Improving kexec boot time

Usama Arif
System Technology & Engineering
LPC 2023, Richmond, VA
Agenda

• Motivation
• Boot time improvement strategies
  • Parallel smpboot
  • Optimizing TSC synchronization
  • streamline the initialization of struct pages when using HVO
  • PCI probe skipping
• Is there anything more that can be done to improve kexec time?
Motivation

• VMs with extended running time in the cloud must operate within a secure, updated hypervisor/host kernel.

• This improves:
  • Security
  • Functionality
  • Performance

• Can be done with:
  • Live migration
  • Live update
Live migration

• Move the guest from one host to another
• Can be used to solve any hardware issues on machine being migrated from
• Challenges:
  • Resource use (networking, extra buffer machines..)
  • Slowdown (for e.g. network faults if using post-copy)
  • Things going wrong during live migration (error recovery)
Live update

• Update host kernel, while staying on the same machine.

• Advantages:
  • No need for extra hardware (buffer machines)
  • No need to migrate VM/workloads to another network
  • Storage data can be easily accessed afterwards

• Issues:
  • Cannot be used if for hardware issues, only updating kernel/VMM
  • High downtime (solvable?)
  • Preserving IOMMU states during kexec for vfio-pci devices
    • https://sched.co/15jLX
  • Downtime from applications restarting DPDK/SPDK applications
    • https://sched.co/17v0u
Reasons for downtime

• VM pause
• VM snapshot
• Kexec reboot (largest time)
• VM restore
• VM resume
Measuring downtime

• Total downtime
  • Downtime in continuous data transfer from guest VM to external machine

• Kernel boot time
  • Kernel timestamp from first log with kernel version to running /init

• Test machine
  • 128 Intel Xeon CPUs
  • 2 sockets, 2 NUMA nodes
  • 512G memory
Initial boot time

Each bar in the above chart represents a timestamp log

Total time 4.3 seconds

Biggest time: Initialization of struct pages (1.7 seconds – 40%)
Defer initialization of struct pages

- Total time 2.7 seconds
- `CONFIG_DEFERRED_STRUCT_PAGE_INIT`: defer initialization of struct pages from single thread at boot to parallel when kswapd starts
- Biggest time left: smpboot (1.5s)
SMP boot

- SMP boot took place serially
- Most time in SMP boot taken by waking each CPU (SIPI/INIT/INIT) and waiting for the CPU to respond before moving to next CPU

```c
enum cpuhp_state {  
  CPUHP_INVALID = -1,

  /* PREPARE section invoked on a control CPU */  
  CPUHP_OFFLINE = 0,

  /* STARTING section invoked on the hotplugged CPU in low level */  
  CPUHP_BRINGUP_CPU,  

  /* Online section invoked on the hotplugged CPU from the hotplug thread */  
  CPUHP_ONLINE,

  CPUHP_AP_IDEottenham,  

  CPUHP_BRINGUP_CPU,  

  CPUHP_ONLINE,

  CPUHP_INVALID
};
```
Parallel smpboot

- Proposed by David Woodhouse
  - https://lore.kernel.org/all/20211215145633.5238-1-dwmw2@infradead.org
  - Rather than kick a CPU and wait for it to come online one by one, kick them all and wait for them to reach synchronization point
  - Resolve certain dependencies
  - Had issues with APIC in AMD CPUs

- Picked up later when it was occupying the largest chunk in boot time for us:
  - https://lore.kernel.org/all/20230328195758.1049469-1-usama.arif@bytedance.com
  - Didn’t have proper synchronization analysis
  - Microcode loading didn’t meet x86 requirements

- Reworked by Thomas Gleixner (and merged!)
  - https://lore.kernel.org/all/20230512203426.452963764@linutronix.de
  - Included patch for reusing CPU0 delay calibrations for secondary CPUs
Parallel SMP boot

- Kernel boot time: 2.7s -> 1s
- SMP boot time 1.7s -> 60ms
Hugepages

• Several 1G hugepages reserved at boot time for DPDK/SPDK applications

• Effect of reserving 500 1G hugepages (worst case):

```
26.4 ms: Fallback order for Node 1: 1 0
26.4 ms: Built 2 zonelists, mobility grouping on, Total pages: 132033393
26.4 ms: Policy zone: Normal
26.4 ms: mem auto-init: stack:off, heap alloc:off, heap free:off
26.5 ms: software IO TLB: area num 128.
1291.7 ms: Memory: 1808176K/536517308K available (12288K kernel code, 1035K rdata, 3112K rodata, 1944K init, 1792K bss, 532906052K reserved, 0K cma-reserved)
1292.7 ms: SLUB: HWallgn=64, Order=0-3, MinObjects=0, CPUs=128, Nodes=2
1293.4 ms: Dynamic Preempt: none
1293.9 ms: rcu: Preemptible hierarchical RCU implementation.
1293.9 ms: rcu: RCU event tracing is enabled.
1293.9 ms: rcu: RCU restricting CPUs from NR_CPUS=240 to nr_cpu_ids=128.

Total: 3888 ms
```
HVO (Hugepage Vmemap Optimize)

- 262144 struct pages initialized per 1G hugepage
- Memory optimization to not use `struct page` for every physical page frame
- Only the first `PAGE_SIZE`/`sizeof(struct page)` (64) struct pages are needed
- Rest of the tail pages contain the same information
- HVO frees them and returns them to the buddy allocator
- Why initialize these struct pages if they are to be freed later??

https://lore.kernel.org/linux-mm/20230913105401.519709-1-usama.arif@bytedance.com/
HVO (Hugepage Vmemap Optimize)

- Total boot time: 3.9s -> 1.3s
- Struct page initialization time reduced by 2.6 seconds
Specialized optimizations

• Total VM downtime (1.5-2 seconds)
  • Kernel boot represents approximately 70% (1 second)
• Low latency applications like machine learning and networking still impacted.
• p99 latency > 100ms triggers alerts on databases
• Would still be noticed by VMs deployed for public clouds?
• Can do more?
Disable purgatory in kexec

• Code that runs between the old and new kernel
  • Checks SHA256 checksum of new kernel, making sure its not corrupted
• Submitted patch to disable purgatory (by default enabled)
  • [https://lore.kernel.org/lkml/20211206164724.2125489-1-usama.arif@bytedance.com/](https://lore.kernel.org/lkml/20211206164724.2125489-1-usama.arif@bytedance.com/)
• Reduces downtime by 200ms (approx. 20% of downtime)
• Patch rejected as saving of 200ms not considered enough to justify disabling checksum
• Probably OK in production environments where its less likely go wrong??
PCI device probe skipping

• Test machine has 57 unique PCI devices
• All of them actually needed?
  • Probing + adding to IOMMU groups take time
• Only 15 needed to boot the host and provide networking and storage to VM
• Specify in kernel command line which devices are needed.
• 120ms to 40ms
• Disclaimer: Not general purpose.
Remaining time

- Initial struct page initialization: 110ms
- Deferred page initialization: 70ms
- ACPI operations (301ms):
  - Includes AML table loads (100ms), OEM table loads, enabling interpreter (100ms), finding idle states
Conclusion and future work

• Is the community interested in PCI whitelist/blacklist during boot?
• Restart purgatory discussion?
• ACPI improvements?
Thanks!