Paravirt Scheduling with BPF

Vineeth Pillai <vineeth@bitbyteword.org>
Joel Fernandes <joel@joelfernandes.org>

Google

A little background

- Double scheduling: Host schedules vcpu threads and the guest schedules the tasks running inside the guest
- But both schedulers are unaware of the other
  - Hosts schedules vcpu threads without knowing what's being run on the vcpu
  - Guest schedules tasks without knowing where the vcpu is running physically
- vCPUs are regular CFS tasks in the host and does not get to run in a timely fashion when the host is experiencing load
- Host scheduler tries to be fair and doesn’t know about the priority requirements
- This can cause issues with latencies, power consumption, resource utilization etc.

Paravirt Scheduling

- Cooperative scheduling framework where host and guest shares scheduling related information for making optimal scheduling decisions (priority, placement etc).
- Shared memory setup between guest and host for communication.
- Paravirt scheduling protocol
  - shared memory layout, details of information shared, scheduling policy decisions etc
- During guest boot up, guest and host go through a handshake process to determine the protocol, communicating the shared memory address etc
- Once the handshake is successful, scheduling information is shared and scheduling decisions are taken based on the information and policies defined.

Paravirt Scheduling: Shared memory page

- Divided into three regions
  - Header
    - Information about the protocol id, version etc
  - Guest Area
    - Information populated by guest and read by host
  - Host Area
    - Information populated by host and read by guest
- Guest allocates the page and shares the GFN with host counterpart

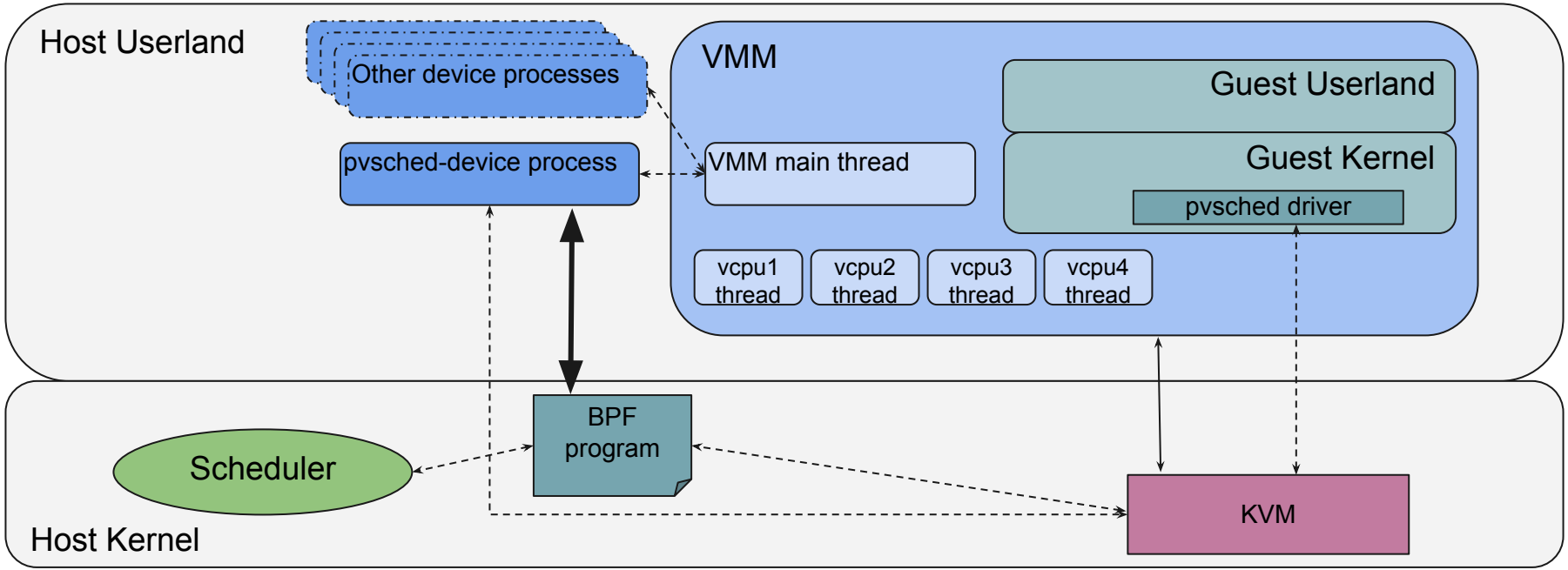
Paravirt Scheduling: History

- V1: https://lkml.org/lkml/2023/12/13/1789
  - Kvm does most of the heavy lifting: handshake, policies, scheduling decision
  - Upstream was against having all these logic in kvm
- V2: https://lwn.net/Articles/968242/
  - Kvm does the handshake with the guest
  - Policy and scheduling decision designed to be implemented separately as a kernel module or a BPF program
  - module/BPF registers to kvm for receiving callbacks on interested events
  - Kvm maintainers does not like the idea of having the handshake also in kernel
- V3: (in works)
  - Handshake in the VMM
  - Policies and scheduling decisions in a BPF program (loaded by the VMM)
    - Could be a kernel module as well

Paravirt Scheduling: A sample use case

- Latency sensitive guest workloads
  - When guest is about to run a latency sensitive workload, it updates the guest area in the shared memory requesting host not to schedule it out.
  - Guest continues running until a VMEXIT.
  - Immediately on VMEXIT, host see the guest request and boosts the vcpu task priority. It also updates the host area with the decision it took.
- Interrupt Injection
  - During an interrupt injection, host proactively boosts the the vcpu tasks priority and updates the host area with the action.

V3 Design


Host Userland
Other device processes
pvsched-device process
VMM
VMM main thread
Guest Userland
Guest Kernel
pvsched driver
vcpu1 thread
vcpu2 thread
vcpu3 thread
vcpu4 thread
BPF program
Scheduler
KVM
Host Kernel

V3 Design : BPF program

- Receives required information from the VMM through maps
  - Shared memory address (GFN)
  - Pids and cpuids of the vcpus
- Registers to kvm for callbacks on interested events
  - VMEXIT, VMENTRY, HALT, Interrupt injection etc
- Implements the policies and executes scheduling decisions on callbacks

BPF Program: Kvm Callbacks

- struct_ops ?
- Raw trace points ?
  - Internal trace points in kvm not exposed to userland

BPF Program: Shared memory

- VMM receives the PFN from guest and passes it on to BPF program
- GFN to PFN conversion helpers in BPF?
    - Use a kfunc wrapper of the kvm helper

BPF Program: Scheduler hooks

- BPF program needs to call scheduler APIs for priority management and placement
  - sched_setscheduler(), …
- kfunc?