaluation described in Section 4 support our claim.
4. Otherwise, the *PRL index* is positive (and less than 511) and is decremented and the GPA is logged. This is done regardless of the value of the dirty flag (in contrast to the current PML where the GPA is logged only if dirty bit is set). In this way, PRL_{PAML} can log both accessed and modified pages. Besides, PRL_{PAML} can log the same page access several times.

3.3 PRL log full event handling

When the PRL log buffer is full, the processor sends (through its LAPIC[6]) an IPI to the CPU of the dom0 that was designated at VMCS configuration time. The LAPIC is configured with the identifier of the target CPU. We have introduced a new interrupt vector that points to the log full event handler. This mechanism is similar to *Lightweight inter-core notifications* introduced by Jefrey C. Mogul et al. [47]. Since the log full event handler must run in the VMX root mode because it processes VMCS data structures, the target processor must trigger a VMExit upon receipt of the IPI. This behavior is enforced by setting bit 0 of the *Pin-Based VM-Execution Controls* of all dom0's vCPUs. Using this configuration, any external interrupt sent to any CPU of a dom0 triggers a VMExit.

When called, the handler first masks the interrupt related to the log full event. Then, it copies the contents of all the PRL log buffers which are full (for all VMs that have sent an interrupt) to larger buffers. Note that a large buffer (also called a cumulative buffer) is allocated to each VM by the system software and that this buffer is unique per VM, even for a multi-vCPU VM. During the copy phase, the handler accumulates the number of occurrences of each GPA in the PRL log buffer. In this way, the WSS estimation system can identify hot pages. Once the copy of PRL log buffers ends, the handler resets the *PRL index* of all VMCSs detected as full to its initial value (511). Note that modifying a *PRL index* only affects its VMCS memory region, not the internal registers of the corresponding processor. Indeed, the synchronization of the VMCS memory region and the processor registers is not automatic. To enforce this, we introduce a new instruction that updates the internal VMCS state of a specific processor using its corresponding VMCS memory region. The execution of the handler ends with the unmasking of the log full interrupt. This algorithm handles several log full events using a single generated interrupt. It is inspired by the New API (NAPI) implemented in modern Linux kernels to handle network packet reception [55].

3.4 PRL-based WSS estimation system

We implemented a WSS estimation system that leverages PRL. Our system launches as many WSS estimation processes as the number of tenant VMs. Each process calculates WSS

[6]Local APIC (Advanced Programmable Interrupt Controller)

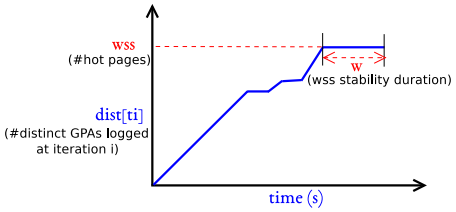


Figure 6. Estimation of the number of hot pages.

using the equation:

$$WSS = hotPages \times pageSize + \varepsilon \quad (1)$$

where $hotPages$ is the number of computed hot pages, $pageSize$ is the size of a memory page, and ε is the size of the guest kernel footprint (the minimal amount of memory needed by the kernel).

Hot pages. The number of hot pages is based on the GPAs that are logged by PRL. Its computation takes three parameters as input:

- τ: the minimal number of times a page's GPA must be logged for this page to be considered hot;
- ω: the stability duration used to determine if the VM has already covered its working set;
- μ: the observation interval.

The values of these parameters are defined by the external entity which launches the WSS estimation system. Different approaches exist to determine these values [64].

Let *Cbuff* be the cumulative PRL buffer of the VM whose WSS is calculated. The estimation of the number of hot pages works iteratively as follows. For each iteration i, the number of distinct GPAs present in *Cbuff* that have been logged more than τ times is computed and stored in an array $dist[t_i]$ (where t_i is the iteration time). The loop ends when $dist[t_i] - dist[t_i - \omega] = 0$, which means that the VM has touched/referenced all memory pages belonging to its current working set. Otherwise, the process goes to sleep for μ seconds before continuing the iteration. Figure 6 illustrates how this algorithm works. We can see the evolution of $dist[t_i]$ over time which corresponds to an increasing monotonic function.

Guest kernel footprint. The value of the guest kernel footprint ε depends on the guest kernel binary. It is estimated once by the data center operator for each kernel binary. The following algorithm can be used:

1. Starts a 2GB VM from the kernel binary;
2. Initialize ε and $currentMem$ (an auxiliary variable) to 2GB;
3. Set $currentMem$ to $95\% \times \varepsilon$;
4. Change the VM memory size to $currentMem$;
5. If the VM crashes then stop the algorithm and return ε;
   - Else set ε to $currentMem$ and go to step 3.

We provide a tool that automates these steps for a machine virtualized with Xen hypervisor.

4 PRL Evaluation

Our evaluation of PRL mainly focuses on WSS estimation. We do not evaluate PRL for live migration as the PRL_{PAML} mode is strictly similar to the current PML design (see Section 2.3 for in-depth PML live migration evaluations).

Our assessment of the WSS estimation system based on PRL covers two aspects: accuracy and overhead. The accuracy is the capability to accurately estimate the WSS of a VM. The overhead is the impact on the VM whose WSS is estimated and the number of resources consumed by the WSS estimation system inside the dom0. We used empirical values for the parameters of the WSS estimation algorithm (let's recall that these inputs are decided and defined by the external entity which launches the WSS estimation system and that different approaches exist to determine them [64]). We set τ to 50, which means that the GPA of a page needs to be logged at least 50 times for this page to be considered hot[7]. We set μ to 30s, which means that the algorithm iterates every 30s. We used this value following the VMWare technique which also iterates every 30s. We set ω to 120s, which corresponds to the observation period.

4.1 Experimental environment

We use the same experimental environment as the one presented in Section 2.2 for the assessment of PML. In addition, we use the hardware simulator Gem5 [23] to emulate a machine that implements PRL. We chose Gem5 because it is a very popular hardware simulator (specifically adapted for research), which was used by 90+ research papers at the time of writing [6]. Although Gem5 allows the execution of a complete Linux distribution, we have extended it to simulate a virtualized system. This improvement consists of the addition of the Extended Page Table (EPT) support, the extension of the hardware page table walker so that it performs a 2D page walk through the EPT, and the implementation of PRL/PML logging mechanisms.

```
1  /*
2   * Main variables description
3   * tab: array to be processed
4   * PERIOD: application duration
5   * SIZE: number of tab memory pages
6   * time: total time in seconds
7   */
8  #define PERIOD ...
9  #define SIZE ... //SIZE * 4096 = size_of_tab
10
11 void synthetic_app()
```

[7] We use a small value to evaluate a worst-case scenario because a small τ value increases the buffer's log filling rate, thus, more log full events.

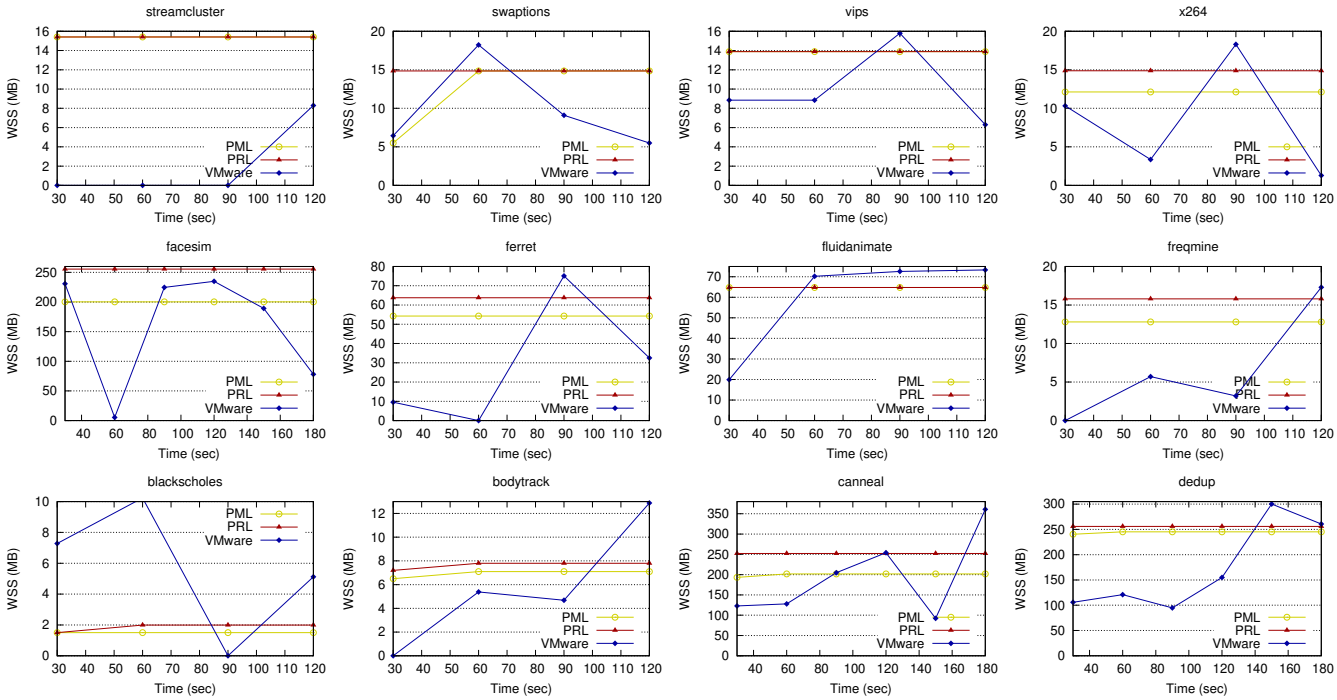


Figure 7. WSS estimation of PARSEC applications using PRL.

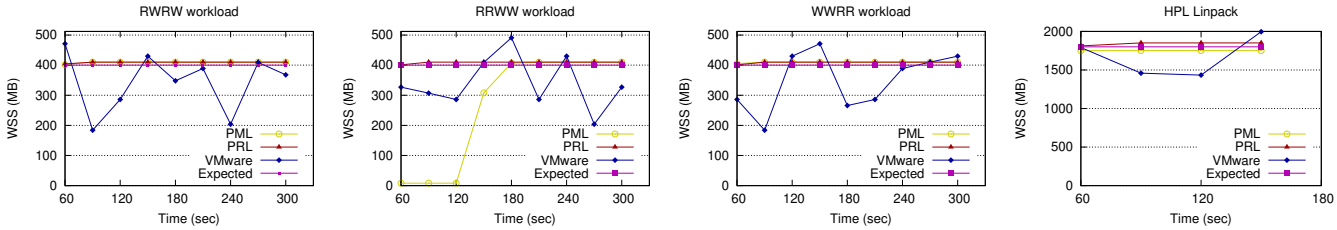


Figure 8. Efficiency of PRL compared with PML and VMware. Expected is the baseline. It corresponds to the WSS of the application (400MB).

```
{
  do{
    for(int j=0;j<SIZE;j++)
      /* Operate on tab (read/write) */
      operation(tab[j]);
      /* Compute the time */
      time = ...;
  }while(time < PERIOD);
  free(tab);
}
```

Listing 2. Second synthetic application skeleton.

The emulation methodology is as follows. We run the application under Gem5 and we collect the logged GPAs, including timestamps. The collected traces are then replayed inside the dom0 of a real virtualized environment that runs the WSS estimation system. A dom0's CPU (noted CPU_0) is dedicated to the latter. The other dom0's CPUs run processes that replay the traces, thus mimicking the functioning of a real machine equipped with PRL. Each process replaying the traces sends an IPI to CPU_0 each time it plays N traces, N beeing the size of the log buffer. This emulation is similar to the actual operation of a PRL compatible machine, as shown in Figure 5. To be fair, we also assessed the accuracy of PML and VMware solution using the same methodology. As a reminder, the VMware WSS estimation solution consists in periodically selecting (every 30 seconds) a sample of 100 memory pages whose present bits are invalidated. The proportion of pages, among these selected 100 pages, which will cause a page fault represents the WSS of the VM.

The skeleton of the synthetic application we use for our experiments is shown in Listing 2. The parameter *SIZE* controls the WSS: SIZE × 4KB. 4KB is the size of `tab[j]` which is also the size of a memory page (in our architecture). The

parameter *PERIOD* represents the duration of the application, and `operation(tab[j])` is the operation performed on the array entry. We consider three types of workload: every read is followed by a write (*RWRW*), a set of reads followed by a set of writes (*RRWW*), and a set of writes followed by a set of reads (*WWRR*). We choose an application with well known read and write phases, instead of random operations as in Section 2.2, because we want to clearly show how PRL performs facing each operation type.

4.2 Accuracy of the simulator

We validate the accuracy of our EPT and PML simulation by running each workload type of the synthetic application atop our simulator, then on a real machine supporting EPT and PML. We compare the content of the PML buffer in both cases. Since in the real environment the application runs inside a VM (which includes a guest OS), several GPAs that do not belong to the synthetic application's address space can be logged into the PML buffer. These GPAs are part of other components inside the VM which perform write operations. Once eliminating these GPAs, we observe that the content of the PML buffer in both environments is the same, validating the accuracy of our simulator. From now, we can study the efficiency of PRL in the task of WSS estimation.

4.3 PRL efficiency

We evaluate the accuracy of the WSS estimation of an application running in a VM. We consider the synthetic application described in Section 4.1 and the HPL Linpack described in Section 2.2. The WSS of the synthetic application is fixed to 400 MB by setting *SIZE* accordingly (i.e., `102400 pages`). We do not use BigDataBench applications because their execution never complete under Gem5 in our experimental environment. We compare our results against PML and VMWare sampling-based WSS estimation techniques. We confront the VMWare WSS estimation technique mainly due to its popularity, albeit its skewed results due to hard sampling period configuration. Additionally, other WSS estimation techniques benefit from extensive evaluation in different research works, such as[50].

The results of our experiments are shown in Figure 8. We observe that the WSS estimation based on PRL has a margin of error of less than 1MB for all workloads, *i.e.*, a precision greater than 99.75%[8]. In contrast, the VMware solution provides inaccurate results, which is in line with previous research observations [52]. The accuracy of the WSS estimation based on PML mainly depends on the number of write operations. In the RWRW and WWRR workloads, all the array entries are updated making PML suitable for correctly estimating the WSS. In contrast, the RRWW workload highlights the incapacity of PML to identify hot pages when only read operations are performed.

Regarding the HPL Linpack application, we observe that both PRL and PML provide similar WSS estimations, while the solution based on VMware still diverge a lot. Because the real WSS is not known in advance, we estimate its value as follows. We dynamically set the memory size of the VM to the WSS estimated using the PRL-based system. Since we do not observe any VM crash and no performance degradation, the PRL-based WSS value is either overestimated or accurate. We decrease the memory size of the VM by 100MB and observe a crash of the VM, which confirms that the previously estimated WSS was accurate.

As illustrated in Table 2, we observe that the number of GPAs missed when handling full log events in PRL is negligible. Indeed, it is very likely that a GPA that is part of the working set is seen very often. Missing one GPA, thus, does not compromise the accuracy of the WSS estimation.

To confirm our findings, we extend our evaluations by including the applications of PARSEC [13], a benchmark suite composed of multithreaded programs. We used PARSEC because it is widely used by the research community, and the WSS of some applications in the suite are known from the technical report of PARSEC in 2008 [27]. Our results are presented in Figure 7. Apart from `freqmine`, all the values estimated by PRL match the initial technical report[9]. We observe that the PML-based solution is accurate for six applications (streamcluster, vips, fluidaminate, blacksholes, bodytrack, dedup) while the VMware-based solution fails to provide correct values. By making the implementation of PRL under Gem5 publicly available, researchers can use it for estimating the WSS of other benchmarks.

4.4 PRL overhead

We evaluate both the impact induced by PRL on the VM whose WSS is estimated and the number of resources consumed by the WSS estimation system inside the dom0.

The overhead on the VM whose WSS is estimated is the the total number of CPU cycles used by the PRL internal circuitry. Because a PRL-capable machine does not exist yet and Gem5 does not measure internal CPU circuitry costs[10], we assume that this number of CPU cycles should be roughly equivalent to that of PML, since the two modes are very similar. Therefore, to assess PRL overhead, we use a PML-capable machine on which the VM runs an application that performs only read operations. This scenario avoids having `VMExits` that can not occur with a PRL-based machine for which `VMExits` are executed on a different processor. Remember that both PML and PRL take place during page table walk, on

[8]Notice that due to the workload nature (array scan), the working set is mostly constant.

[9]Regarding freqmine, the latter's dataset evolved since the [27] report, which explains the discrepancy between the [27] results and ours.

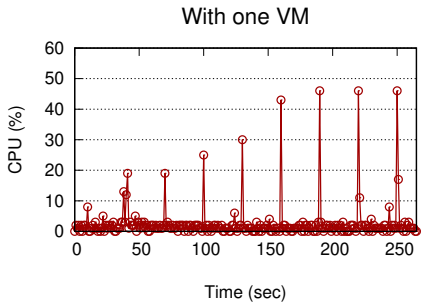
[10]Gem5 releases till the writing of this paper do not provide this feature.

Buffer size (MB)	RRWW			HPL		
	# full log events	# missed GPAs	mean of missed GPAs per full log event	# full log events	# missed GPAs	mean of missed GPAs per full log event
512	17094	0	0	20701	741	0.04
1024	8543	213	0.02	10510	116	0.01

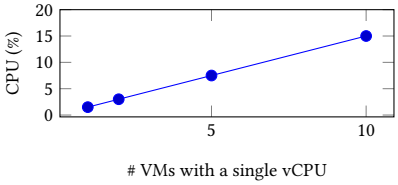
Table 2. Number of missed GPAs during the treatment of full log events. Note that results for WWRR and RWRW are almost the same as for RRWW, thus we decide not to present them.

each TLB miss. Thus, we repeated this experiment by varying the number of tenant VMs to increase the pressure on the TLB. We hardly measure any difference in performance (around 0.001%), which means that PRL will likely not experience any performance degradation on the VM whose WSS is computed.

We use the emulated environment to evaluate the CPU consumption of the WSS estimation system inside the dom0, which is pinned to a specific core. We use a single VM and a write-only workload (WWRR in which RR is null) as the worse case. Figure 9a presents the percentage of CPU consumed by the WSS estimation process in the dom0 for a VM with a single vCPU. For readability, only a representative portion of results is presented. The CPU consumed during the treatment of the full log event increases (with the size of the cumulative buffer) until the WSS is discovered. However, we can observe that the CPU is most of the time idle, waiting for a PRL log full event. On average, the total CPU time consumed by the WSS estimation process for a VM with a single vCPU is less than 1.5%. We repeated the same experiment by varying the number of VMs. As illustrated in Figure 9b, the average percentage of CPU consumed by the WSS estimation system increases linearly.



(a) CPU consumption over time for a single VM.



(b) Average CPU consumption for several VMs.

Figure 9. CPU consumption for the WSS estimation process on VMs with a single vCPU

5 Related work

Hardware assisted virtualization (HAV). Contrary to A. Baumann's conclusion in his HotOS'17 paper [21], we believe that virtualization is the System's sub-domain which mainly influences hardware architecture research. In fact, HAV contributions have evolved at the rhythm of the limitations of software solutions. Notably, Extended/Nested Page Table [22] has been introduced for addressing the tremendous number of context switches caused by shadow paging [58]. In summary, several hardware features (e.g., Intel VT [10], VMFUNC [14], Intel CAT [11], APICv [48]) have been integrated inside CPU, memory subsystems, I/O devices and many other motherboard components by hardware manufacturers these recent years for achieving basic virtualization functionalities. An extreme application of the HAV approach has been proposed by E. Keller with NoHype [38] which is a hardware only hypervisor. In 2017, Amazon anounced its new hypervisor called Nitro [41], which can be seen as a concretization of the NoHype vision. In the academia, a lot of efforts have been made in the topic of memory virtualization [17, 18, 20, 32, 33, 42, 60, 61] to minimize the overhead of the 2D page walk imposed by EPT.

Page tracking. Page tracking is, in essence, the core of working set estimation techniques. The most popular approach for page tracking consists in denying access to memory pages which need to be monitored so that next accesses trap inside the system software (hypervisor or OS). This approach is used by the majority of checkpointing, live migration and WSS estimation solutions. Very few research works have investigated hardware features for page access tracking. Pin Zhou et al. [65] proposed a Miss Ratio Curve (MRC) monitoring hardware feature which can be used as an alternative to page access tracking in the task of WSS estimation [64]. [65] proposed a solution which consists in snooping the address bus and requires a collaboration with the OS page fault handler. As PML/PRL, [65] showed that tracking page accesses at the hardware level is possible. However, [65] is dedicated to native systems and it needs to collaborate with the OS, unlike PML/PRL.

Live migration. [46, 56] presented surveys that the reader could refer to. We would like to highlight among them [16, 49] who studied live migration of the VM storage along with its memory. Very few research work has investigated VM storage live migration because it increases the VM downtime. Migrating the VM storage is necessary when dealing with data intensive applications because they generally use local storage instead of the classical network storage. The utilization of PRL/PML is also beneficial for this use case because disk accesses go through the buffer cache, which resides in RAM. [63] addressed another important aspect of live migration which is the prediction of the right migration instant. In fact, live migration could fail due to lack of resources or sudden VM behavior changes. The authors use the VM WSS to track such behavior changes. Thus PRL is likely to improve [63]'s contribution.

WSS estimation. *Committed_AS*, a Linux kernel statistic, is generally used (e.g., by Xen) to estimate the VM WSS. This statistic corresponds to the total number of anonymous memory pages allocated by all processes, but not necessary backed by physical pages. Therefore, *Committed_AS* overestimates the WSS. Another limitation of this approach is the fact that it requires a collaboration between the hypervisor and the guest OS. Zballoond [26] relies on the following observation: when a VM's memory size is larger than or equal to its WSS, the number of swap-in and refault (occurs when a previously evicted page is later accessed) events is close to zero. The basic idea behind *Zballoond* consists in gradually decreasing the VM's memory size until these counters start to increase. The VM's WSS is the lowest memory size which leads the VM to zero swap-in and refault events. Like *Committed_AS*, *Zballoond* requests the collaboration with the guest kernel. Furthermore, *Zballoond* is very active in the sense that it performs memory pressure on the VM, which could degrade the VM performance. Geiger [35] monitors the evictions and subsequent reloads from the guest OS buffer cache from/to the swap device. It relies on a ghost buffer [53] which represents 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, the VM's WSS can be computed using the following formula: $WSS = m_{cur} + m_{ghost}$ if $m_{ghost} > 0$. Unlike the two previous solutions, *Geiger* is transparent from the VM's point of view. However, Geiger has an important drawback which derives from its non-intrusiveness. It is able to estimate the WSS only when the size of the ghost buffer is greater than zero (the VM is in a swapping state). Geiger is inefficient if the VM's WSS is smaller than the current memory allocation. Hypervisor Exclusive Cache [45] is fairly similar to Geiger. Badis [50] combined VMware and *Geiger* in order to take advantage of their non intrusivity on the VM's codebase. Badis suffers from VMware and Geiger's drawbacks presented above. [64] computes the WSS of an application based on its miss-ratio curve (MRC). The latter shows the fraction of the cache misses that would turn into cache hits if the VM's allocated memory increases. [64] presents a set of methods to determine the values of the input parameters of our WSS estimation system (τ, ω, and μ). [40] presents an application-assisted WSS estimation solution in virtualized systems. In contrast to our solution which considers the VM as a black blox, [40] relied on the application inside the VM to estimate the WSS, which is very intrusive.

6 Conclusion

This paper presents an analysis of Page Modification Logging (PML), a memory page tracking technology introduced by Intel and VMware as a key virtualization functionality. We show that the current design of PML makes it effective for VM live migration. We propose Page Reference Logging (PRL), an extension to PML which also makes it effective for WSS estimation. We implemented PRL in Gem5, a popular hardware simulator, and described a WSS estimation system that takes advantage of PRL. We evaluated our solution using real and synthetic applications and compared it with existing solutions. Our results demonstrate that our solution is both accurate and does not impact user VMs.

7 Acknowledgment

We thank our shepherd, Steve Blackburn, and the anonymous reviewers for their insightful comments. This work was funded by the "Scalevisor" project of the Agence Nationale de la Recherche, number ANR-18-CE25-0016, the HYDDA Project of BPI Grant and the "IDEX IRS" (COMUE UGA grant).

References

[1] 2018. http://www.linux-kvm.org. (Dec. 2018).
[2] 2018. Hyper-V Architecture. http://msdn.microsoft.com/en-us/library/cc768520.aspx.. (Dec. 2018).
[3] 2018. Page-Modification Logging for Virtual-Machine Monitor. https://www.intel.fr/content/www/fr/fr/processors/page-modification-logging-vmm-white-paper.html. (Feb. 2018).
[4] 2018. VMKernel Architecture. https://communities-gbot.vmware.com/thread/93870. (Dec. 2018).
[5] 2018. VMware VMkernel. https://microage.com/wp-content/uploads/2016/02/ESXi_architecture.pdf.. (Dec. 2018).
[6] 2019. (July 2019). http://gem5.org/Publications
[7] 2019. A Scalable Big Data and AI Benchmark Suite, ICT, Chinese Academy of Sciences. http://prof.ict.ac.cn/BigDataBench/. (March 2019).
[8] 2019. HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. http://www.netlib.org/benchmark/hpl/. (March 2019).
[9] 2019. improving real-time performance by utilizing cache allocation technology. White paper. https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/cache-allocation-technology-white-paper.pdf. (July 2019).

[10] 2019. Intel® Virtualization Technology (Intel® VT). https://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html. (April 2019).

[11] 2019. Introduction to Cache Allocation Technology in the Intel® Xeon® Processor E5 v4 Family. https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology. (April 2019).

[12] 2019. Memory Overcommit. https://www.techopedia.com/definition/14761/memory-overcommi. (July 2019).

[13] 2019. PARSEC. http://parsec.cs.princeton.edu/. (April 2019).

[14] 2019. VMFUNC - Invoke VM function. https://www.felixcloutier.com/x86/vmfunc. (April 2019).

[15] March 2018. *Intel 64 and IA-32 Architectures Software Developer's Manual* (volume 3, chapter 24 virtual machine control structures ed.). Intel Corporation.

[16] P. Santhi Thilagam Abhinit Modi, Raghavendra Achar. 2017. Live Migration of Virtual Machines with Their Local Persistent Storage in a Data Intensive Cloud. *Int. J. High Perform. Comput. Netw.* 10, 1-2 (Jan. 2017), 134–147. http://dl.acm.org/citation.cfm?id=3070823.3070837

[17] Jeongseob Ahn, Seongwook Jin, and Jaehyuk Huh. 2012. Revisiting Hardware-assisted Page Walks for Virtualized Systems. In *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA '12)*. IEEE Computer Society, Washington, DC, USA, 476–487. http://dl.acm.org/citation.cfm?id=2337159.2337214

[18] Hanna Alam, Tianhao Zhang, Mattan Erez, and Yoav Etsion. 2017. Do-It-Yourself Virtual Memory Translation. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. ACM, New York, NY, USA, 457–468. DOI: http://dx.doi.org/10.1145/3079856.3080209

[19] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the art of virtualization. In *IN SOSP*. 164–177.

[20] Thomas W. Barr, Alan L. Cox, and Scott Rixner. 2010. Translation Caching: Skip, Don'T Walk (the Page Table). In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA '10)*. ACM, New York, NY, USA, 48–59. DOI: http://dx.doi.org/10.1145/1815961.1815970

[21] Andrew Baumann. 2017. Hardware is the New Software. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS '17)*. ACM, New York, NY, USA, 132–137. DOI: http://dx.doi.org/10.1145/3102980.3103002

[22] Ravi Bhargava, Benjamin Serebrin, Francesco Spadini, and Srilatha Manne. 2008. Accelerating Two-dimensional Page Walks for Virtualized Systems. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XIII)*. ACM, New York, NY, USA, 26–35. DOI: http://dx.doi.org/10.1145/1346281.1346286

[23] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7. DOI: http://dx.doi.org/10.1145/2024716.2024718

[24] Marc Brooker and Holly Mesrobian. 2018. A Serverless Journey: Under the Hood of AWS Lambda. https://www.youtube.com/watch?v=QdzV04T_kec. (Nov. 2018).

[25] 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

[26] 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

[27] Jaswinder Pal Singh Christian Bienia, Sanjeev Kumar and Kai Li. 2008. The PARSEC Benchmark Suite: Characterization and Architectural Implications. https://parsec.cs.princeton.edu/doc/parsec-report.pdf. (Jan. 2008).

[28] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. 2005. Live Migration of Virtual Machines. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2 (NSDI'05)*. USENIX Association, Berkeley, CA, USA, 273–286. http://dl.acm.org/citation.cfm?id=1251203.1251223

[29] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. ACM, New York, NY, USA, 153–167. DOI: http://dx.doi.org/10.1145/3132747.3132772

[30] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-efficient and QoS-aware Cluster Management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14)*. ACM, New York, NY, USA, 127–144. DOI: http://dx.doi.org/10.1145/2541940.2541941

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

[32] Jayneel Gandhi, Arkaprava Basu, Mark D. Hill, and Michael M. Swift. 2014. Efficient Memory Virtualization: Reducing Dimensionality of Nested Page Walks. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-47)*. IEEE Computer Society, Washington, DC, USA, 178–189. DOI: http://dx.doi.org/10.1109/MICRO.2014.37

[33] Jayneel Gandhi, Mark D. Hill, and Michael M. Swift. 2016. Agile Paging: Exceeding the Best of Nested and Shadow Paging. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)*. IEEE Press, Piscataway, NJ, USA, 707–718. DOI: http://dx.doi.org/10.1109/ISCA.2016.67

[34] 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. DOI: http://dx.doi.org/10.1145/1168857.1168861

[35] Stephen T. Jones, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2006. Geiger: Monitoring the Buffer Cache in a Virtual Machine Environment. *SIGARCH Comput. Archit. News* 34, 5 (Oct. 2006), 14–24. DOI: http://dx.doi.org/10.1145/1168919.1168861

[36] 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. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, Berkeley, CA, USA, 117–134. http://dl.acm.org/citation.cfm?id=3026877.3026887

[37] Harshad Kasture and Daniel Sánchez. 2016. Tailbench: a benchmark suite and evaluation methodology for latency-critical applications. In *2016 IEEE International Symposium on Workload Characterization, IISWC 2016, Providence, RI, USA, September 25-27, 2016*. 3–12. DOI: http://dx.doi.org/10.1109/IISWC.2016.7581261

[38] Eric Keller, Jakub Szefer, Jennifer Rexford, and Ruby B. Lee. 2010. NoHype: Virtualized Cloud Infrastructure Without the Virtualization. *SIGARCH Comput. Archit. News* 38, 3 (June 2010), 350–361. DOI: http://dx.doi.org/10.1145/1816038.1816010

[39] Jinchun Kim, Viacheslav Fedorov, Paul V. Gratz, and A. L. Narasimha Reddy. 2015. Dynamic Memory Pressure Aware Ballooning. In *Proceedings of the 2015 International Symposium on Memory Systems (MEMSYS '15)*. ACM, New York, NY, USA, 103–112. DOI: http://dx.doi.org/10.1145/2818950.2818967

[40] Min Lee, A. S. Krishnakumar, P. Krishnan, Navjot Singh, and Shalini Yajnik. 2011. Hypervisor-assisted Application Checkpointing in Virtualized Environments. In *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems&Networks (DSN '11)*. IEEE Computer Society, Washington, DC, USA, 371–382. DOI: http://dx.doi.org/10.1109/DSN.2011.5958250

[41] Anthony Liguori. 2018. Powering Next-Gen EC2 Instances – Deep Dive into the Nitro System. https://www.youtube.com/watch?v=e8DVmwj3OEs. (Nov. 2018).

[42] Jin Tack Lim, Christoffer Dall, Shih-Wei Li, Jason Nieh, and Marc Zyngier. 2017. NEVE: Nested Virtualization Extensions for ARM. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP'17), Shanghai, China, October 28-31, 2017*. 201–217.

[43] Kevin T. Lim, Jichuan Chang, Trevor N. Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, and Thomas F. Wenisch. 2009. Disaggregated Memory for Expansion and Sharing in Blade Servers. In *Proceedings of the 2009 ACM/IEEE Annual International Symposium on Computer Architecture (ISCA'09)*. ACM, 267–278.

[44] 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

[45] 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

[46] Violeta Medina and Juan Manuel García. 2014. A Survey of Migration Mechanisms of Virtual Machines. *ACM Comput. Surv.* 46, 3, Article 30 (Jan. 2014), 33 pages. DOI: http://dx.doi.org/10.1145/2492705

[47] Jeffrey C. Mogul, Andrew Baumann, Timothy Roscoe, and Livio Soares. 2011. Mind the gap: Reconnecting architecture and OS research. In *Proceedings of the 13th Workshop on Hot Topics in Operating Systems (HotOS)*.

[48] Khang T (Intel) Nguyen. 2018. APIC Virtualization Performance Testing and Iozone. https://software.intel.com/en-us/blogs/2013/12/17/apic-virtualization-performance-testing-and-iozone. (Dec. 2018).

[49] Bogdan Nicolae and Franck Cappello. 2012. A Hybrid Local Storage Transfer Scheme for Live Migration of I/O Intensive Workloads. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing (HPDC '12)*. ACM, New York, NY, USA, 85–96. DOI: http://dx.doi.org/10.1145/2287076.2287088

[50] 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. DOI: http://dx.doi.org/10.1145/3179422

[51] 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. In *Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '17)*. ACM, New York, NY, USA, 1–14. DOI: http://dx.doi.org/10.1145/3050748.3050758

[52] Vlad Nitu, Boris Teabe, Alain Tchana, Canturk Isci, and Daniel Hagimont. 2018. Welcome to Zombieland: Practical and Energy-efficient Memory Disaggregation in a Datacenter. In *Proceedings of the Thirteenth EuroSys Conference (EuroSys '18)*. ACM, New York, NY, USA, Article 16, 12 pages. DOI: http://dx.doi.org/10.1145/3190508.3190537

[53] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. 1995. Informed Prefetching and Caching. *SIGOPS Oper. Syst. Rev.* 29, 5 (Dec. 1995), 79–95. DOI: http://dx.doi.org/10.1145/224057.224064

[54] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'18)*. USENIX Association, Carlsbad, CA, USA, 69–87.

[55] Bharathi Subramanian. 2018. NAPI - The New API for Linux Network Drivers. https://bharathisubramanian.wordpress.com/2010/04/13/napi-the-new-api-for-linux-network-drivers/. (Dec. 2018).

[56] Petter Svärd, Benoit Hudzia, Steve Walsh, Johan Tordsson, and Erik Elmroth. 2015. Principles and Performance Characteristics of Algorithms for Live VM Migration. *SIGOPS Oper. Syst. Rev.* 49, 1 (Jan. 2015), 142–155. DOI: http://dx.doi.org/10.1145/2723872.2723894

[57] 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. DOI: http://dx.doi.org/10.1109/MC.2005.163

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

[59] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking Behind the Curtains of Serverless Platforms. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 133–146.

[60] Xiaolin Wang, Jiarui Zang, Zhenlin Wang, Yingwei Luo, and Xiaoming Li. 2011. Selective Hardware/Software Memory Virtualization. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '11)*. ACM, New York, NY, USA, 217–226. DOI: http://dx.doi.org/10.1145/1952682.1952710

[61] Idan Yaniv and Dan Tsafrir. 2016. Hash, Don'T Cache (the Page Table). In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science (SIGMETRICS '16)*. ACM, New York, NY, USA, 337–350. DOI: http://dx.doi.org/10.1145/2896377.2901456

[62] Irene Zhang, Alex Garthwaite, Yury Baskakov, and Kenneth C. Barr. 2011. Fast Restore of Checkpointed Memory Using Working Set Estimation. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '11)*. ACM, New York, NY, USA, 87–98. DOI: http://dx.doi.org/10.1145/1952682.1952695

[63] Jinshi Zhang, Eddie Dong, Jian Li, and Haibing Guan. 2017. MigVisor: Accurate Prediction of VM Live Migration Behavior Using a Working-Set Pattern Model. *SIGPLAN Not.* 52, 7 (April 2017), 30–43. DOI: http://dx.doi.org/10.1145/3140607.3050753

[64] Weiming Zhao, Xinxin Jin, Zhenlin Wang, Xiaolin Wang, Yingwei Luo, and Xiaoming Li. 2011. Low Cost Working Set Size Tracking. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference (USENIXATC'11)*. USENIX Association, Berkeley, CA, USA, 17–17. http://dl.acm.org/citation.cfm?id=2002181.2002198

[65] Pin Zhou, Vivek Pandey, Jagadeesan Sundaresan, Anand Raghuraman, Yuanyuan Zhou, and Sanjeev Kumar. 2004. Dynamic Tracking of Page Miss Ratio Curve for Memory Management. *SIGOPS Oper. Syst. Rev.* 38, 5 (Oct. 2004), 177–188. DOI: http://dx.doi.org/10.1145/1037949.1024415