Retrofitting VMM Optimizations to Confidential VMs

Yuma Okamura, Kenta Ishiguro*, Terufumi Hata, Kenji Kono
Keio University, *Grenoble INP - UGA

Motivation. Confidential Virtual Machines (CVMs) are attracting attention in both academic and industry. CVMs protect the confidentiality and integrity of code, data, and CPU registers within virtual machines, even in scenarios where the hypervisor may be malicious. A significant advantage of CVMs is that they ensure backward compatibility for user-level applications.

A key drawback of CVMs is that they limit well-known hypervisor optimizations. Hypervisors typically analyze the CPU registers and memory contents of guest virtual machines (VMs) to infer their internal states and optimize the hypervisors' behaviors accordingly. Since CVMs prevent hypervisors from accessing the guest's registers and memory, the hypervisor cannot infer the guest's internal states, which hampers hypervisor optimizations.

AMD SEV-ES/SNP. Major cloud vendors all support CVMs based on AMD SEV-ES/SNP, which is designed to allow guest operating systems (OSes) to provide the hypervisor with the information necessary to handle VMExits. When a CVM exits to the hypervisor, the processor triggers an exception, called #VC exception, to the guest VM if some of the register values are necessary for the hypervisor to handle the VMExit. In response, the guest VM sends the required register values in plaintext through a communication channel referred to as GHCB (Guest-Hypervisor Communication Block).

AMD defines a set of events that trigger #VC exceptions. Non-Automatic Exits (NAEs) are VMExit events that trigger #VC exceptions, while Automatic Exits (AEs) do not. Guest OSes must be modified to provide information to the hypervisor when responding to #VC exceptions (Fig. 1).

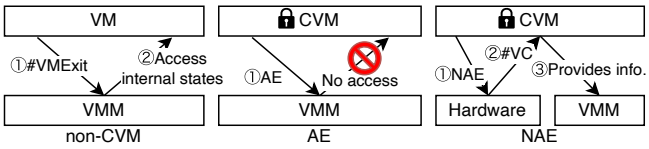

Figure 1. AE and NAE

Key Observations. *Automatic or Non-Automatic Exits should be configurable.* In the current design, AEs cannot be converted into NAEs (NAEs can be configured to be AEs). If an event is defined as an AE, the guest cannot receive #VC exceptions upon VMExits due to that event. Since NAEs allow guest OSes to handle VMExits via #VC exceptions, they can be leveraged to retrofit existing hypervisor optimizations to CVMs.

A typical example is Pause Filtering (PF), which should be redefined as NAE. PF is a hardware mechanism that detects excessive vCPU (virtual CPU) spinning in the guest, which occurs when a vCPU waits in a tight loop, awaiting the completion of a short synchronous task by a descheduled vCPU. A recently proposed solution [1] cannot be applied to AMD SEV because PF is not classified as NAE.

Limited form of paravirtualization through Non-Automatic Exits. NAEs can also be leveraged to incorporate a limited form of paravirtualization. The information exchanged through the GHCB is not architectural and can be defined entirely in software. If the guest OS exposes more information than non-CVM guests, there is potential for CVMs to perform better than fully virtualized non-CVM guests.

Solution. This poster presents vNAE (Virtual Non-Automatic Exit), a software-based mechanism that converts AEs to NAEs. vNAE explicitly triggers a #VC exception even when the hypervisor is invoked via AEs, allowing the guest OS to handle VMExits caused by PF. For experimental purposes, we have implemented some optimizations to mitigate excessive vCPU spinning. First, the vCPU's privilege level is communicated to the hypervisor, which is possible if the hypervisor can access the vCPU's registers. Second, the guest OS determines which vCPU should be scheduled to resolve the excessive spinning. This optimization represents a form of paravirtualization, whereby the guest OS provides information that cannot be easily deduced from CPU registers or memory contents.

We conducted experiments to evaluate the effectiveness of the optimizations using the vips benchmark from the Parsec. Results indicate that due to the mitigation of excessive spinning, scheduling latency decreases significantly. As shown in Fig. 2, the 50th, 90th, and 99th percentile latencies with the optimization are 61 μs, 529 μs, and 747 μs, respectively, compared to 265 μs, 671 μs, and 882 μs without the optimization.

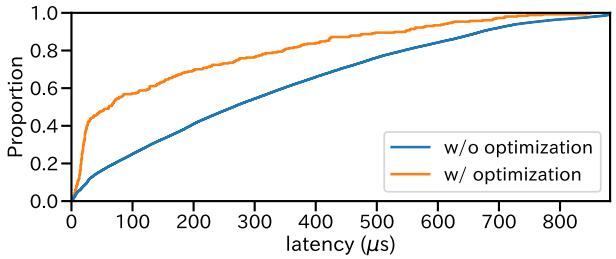

Figure 2. Scheduling latency (CDF)

References

[1] Kenta Ishiguro, Naoki Yasuno, Pierre-Louis Aublin, and Kenji Kono. Revisiting vm-agnostic kvm vcpu scheduler for mitigating excessive vcpu spinning. *IEEE Transactions on Parallel and Distributed Systems*, 34(10):2615–2628, 2023.