How Fast Can We Go? Booting a Linux VM to Userspace in 100ms and Beyond

Robert Bradford
Intel
👍 64ms
Fin.
1. Your numbers may vary. Numbers² go up and down. Past performance does not predict future performance. Always read the label.

2. NaN NaN NaN NaN NaN. BATMAN BATMAN!
Motivation

• Why aim to boot fast?
  – Enabling modern cloud use cases: FaaS, CaaS (Function/Container as a Service)
  – Improve user experience
  – Reduction in resource consumption; reducing environmental impact
  – Boot optimisation can improve steady state performance
  – Intellectual curiosity
How to define boot time

• Potential options:
  – Time from...
    • Management layer
    • Starting VMM...
    • Starting VM...
    • Accepting a HTTP request...
  – Time to...
    • … start firmware
    • … start kernel
    • … start userspace
    • … start user's workload
    • … respond to HTTP request

Our focus today
Measuring boot time

- Instrument kernel to cause VM exit
- Log VM exits with timestamp relative to VM start
- In Cloud Hypervisor we use I/O port 0x80 and write to our log
- Alternative using perf: https://github.com/stefano-garzarella/qemu-boot-time
Tooling

- Frequency domain – perf, counters
- Time domain – tracing, VMM logs, guest dmesg
Our adventure begins

- Initial test:
  - cloud-hypervisor --kernel
    ~/src/linux/vmlinux --disk path=jammy.raw --
    --cmdline "root=/dev/vda1 console=ttys0" --
    serial tty --console off --memory size=512M
  - Boot time: 166ms
Our adventurer enters the caverns

perf shows: PIO exits for serial console!
Our adventurer probes deeper

- Use virtio-console
  - cloud-hypervisor --kernel
    ~/src/linux/vmlinux --disk path=jammy.raw --
    --cmdline "root=/dev/vda1 console=hvc0" --
    serial null --console tty --memory
    size=512M
  - Boot time: 94ms (was 166ms)
Our adventurer seeks some silence

- Use virtio-console and quiet
  - cloud-hypervisor --kernel
    ~/src/linux/vmlinux --disk path=jammy.raw --cmdline "root=/dev/vda1 console=hvc0 quiet" --serial null --console tty --memory size=512M
  - Boot time: 89ms (was 94ms)
A new foe arrives

The block device now dominates the perf output:

|reno| 56.64| 0.00| _disk0_q0| cloud-hypervisor| std::sys::unix::thread::Thread::new::thread_start
|reno| 56.64| 0.00| _disk0_q0| cloud-hypervisor| core::ops::function::FnOnce::call_once{vtable.shim}
|reno| 56.64| 0.00| _disk0_q0| cloud-hypervisor| std::sys_common::backtrace::__rust_begin_short_backtrace
|reno| 56.50| 0.17| _disk0_q0| cloud-hypervisor| virtio_devices::epoll_helper::EpollHelper::run_with_timeout
|reno| 54.50| 6.28| _disk0_q0| cloud-hypervisor| <virtio_devices::block::BlockEpollHandler as virtio_devices::epoll_helper>
|reno| 27.57| 11.18| _disk0_q0| cloud-hypervisor| virtio_devices::block::BlockEpollHandler::process_queue_submit
|reno| 9.78 | 0.00| _disk0_q0| [unknown] | 0x0000000000000001
|reno| 9.29 | 9.29| _disk0_q0| cloud-hypervisor| <block_util::raw_async::RawFileAsync as block_util::async_io::AsyncIo>
|reno| 8.69 | 1.25| _disk0_q0| cloud-hypervisor| vm_memory::guest_memory::impl vm_memory::bytes::Bytes<vm_memory::guest
|reno| 6.96 | 4.60| _disk0_q0| libc.so.6 | read
|reno| 6.25 | 6.25| _disk0_q0| cloud-hypervisor| vm_memory::volatile_memory::copy_slice_impl::copy_slice
|reno| 5.65 | 0.00| payload_loader| cloud-hypervisor| std::sys::unix::thread::Thread::new::thread_start
|reno| 5.65 | 0.00| payload_loader| cloud-hypervisor| core::ops::function::FnOnce::call_once{vtable.shim}
|reno| 5.65 | 0.00| payload_loader| cloud-hypervisor| std::sys_common::backtrace::__rust_begin_short_backtrace
|reno| 5.65 | 0.00| payload_loader| cloud-hypervisor| vmm::vm::Vmm::load_kernel
|reno| 5.65 | 5.65| payload_loader| cloud-hypervisor| vm_memory::guest_memory::impl vm_memory::bytes::Bytes<vm_memory::guest
|reno| 5.45 | 5.45| _disk0_q0| libc.so.6 | _int_free
|reno| 5.26 | 0.00| _disk0_q0| cloud-hypervisor| <virtio_devices::transport::pci_device::VirtioInterruptMsix as virtio_device::
|reno| 5.05 | 5.05| _disk0_q0| libc.so.6 | malloc
|reno| 4.91 | 0.69| _disk0_q0| libc.so.6 | __GI__libc_write
|reno| 4.91 | 0.00| _disk0_q0| [unknown] | 0xfffffffffae80000

---

[Image: The Linux Foundation]
Unexpected plot twist

- Switch to virtio-pmem
  - cloud-hypervisor --kernel
    ~/src/linux/vmlinux --pmem file=jammy.raw --
    cmdline "root=/dev/pmem0p1 console=hvc0 quiet" --serial null --console tty --memory
    size=512M
  - Boot time: 110ms (was 89ms)
The plot thickens

Handling page faults now dominates the profile:

```
- 63.38%  0.00%  vcpu0  [kvm]  [k]
  kvm_vcpu_ioctl
    - kvm_arch_vcpu_ioctl_run
      - 48.51%  vmx_handle_exit
        - 46.52%  kvm_mmu_page_fault
          - 45.48%  direct_page_fault
            + 41.24%  kvm_faultin_pfn
            + 3.06%  kvm_tdp_mmu_map
            + 0.68%  kvm_tdp_page_fault
          0.62%  handle_ept_violation
        + 10.52%  vmx_vcpu_run
      0.74%  __srcu_read_lock
      0.51%  __srcu_read_unlock
```
Our hero is victorious

• Switch to 2MiB huge pages
  – cloud-hypervisor --kernel
    ~/src/linux/vmlinux --pmem file=jammy.raw --
    cmdline "root=/dev/pmem0p1 console=hvc0
    quiet" --serial null --console tty --memory
    size=512M,hugepages=on
  – Boot time: 64ms (was 110ms)
Optimised VM config

VM Exits / Boot time

- Serial console
- Virtio console
- Quieter
- Virtio-pmem
- Huge pages

- CPUID
- CR_ACCESS
- DR_ACCESS
- EPT_MISCONFIG
- EPT_VIOLATION
- EXTERNAL_INTERRUPT
- HLT
- IO_INSTRUCTION
- MSR_READ
- MSR_WRITE
- PAUSE_INSTRUCTION
- PREEMPTION_TIMER
- VMCALL
- XSETBV
- Boot time
What about THP?

"Getting a SCSI chain working is perfectly simple if you remember that there must be exactly three terminations: one on one end of the cable, one on the far end, and the goat, terminated over the SCSI chain with a silver-handled knife whilst burning black candles." -- Anthony DeBoer
Cloud Hypervisor: Key optimisations

- MCFG for PCI buses count
- Fast path for most common VM exits (PCI I/O port)
- Async kernel loading
ACPI MCFG Bus Count

- ACPI MCFG static table that provides PCI segment details
- Cloud Hypervisor only has 1 bus per segment (up to 16 segments)
- Specifying 1 bus in MCFG entry makes Linux kernel skip scanning other buses
PCI I/O Port Fast Path

- Bus model similar to Firecracker & CrosVM
- Resolving address on PCI bus has a cost to find the device to handle the PIO (visible in perf)
- Fast path bypasses and sends direct to device during KVM_RUN exit handling
- Total optimisation impact for boot PCI handling: 102ms to 64ms (37%)
Tracing: Async kernel loading

- Trace points added at key locations
- Added asynchronous loading to parallelise activity before VM booting
Cloud Hypervisor: Monitoring

• Run a peformance metrics test suite on bare metal system
• Variety of tests: boot time, network throughput & latency,
• https://www.cloudhypervisor.org/metrics
Cloud Hypervisor: Monitoring

boot_time_pmem_ms

Mar 27 2022
Apr 10
Apr 24
May 8
May 22
Jun 5
Jun 19
Jul 3
Jul 17
Jul 31
Aug 14
commit 71020a3c0dff4a00d96922a4a95a067f524a7dcb
Author: Thomas Gleixner <tglx@linutronix.de>
Date:  Mon Dec 6 23:51:15 2021 +0100

PCI/MSI: Use msi_add_msi_desc()

Simplify the allocation of MSI descriptors by using msi_add_msi_desc() which moves the storage handling to core code and prepares for dynamic extension of the MSI-X vector space.

Signed-off-by: Thomas Gleixner <tglx@linutronix.de>
Tested-by: Michael Kelley <mikelley@microsoft.com>
Tested-by: Nishanth Menon <nm@ti.com>
Reviewed-by: Jason Gunthorpe <jgg@nvidia.com>
Acked-by: Bjorn Helgaas <bhelgaas@google.com>
Link: https://lore.kernel.org/r/20211206210748.035348646@linutronix.de

drivers/pci/msi/msi.c | 104 +++++++++++++++++++++++-----------------------------
1 file changed, 47 insertions(+), 57 deletions(-)
Below 50ms

- Reduce kernel config
  - Who needs networking anyway?
  - Smaller config = smaller binary = shorter load time
- Very fast storage (pmem'esque e.g. Optane DC)
Summary

- Custom tooling may be useful (i.e. tracing)
- Is boot time optimisation valuable?:
  - Kubernetes takes seconds to start a pod
  - Easily lost in kernel changes
- Intrinsic vs extrinsic motivation
- Automated monitoring
- Booting fast can be an alternative to templating
Counting VM exits

- perf record -e kvm:kvm_exit
target/release/cloud-hypervisor --kernel
~/.src/linux/vmlinux --pmem
file=~/.workloads/jammy-l1.raw --cmdline
"root=/dev/pmem0p1 console=hvc0 quiet" -
-serial null --console tty --memory
size=512M,hugepages=on

- perf script | grep "reason" | awk
  '{print $7}' | sort | uniq -c | awk
  '{print $2"\t"$1}' | sort