Contribution ID: **48** Type: **not specified**

Mediated passthrough vPMU for KVM

Thursday, 19 September 2024 10:30 (30 minutes)

BACKGROUND

KVM has supported vPMU for years as the emulated vPMU. In particular, KVM presents a virtual PMU to guests where accesses to PMU get trapped and converted into perf events. These perf events get scheduled along with other perf events at the host level, sharing the HW resource. In the emulated vPMU design, KVM is a client of the perf subsystem and has no control of the HW PMU resource at host level.

This emulated vPMU has these drawbacks:
Poor performance. Existing emulated vPMU has a terrible performance [1]. When guest PMU is multiplexing its counters, the situation is even worse, ie., KVM wastes the majority of time re-creating/starting/releasing KVM perf events, leading to significant performance degradation.

Silent error. Guest perf events's backend may be swapped out or disabled silently. This is because the host perf scheduler treats KVM perf events and other host perf events equally, they will contend HW resources. KVM perf events will be inactive when all HW resources have been owned by host perf events. But KVM can not notify this backend error into guests, this silent error is a red flag for vPMU as a production.

Hard to add new vPMU features. For each vPMU new feature, KVM may need to emulate new PMU MSRs, this involves changes to the perf API. Vendor specific changes that complicate the perf API are hard to be accepted. In addition, the whole design becomes complicated because of the "sharing" requirement, and makes the above "silent error" even worse. Because of these reasons, features like PEBS, vIBS, topdown are hard to be added.

New vPMU Design

In the new vPMU implementation, we pass through all PMU MSRs instructions except for event selectors (for security reasons), ie., all PMU MSR accesses directly touch the PMU HW instead of going to perf subsystem on the host. This means when guest is running, guest PMU is excluding owning the whole PMU hardware until it context switches back to the host.

For PMU Context switches, we do full context save/restore on the VM Enter/Exit boundary, in which we save the guest PMU MSR values that we pass through and restore the corresponding values for the host.

For PMI handling, our design leverages a dedicated interrupt vector for the guest PMI, i.e., When a guest is running and using PMU, PMIs for the guest are then delivered to the PMI handler (causing a VMEXIT) and then KVM injects the PMI into the guest.

With mediated passthrough vPMU design, VM can enjoy the transparency of x86 PMU HW. Our latest version integrates AMD support for mediated passthrough, making it complete for the whole x86 architecture.

Overall this new design has the following benefits:
Better performance. when guest access x86 counter PMU MSRs, no VM-exit and no host perf API call.

Exclusive ownership on PMU HW resources. Host perf events are stopped and give up HW resource at VM-entry, and restart running after VM-exit.

Easy to enable PMU new features. KVM just needs to pass through new MSRs and save/restore them at VM-exit and VM-entry, no need to add perf API.

Note, passthrough vPMU does satisfy the enterprise-level requirement of secure usage for PMU by intercepting guest access to all event selectors. In addition, the new vPMU design checks the exposure of PMU counter MSRs and decides whether to intercept RDPMC or not. We pass through RDPMC if and only if all PMU counters are exposed.

Drawbacks

The key problem of mediated passthrough vPMU is that the host user loses the capability to profile guests. If any users want to profile guests from the host, they should not enable the new vPMU mode. In particular, perf events with attr.exclude_guest = 1 will be stopped here and restarted after vm-exit. In RFCv1, these events without attr.exclude_guest=1 will be in error state, and they cannot recover back to active state even if the guest stops running. This impacts host perf a lot and requests host system wide perf events have attr.exclude_guest=1. In RFCv2, we update the design by making sure the VM can't be started when !exclude_guest events exist on host and host !exclude_guest events are blocked to create when VM has been started. In addition, exclude_guest attribute would be set by default when perf tool creates events.

Open Discussions

Other than the above, there are several open topics under intensive discussion:

NMI watchdog. the perf event for NMI watchdog is a system wide cpu pinned event, it will be stopped also during vm running, but it doesn't have attr.exclude_guest=1, we add it in this RFC. But this still means the NMI watchdog loses function during VM running. Two candidates exist for replacing perf event of NMI watchdog:
Buddy hardlock detector[3] may be not reliable to replace perf events.
HPET-based hardlock detector [4] isn't in the upstream kernel.

Dedicated kvm_pmi_vector. In emulated vPMU, host PMI handlers notify KVM to inject a virtual PMI into the guest when the physical PMI belongs to the guest counter. If the same mechanism is used in passthrough vPMU and PMI skid exists which causes physical PMI belonging to guests to happen after VM-exit, then the host PMI handler couldn't identify this PMI belongs to host or guest. So this RFC uses a dedicated kvm_pmi_vector, PMI belonging to guests has this vector only. The PMI belonging to the host still has an NMI vector.

The location of the PMU context switch. There is an intensive discussion on the location of the PMU context switch. Existing implementation does the context switch at VM-enter/exit boundary. This generates a moderate performance overhead per VMEXIT due to PMU register reads and writes. One alternative idea is to put the PMU context switch location to the VCPU_RUN loop boundary. However, the downside of that is the missing functionality of profiling KVM code within the VCPU_RUN loop. The debate is still ongoing.

References

[1] Efficient Performance Monitoring in the Cloud with Virtual Performance Monitoring Units (PMUs) https://static.sched.com/hosted_file
[2] Mediated Passthrough vPMU v1 https://lore.kernel.org/all/20240126085444.324918-1-xiong.y.zhang@linux.intel.com/
[3] Mediated Passthrough vPMU v2 https://lore.kernel.org/all/20240506053020.3911940-1-mizhang@google.com/

Primary authors: MI, Dapeng (Intel); MATTSON, Jim (Google); LIANG, Kan (Intel); SHUKLA, Manali (AMD); ZHANG, Mingwei (Google); DADHANIA, Nikunj (AMD); ALT, Samantha (Intel); DAS, Sandipan (AMD); SHUKLA, Santosh (AMD); CHRISTOPHERSON, Sean (Google); ERANIAN, Stephane (Google); ZHANG, Xiong (Intel); WANG, Zhenyu (Intel)

Presenter: ZHANG, Mingwei (Google)

Session Classification: KVM Microconference

Track Classification: KVM Microconference