POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Characterization of Virtualization-Induced Noise in High-Performance Computing and Real-Time Virtual Machines

Laurea Magistrale in Computer Science and Engineering

Author: Francesco Aristei

Advisor: Prof. Vittorio Zaccaria

Co-advisors: Paolo Bonzini, Daniel Bristot de Oliveira

Academic year: 2022-2023

1. Introduction

The scientific community widely accepts virtualization for latency-sensitive applications, especially in HPC and real-time contexts [1, 12]. HPC has moved from expensive dedicated infrastructures to cost-effective cloud options. Virtual machines are increasingly popular in real-time applications to integrate various software components with different criticality levels. Networking, including SDN and NFV [5, 16], is another vital use case. A key concern in these applications is the VM transition delays from guest to hypervisor, which introduces overhead. However, extensive research has found these delays acceptable [8, 10]. Indeed, latency-sensitive workloads on virtualized and bare metal platforms perform comparably. However, studies often lack detailed performance explanations, requiring users to analyze the execution environment for insight. In the HPC world, an essential latency metric is Operating System Noise, defined as the CPU time the system takes from a workload running on a given processor. Linux is the preferred choice in both HPC and Real-Time applications due to its flexibility. Osnoise [4] is a tool within the Linux kernel's tracing facilities that measures OS noise. The tool offers a quantitative and qualitative view of the metric, linking delays with the triggering events. In both HPC and real-time systems in virtualized environments, various applications benefit from this analysis. For example, Network Function Virtualization replaces network appliance hardware with virtual machines, running networking processes like routing and load balancing via a hypervisor. These workloads require precise timing, and osnoise could help identify virtualization layer interferences for improved real-time performance. However, currently, when run on a virtual machine, osnoise cannot offer insights into virtualization layer noise, including time taken from the guest by the hypervisor or higher-priority tasks. This knowledge would improve system debugging by pinpointing the source of the noise. The tool misinterprets this disturbance as generic hardware noise, providing a misleading explanation for system delays. This work proposes extending osnoise to address virtualization noise. New statistics within the guest measure the noise of the hypervisor and the host, providing quantitative information. On the host, when osnoise runs while virtual machines are active, a tracepoint records vCPUs leaving the physical processor to execute

other tasks. This, along with osnoise's tracing data, identifies the events causing the noise measured on the guest, providing both noise measurement and its root causes. A series of experiments confirmed osnoise's output consistency with expectations. Hardware noise, including unmaskable hardware interrupts (e.g., SMIs), is the same in both the host and its guest VMs. Comparing osnoise runs on a VM in its vanilla and new versions showed a significant reduction in hardware-induced noise, indicating that much of the noise in the vanilla version stemmed from the virtualization layer. In particular, the noise in the last case matched that of the osnoise directly on the host, as expected. It demonstrated osnoise's practical use for system debugging and tuning. For low-latency applications in HPC and real-time scenarios, adjustments like processor speed and CPU partitioning into isolated and housekeeping sets are common. Osnoise was run on a virtual machine with and without these adjustments to assess their impact on running low-latency tasks on virtualized platforms. Although there was improvement compared to the non-tuned case, the resulting noise after tuning still exceeded the required values for latency-sensitive workloads. A deeper analysis revealed that, based on host extensions' tracing information from osnoise, the cause of excessive noise is higher-priority tasks preempting the vCPU running the guest. This results from the hardware limitations of the experiments with only four available cores, overloading the system, and prompting the OS to run housekeeping tasks on isolated CPUs. This showcases osnoise's debugging utility by pinpointing the noise source. Experimental results validate osnoise's output, aligning with expected behavior, and demonstrating its effectiveness in accelerating system debugging and tuning, providing both quantitative and qualitative noise insights.

2. Background

First, the osnoise tool is presented, then the virtualization technology of Linux, KVM, is briefly described.

2.1. Osnoise

The Operating System Noise is defined as all the time spent by a CPU executing instructions not belonging to a given application task assigned to that CPU while the task is ready to run. Osnoise comprises workload and tracing components, with each CPU hosting a periodic kernel thread. These threads continuously read time, collecting a new noise sample when the gap between consecutive readings exceeds a set tolerance threshold. The osnoise tracer utilizes the Linux tracing infrastructure through tracepoints, which are points in the kernel code where probes can be attached to run functions, typically used for collecting trace information. It does this by adding probes to existing tracepoints for data collection and introducing a new set of tracepoints. This allows for the generation of an osnoise tracepoint each time noise is detected, reporting the observed task's noise description.

2.2. Kernel Virtual Machine

KVM is the Linux module that allows to use the hardware assisted virtualization technology to create and manage virtual machines. For I/O emulations, KVM uses a userland software, QEMU [3], which performs as said, hardware emulation. Each virtual machine is treated as a simple process by QEMU. Virtual CPUs (vCPUs) on a virtual machine are Linux threads, scheduled like other tasks. This results in interruptions from other virtualized tasks and noise from the host. Additionally, when the guest performs sensitive instructions, the hypervisor takes control of the processor in a vmexit, introducing overhead. This, combined with host interruptions from higher-priority contexts, defines virtualization-induced noise.

3. Related Work

In HPC, a common method to measure OS noise is running micro-benchmarks with known durations and comparing the expected and actual processing times. The Fixed Work Quantum (FWQ) benchmark [13] is an example. Hensbergen [9] pioneered the analysis of virtualization's impact on HPC applications, using the FWQ benchmark to assess noise levels in virtual machine execution. In HPC, the NAS parallel benchmarks [6] and the HPC Challenge benchmark [15] are commonly used to assess the performance of the parallel computing system, including memory bandwidth, network communication, and computation capabilities. To eval-

uate virtualization in HPC, Kudryavtsev et al. [2] used these benchmarks to compare the Palacios hypervisor and the Kernel Virtual Machine (KVM). These benchmarks primarily focus on performance metrics like operations per second (e.g., FLOPS), highlighting performance issues without detailed root cause explanations. In the Real-Time community, tools like oslat [17] and sysjitter measure OS noise by running a thread on each CPU and tracking intervals where the thread is not running due to OS activities. These tools have demonstrated equivalent performance for Radio Access Network (RAN) workloads in VMware vSphere and bare metal [11]. However, in the real-time context, the results of the experiment quantify the latency of the system, and it is up to practitioners to identify its source.

4. Approach

Osnoise has been extended on both the guest and host side. From the guest extension, it measures noise from the hypervisor and the host, indicating how long the virtual machine was taken away from the processor to execute the hypervisor and higher-priority tasks. However, within the guest, you cannot determine the specific events that caused the interruptions. To address this, osnoise was extended for host-side execution to identify these events, providing a comprehensive understanding of virtualization-induced noise, including numerical values and their root causes. The extensions are presented in conjunction with the evaluation methodology.

4.1. Guest Side

In the vanilla version of the tool, noise detection checks for changes in the number of interferences experienced by the osnoise thread between iterations. If this value remains unchanged, it signifies hardware-induced noise without other task preemptions. However, in virtual machine environments, the virtual CPU executing the osnoise thread could be preempted by the hypervisor or the host for sensitive tasks. The guest-side extension of the tool distinguishes this virtualization-induced noise from hardware-related noise. In virtualization, when the hypervisor or host takes control of the physical processor, it triggers a vmexit event. KVM distinguishes between a lightweight vmexit handled by KVM itself and a heavyweight vmexit, typically caused by device accesses and involving QEMU intervention, introducing more latency. The osnoise output now includes two counters, one for each vmexit type. The time a virtual CPU is unscheduled from the physical processor due to host preemption is known as steal time. To calculate the time the hypervisor manages sensitive instructions on behalf of the virtual machine, we subtract the steal time from the total noise experienced by the osnoise thread. Any time not categorized as steal time or hypervisor noise represents the time spent on hardware-related tasks.

4.2. Host Side

On the host side, the tracepoints provided by osnoise are extended by adding a new one to detect vmexit events affecting virtual machines. This new tracepoint captures crucial information about vmexit events, including the time of occurrence, the exited CPU, the triggering reason, duration, and hypervisor overhead. When combined with the existing Linux kernel tracing capabilities, it allows the reconstruction of events during the exit, providing insights into the causes of noise detected when running osnoise on the guest.

4.3. Evaluation Methodology

First, each extension of the tool has been verified in order to guarantee that the output provided by osnoise is coherent with the expected results. After that, experiments have been conducted in order to show the validity of the extension introduced, and how it brings significant results that can be practically used to speed up the debugging of a system. Measurements were carried out on the ODROID-H2 board, equipped with an Intel Celeron J4105 processor with 4 cores. The system and virtual instances ran Fedora Linux 37 with the Linux kernel version 6.2, patched with the PREEMPT-RT patchset [7]. Virtual machines were created using QEMU, allocated 2 GB of RAM, and configured with 3 cores.

4.3.1 Verification

On the guest side, the initial step involved assessing the accuracy of the modified osnoise output. First, it has been verified that the hardware noise measured within the virtual machine, using the modified tool, matched the hardware

noise obtained when running osnoise directly on the host, as hardware noise is not related to virtualization. To further assess the coherency of the output, a second run examined the types of vmexits that affect the guest. Indeed, in a CPU-bound workload like osnoise, most of the exits are expected to be of the lightweight variety. To verify the host extension of the tool, osnoise was executed directly on the host. At the same time, small virtual machines were launched. These VMs executed two different instructions: CPUID and I/O ports reading. The former involved a guest performing multiple CPUID instructions, which could be handled by KVM, resulting in a lightweight vmexit. The latter, in contrast, had a guest reading a 32-bit value from an I/O port, requiring QEMU emulation and causing a heavyweight vmexit. While each of the VMs was running, the osnoise tracer was set to trace both the osnoise and kvm events. The vmexit duration measured by the new osnoise tracepoint needs to coincide with the time passed between the vmentry and vmexit events traced for the measured vmexit.

4.3.2 Validation

The tool has been used practically to assess the viability of executing low-latency tasks on a virtualized platform. To do so, following established best practices in both the HPC and the real-time domains, the CPUs were divided into isolated and housekeeping sets. Housekeeping CPUs were designated for general system tasks, while isolated CPUs were exclusively reserved for low-latency operations. This configuration was applied to both the host and the guest operating systems. At this point, osnoise was run with various tuning configurations. Initially, CPU isolation was applied solely to the host system, followed by isolation only in the guest, and lastly, the most rigorous isolation scenario was tested by tuning both the host and guest systems. In each experiment, osnoise statistics regarding virtualization noise were recorded. To understand the causes of the noise measured on the guest side, a virtual machine was launched with CPU isolation applied to both the host and the virtual instance. Osnoise was run on isolated vCPUs within the guest while osnoise was running simultaneously on the host. The new tracepoint was used to collect the events that caused virtualization noise in the guest, showing how osnoise is able to provide both quantitative insights into hypervisor/host-induced noise and the underlying causes. Every CPU-intensive task with tight timing requirements working in user context can benefit from the extension provided to the tool. For example, NFV workloads for packet processing work by polling the network for packets, resembling the osnoise workload. Therefore, osnoise could be used to debug the systems where these kinds of applications need to run.

5. Evaluation

The experimental results are presented. First experiments have been performed to verify the correctness of the output provided by osnoise; then the tool has been practically used in order to discover the causes of non-negligible noise, validating its practical utility.

5.1. Verification

5.1.1 Guest Side

The baseline version of osnoise, when run on a virtual machine, showed a significant contribution of hardware noise to the overall one. However, when the extended version of the tool was used, the hardware noise decreased.The reason being that in the vanilla version of the tool, the hardware noise included the noise coming from the virtualization layer. Therefore, with the new changes introduced, the tool is able to distinguish between the noise coming from the virtualization layer and the hardware noise. Notably, the hardware noise measured in the guest with the extended version matched that obtained when running osnoise directly on the host. This shows the coherency of the output provided, the hardware noise being not related to virtualization. As a second step to verify the correctness of the output, osnoise was utilized to examine the types of vmexits encountered by the guest. Most detected vmexits were of the lightweight variety, which was anticipated, as heavyweight exits associated with device accesses are less common in a CPU-bound workload like osnoise.

5.1.2 Host Side

To verify the correctness of the host extension of the tool, the following steps were performed:

- Osnoise was executed directly on the host alongside a virtual machine running the CPUID instruction.
- Osnoise was executed on the host alongside a virtual machine performing I/O port reads.
- The osnoise tracer was configured to trace both osnoise and kvm events.

The output generated by the additional tracepoint aligned with the type of instruction executed in both cases. The CPUID instruction led to lightweight vmexits handled directly by the hypervisor, resulting in a duration of a few microseconds. On the contrary, the I/O instructions required hardware emulation by QEMU, causing an increase in the duration of vmexit.

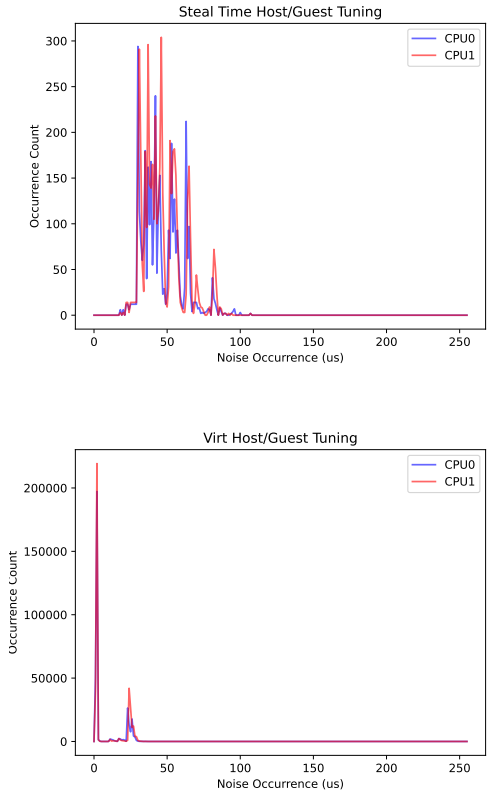

Figure 1: Top: the steal time with the maximum tuning applied. Bottom: Virtualization noise after host/guest tuning.

5.2. Validation

After verifying the coherency of the output, osnoise was used to measure the noise of the virtualization layer and assess the feasibility of running low-latency tasks within a virtual machine. Experiments were conducted using various tuning configurations as previously described.

- Running osnoise with CPU isolation applied on the host led to a significant drop in steal time, indicating minimal descheduling of the guest. Noise caused by KVM and QEMU also decreased, because fewer preemptions, requires less job by the hypervisor to save the virtual machine state.
- In the case where CPU isolation was implemented only on the guest side, improvements were modest compared to host tuning. Steal time remained unchanged, while the KVM/QEMU noise decreased slightly.
- Tuning both host and guest with CPU isolation achieved the lowest noise levels. Nevertheless, even in this extreme isolation scenario, osnoise's statistics indicated that both steal time and hypervisor noise exceeded acceptable values for low-latency tasks as shown in Figure 1.

The analysis of virtualization noise, previously measured by running osnoise on the guest, was carried out by replicating the experimental setup used to test the guest extension of the tool. The following steps were followed:

- A virtual machine was launched with CPU isolation applied to both the host and the guest, with osnoise running on top of it.
- Simultaneously, osnoise was launched on the host to trace the events on the CPUs where the virtual machine was executed.

By combining the existing osnoise tracepoint with the newly added one, the events responsible for causing spikes in the noise observed on the guest were identified. As seen previously on the guest, the results revealed that, even after system tuning, some housekeeping tasks were still executed on the isolated CPUs, introducing non-negligible noise to the virtual machine. This was due to the limitations of the experimental hardware, which featured only four available cores and could not completely isolate the CPUs. The experiment demonstrated that with these new features, osnoise not only quantifies virtualization-induced noise, but also provides insights into the events that trigger it, thereby expediting system debugging.

6. Conclusions and Future Works

In today's High Performance Computing and Real-Time applications, the use of virtualization has become an accepted practice. However, existing tools do not identify the underlying causes of performance issues, leaving users to manually analyze system complexities. To bridge this gap, osnoise, a tracer that measures OS noise, has been extended. It now encompasses noise arising from virtualization components like KVM and QEMU. Yet, a more refined capability to differentiate between these sources of noise would greatly improve its precision. Moreover, osnoise comes together with timerlat [14], a tool that assesses scheduling latency. A desirable future work would be to expand its functionality to account for hypervisor-induced scheduling latency as it could significantly improve evaluations of real-time applications within virtualized settings.

References

[1] J. T. Brown G. von Laszewski A. J. Younge, R. Henschel. Analysis of virtualization technologies for high performance computing environments. *IEEE International Conference*, pages 9–16, 7 2011.

[2] A. I. Avetisyan A. O. Kudryavtsev, V. K. Koshelev. Prospects for virtualization of high-performance x64 systems. *Programming and Computer Software*, 72:196–207, 2023.

[3] F. Bellard. Qemu, a fast and portable dynamic translator. *Proceedings of the annual conference on USENIX Annual Technical Conference*, page 41, 2005.

[4] T. Cucinotta D. B. De Oliveira, D. Casini. Operating system noise in the linux kernel. *IEEE Transactions on Computers*, 72:196–207, 2023.

[5] P. E. Verıssimo D. Kreutz, F. M. V. Ramos. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103:14–76, 2015.

[6] NAS Division. Nas parallel benchmarks. https://www.nas.nasa.gov/software/npb.html.

[7] W. Fornaciari F. Reghenzani, G. Massari. The real-time linux kernel: A survey on preempt_rt. *ACM Computing Surveys*, 52:1–36, 2019.

[8] Kumar S. Raj H. Schwan K. et al. Gavrilovska, A. High-performance hypervisor architectures: Virtualization in hpc systems. *HPCVirt*, pages 1–8, 2007.

[9] E. V. Hensbergen. The effect of virtualization on os interference. *IEEE Transactions on Computers*, 72:196–207, 2023.

[10] D. Faggioli L. Abeni. An experimental analysis of the xen and kvm latencies. *IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, pages 18–26, 5 2019.

[11] S. Hoenisch L. Mandyam. Ran workload performance is equivalent on bare metal and vsphere. https://blogs.vmware.com/telco/ran-workload-performance-tests-on-vmware-vsphere/.

[12] L. De Simone S. Rosiello M. Cinque, D. Cotroneo. Virtualizing mixed-criticality systems: A survey on industrial trends and issues. *Future Generation Computer Systems*, 129:315–330, 2022.

[13] R. Minnich M. Sottile. Analysis of microbenchmarks for performance tuning of clusters. *Cluster Computing, 2004 IEEE*, 2004.

[14] D. B. De Oliveira. Timerlat tracer. https://docs.kernel.org/trace/timerlat-tracer.html.

[15] D. Koester P. Luszczek1, J. J. Dongarra. Introduction to the hpc challenge benchmark suite. https://www.osti.gov/servlets/purl/860347.

[16] SDN and OpenFlow World Congress. Network functions virtualisation – introductory white paper – an introduction, benefits, enablers, challenges call for action. https://portal.etsi.org/NFV/NFV White Paper.pdf.

[17] Peter Xu. Oslat. https://github.com/xzpeter/oslat.