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

Robert Bradford
Intel

64ms

THE LINUX FOUNDATION

Fin.

THE LINUX FOUNDATION

64ms[1]

1. Your numbers may vary. Numbers[2] go up and down. Past performance does not predict future performance. Always read the label.
2. NaNaNaNaNaNaN. BATMAN BATMAN!

THE LINUX FOUNDATION

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

THE LINUX FOUNDATION

How to define boot time

- Potential options:
  - Time from...
    - Management layer
    - Starting VMM...
    - Starting VM...
    - Accepting a HTTP request...

    } Our focus today
  - Time to...
    - … start firmware
    - … start kernel
    - … start userspace
    - … start user's workload
    - … respond to HTTP request

    } Our focus today

THE LINUX FOUNDATION

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

THE LINUX FOUNDATION

Tooling

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

THE LINUX FOUNDATION

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

THE LINUX FOUNDATION

Our adventurer enters the caverns

`perf` shows: PIO exits for serial console!

```
- 46.88% <hypervisor::kvm::KvmVcpu as hypervisor::cpu::Vcpu>::run
   - 29.23% <vmm::vm::VmOpsHandler as hypervisor::vm::VmOps>::pio_write
      - 27.84% vm_device::bus::Bus::write
         - 8.18% vm_device::bus::Bus::resolve
              alloc::collections::btree::search::<impl alloc::collections::btree::node::NodeRef<BorrowType,K,V,Type>>::find_up
         - 4.88% <devices::legacy::serial::Serial as vm_device::bus::BusDevice>::write
            - <vmm::interrupt::LegacyUserspaceInterruptGroup as vm_device::interrupt::InterruptSourceGroup>::trigger
               - 2.97% <devices::ioapic::Ioapic as devices::interrupt_controller::InterruptController>::service_irq
                  - 1.40% <vmm::interrupt::MsiInterruptGroup as vm_device::interrupt::InterruptSourceGroup>::trigger
                       0.54% core::hash::BuildHasher::hash_one
                    0.67% __GI___libc_write
           4.85% <std::io::stdio::Stdout as std::io::Write>::write_all
           2.08% <std::io::stdio::Stdout as std::io::Write>::flush
           1.55% __memrchr_avx2
           1.36% __GI___libc_write
           1.15% __GI___pthread_disable_asynccancel
           0.96% __GI___pthread_enable_asynccancel
           0.54% std::io::buffered::bufwriter::BufWriter<W>::flush_buf
   - 8.51% <vmm::vm::VmOpsHandler as hypervisor::vm::VmOps>::pio_read
      - 7.36% vm_device::bus::Bus::read
         - 4.74% vm_device::bus::Bus::resolve
              alloc::collections::btree::search::<impl alloc::collections::btree::node::NodeRef<BorrowType,K,V,Type>>::find_up
           0.75% <devices::legacy::serial::Serial as vm_device::bus::BusDevice>::read
```

THE LINUX FOUNDATION

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)

THE LINUX FOUNDATION

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)

THE LINUX FOUNDATION

A new foe arrives

The block device now dominates the perf output:

```
+   56.64%     0.00%  _disk0_q0         cloud-hypervisor          [.] std::sys::unix::thread::Thread::new::thread_start
+   56.64%     0.00%  _disk0_q0         cloud-hypervisor          [.] core::ops::function::FnOnce::call_once{{vtable.shim}}
+   56.64%     0.00%  _disk0_q0         cloud-hypervisor          [.] std::sys_common::backtrace::__rust_begin_short_backtrace
+   56.50%     0.17%  _disk0_q0         cloud-hypervisor          [.] virtio_devices::epoll_helper::EpollHelper::run_with_timeout
+   54.50%     6.28%  _disk0_q0         cloud-hypervisor          [.] <virtio_devices::block::BlockEpollHandler as virtio_devices::epoll_help
+   27.57%    11.18%  _disk0_q0         cloud-hypervisor          [.] virtio_devices::block::BlockEpollHandler::process_queue_submit
+    9.78%     0.00%  _disk0_q0         [unknown]                 [.] 0x0000000000000001
+    9.29%     9.29%  _disk0_q0         cloud-hypervisor          [.] <block_util::raw_async::RawFileAsync as block_util::async_io::AsyncIo>:
+    8.69%     1.25%  _disk0_q0         cloud-hypervisor          [.] vm_memory::guest_memory::<impl vm_memory::bytes::Bytes<vm_memory::guest
+    6.96%     4.60%  _disk0_q0         libc.so.6                 [.] read
+    6.25%     6.25%  _disk0_q0         cloud-hypervisor          [.] vm_memory::volatile_memory::copy_slice_impl::copy_slice
+    5.65%     0.00%  payload_loader    cloud-hypervisor          [.] std::sys::unix::thread::Thread::new::thread_start
+    5.65%     0.00%  payload_loader    cloud-hypervisor          [.] core::ops::function::FnOnce::call_once{{vtable.shim}}
+    5.65%     0.00%  payload_loader    cloud-hypervisor          [.] std::sys_common::backtrace::__rust_begin_short_backtrace
+    5.65%     0.00%  payload_loader    cloud-hypervisor          [.] vmm::vm::Vm::load_kernel
+    5.65%     5.65%  payload_loader    cloud-hypervisor          [.] vm_memory::guest_memory::<impl vm_memory::bytes::Bytes<vm_memory::guest
+    5.45%     5.45%  _disk0_q0         libc.so.6                 [.] _int_free
+    5.26%     0.00%  _disk0_q0         cloud-hypervisor          [.] <virtio_devices::transport::pci_device::VirtioInterruptMsix as virtio_d
+    5.05%     5.05%  _disk0_q0         libc.so.6                 [.] malloc
+    4.91%     0.69%  _disk0_q0         libc.so.6                 [.] __GI___libc_write
+    4.91%     0.00%  _disk0_q0         [unknown]                 [.] 0xffffffffaee00000
```

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 LINUX FOUNDATION

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
```

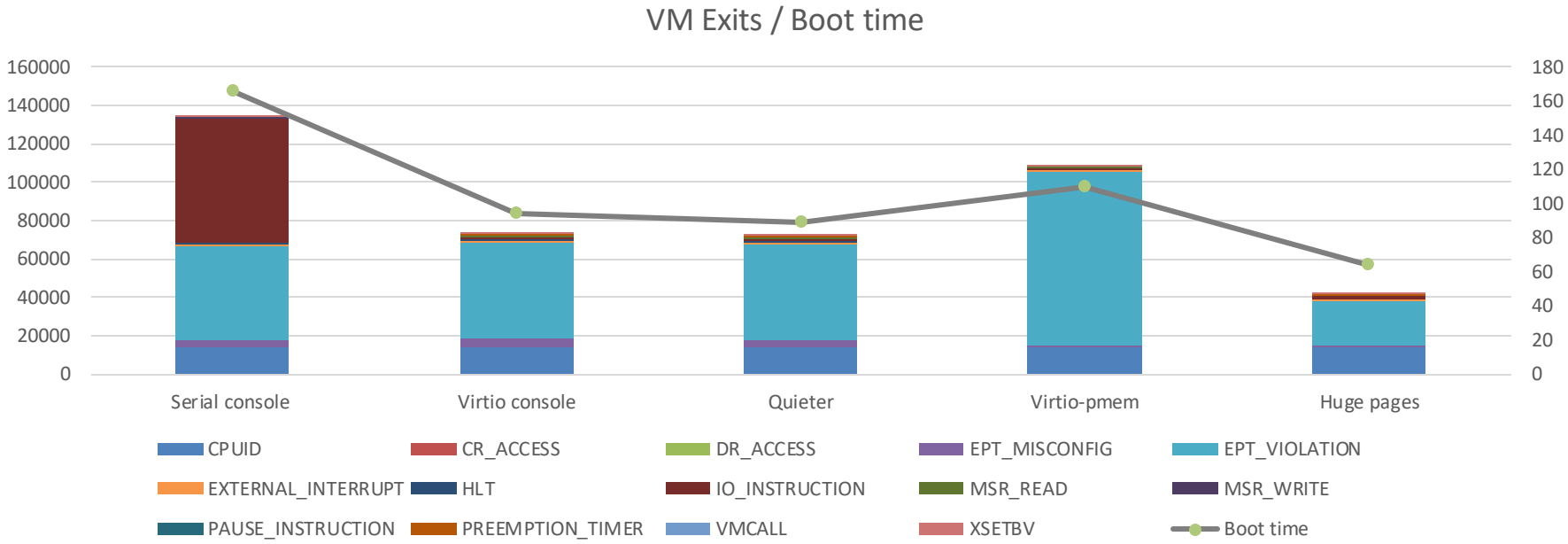
THE LINUX FOUNDATION

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)

THE LINUX FOUNDATION

Optimised VM config



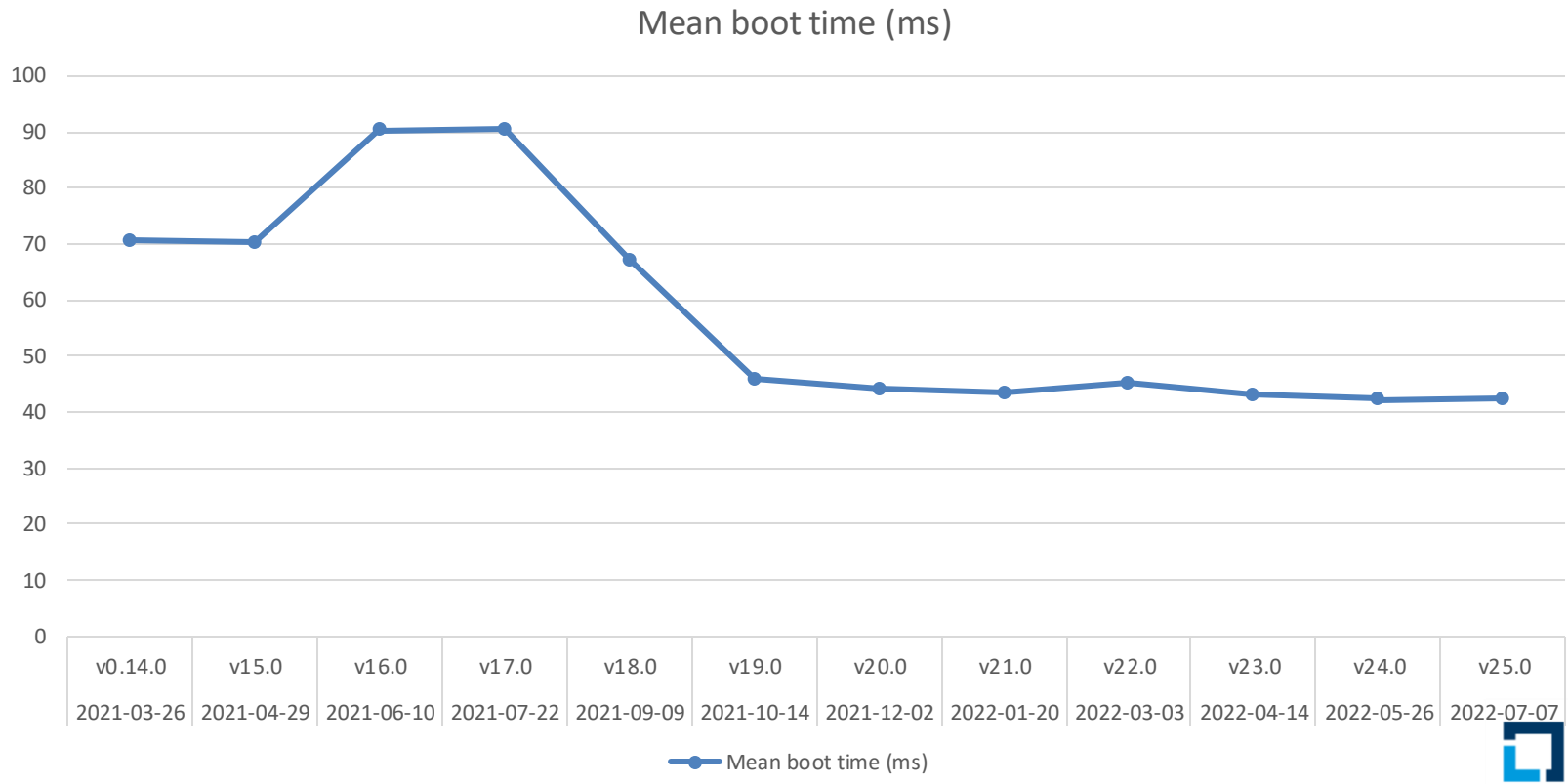
THE LINUX FOUNDATION

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*

THE LINUX FOUNDATION

Cloud Hypervisor: History



THE LINUX FOUNDATION

Cloud Hypervisor: Key optimisations

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

THE LINUX FOUNDATION

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

THE LINUX FOUNDATION

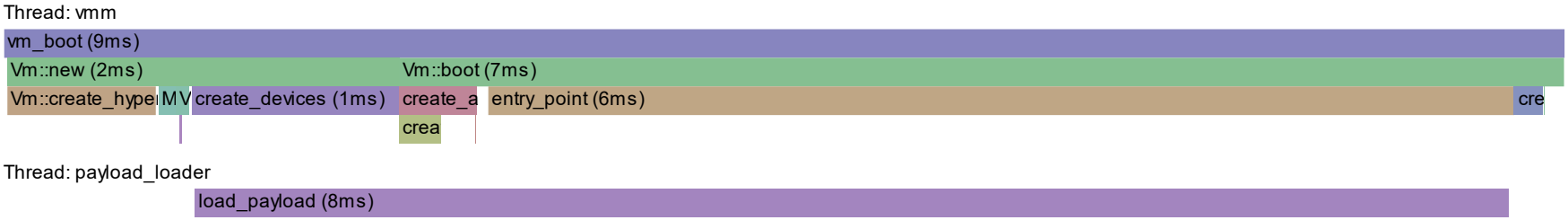
PCI I/O Port Fast Path

- 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%)

THE LINUX FOUNDATION

Tracing: Async kernel loading

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



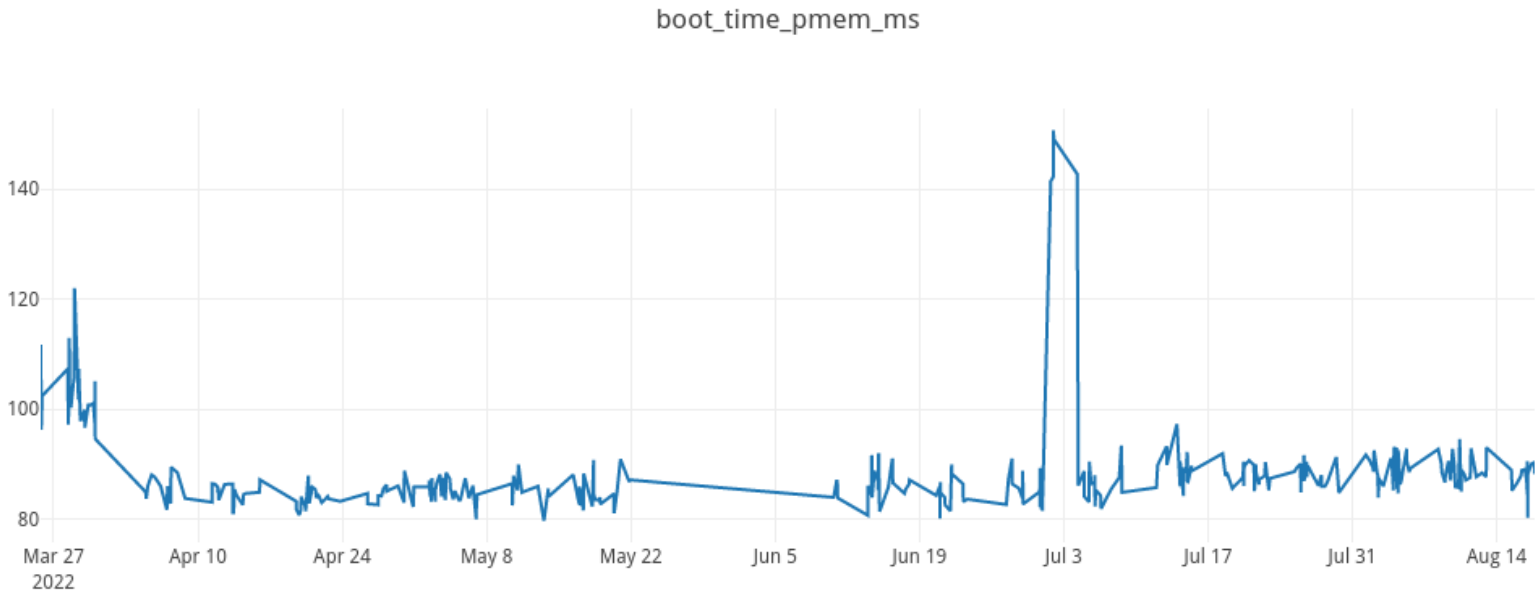
THE LINUX FOUNDATION

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

THE LINUX FOUNDATION

Cloud Hypervisor: Monitoring



LINUX FOUNDATION

What happened?

```
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(-)
```

THE LINUX FOUNDATION

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)

THE LINUX FOUNDATION

Summary

- Custom tooling may 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 alternative to templating

THE LINUX FOUNDATION

KVM FORUM

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
  ```

THE LINUX FOUNDATION