Seamless Cloud System Upgrade with VMM Fast Restart

Jason.Zeng@intel.com

Disclaimers

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request. No product or component can be absolutely secure.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

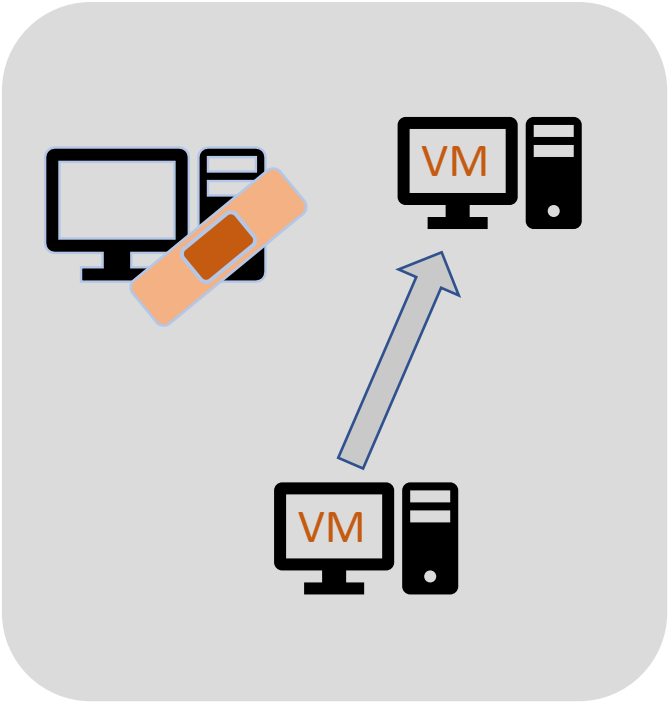
*Other names and brands may be claimed as the property of others.

The Headache of System Upgrade

- Frequent urgent security updates
  - Firmware/microcode/OS/VMM
- System update & reboot take long time
- Cloud vendor sees more service breakage to customers

Existing Solutions

- Live patching
  - Good for small fixes (no service down)
  - High failure rate for big changes
- VM live migration
  - Mature feature in most VMMs
    - Typically <1s service shutdown
  - Limitations
    - Poor support of passthrough devices
    - Unfriendly to memory-intensive workloads
    - Requires space machine to migrate into
    - Infrastructure-imposed failures (e.g. network configuration, etc.)

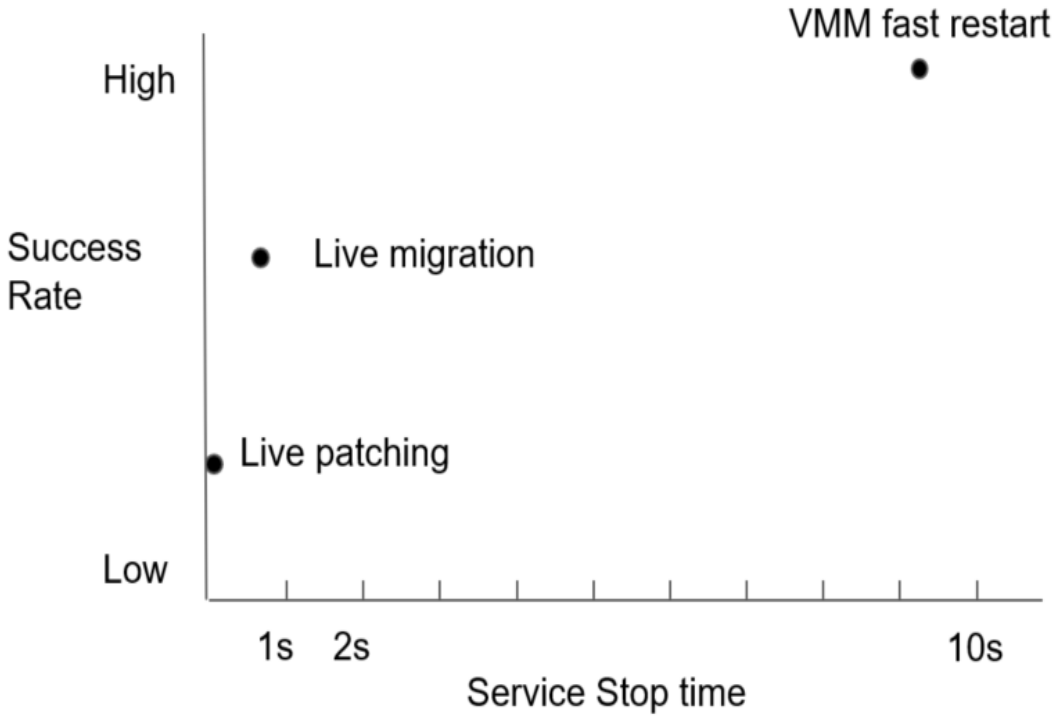


New Proposal – VMM Fast Restart

- Leverage Kexec reboot to bypass firmware
- Seamless guest state saving/reloading through DRAM-as-PMEM
- Sustain passthrough device state across reboot
- Reuse DMA/IRQ remapping in new kernel

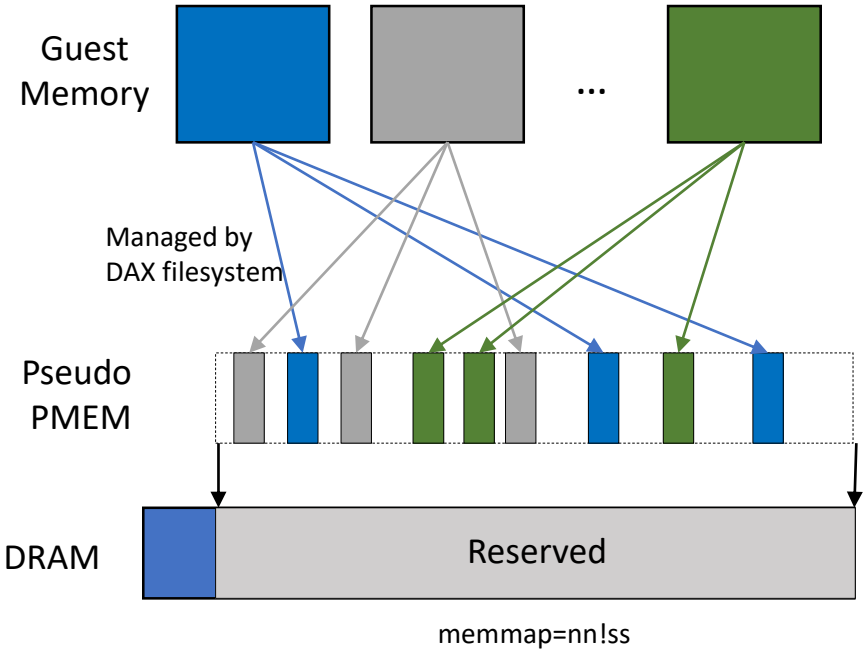
VMM Fast Restart (Cont.)

- A new option when OS/VMM upgrade is required
- Benefits
  - Higher success rate than prior approaches
  - Invisible to guest OS
    - Although with relatively longer down time
  - Support passthrough devices



DRAM-as-PMEM

- Reserve DRAM to emulate PMEM
  - https://pmem.io/2016/02/22/pm-emulation.html
- Allocate guest memory from PMEM
  - Create DAX filesystem in PMEM
  - Mmap guest memory in DAX filesystem
- Simplified memory model
  - Avoid intrusive changes to Linux MM
  - Naturally sustain VM state due to the persistent attribute



Saving/Reloading VM State across Reboot

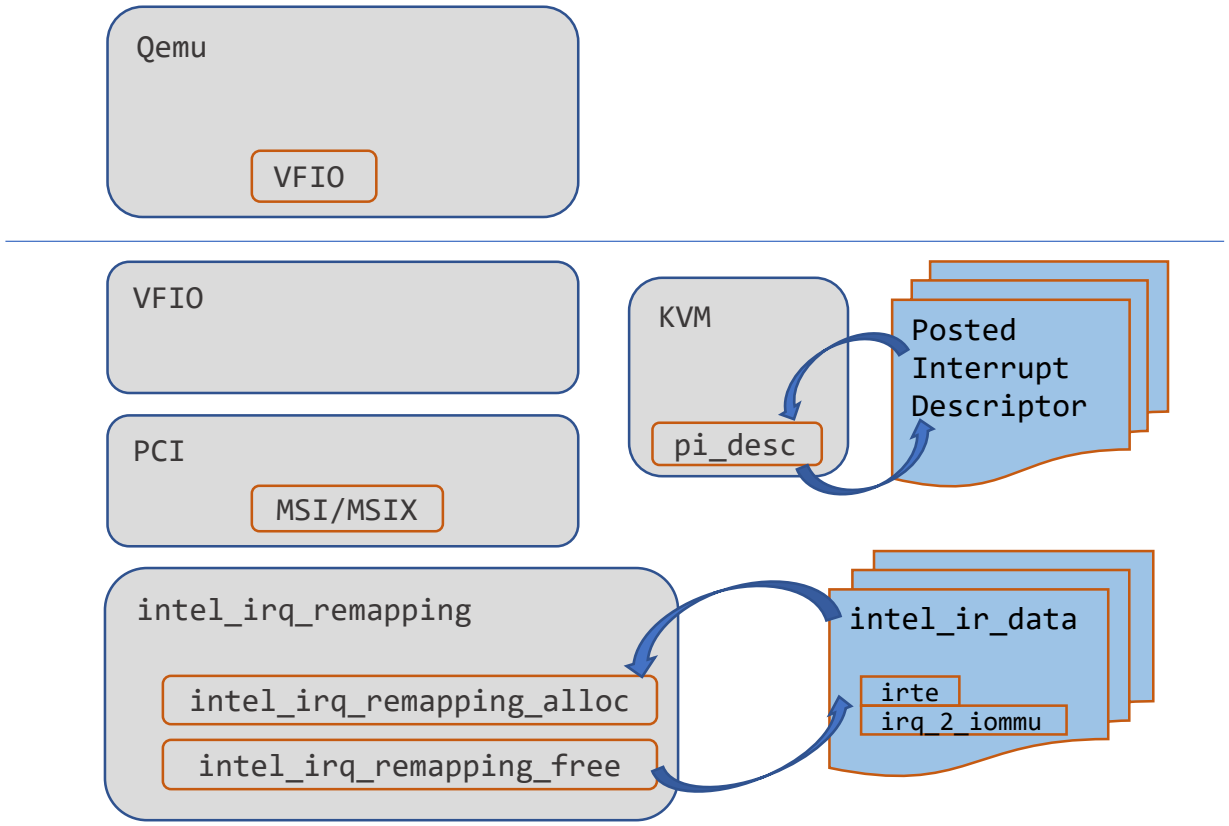
- Optimized snapshot - “savevm-in-pmem”
  - Keep VM memory in DAX, instead of saving to disk
  - Non-memory state could be either in disk or memory (except storage)
- “loadvm-in-pmem” when relaunching Qemu
  - Remap guest memory from DAX filesystem
- Pass DRAM-as-PMEM reservation info through kexec
  - Same “memmap=8G!4G” boot parameter between old/new kernels

Passthrough Device Support

- Keep DMA/Interrupt alive for passthrough devices
- Avoid clobbering hardware state during reboot
- Keep upstream devices (IOMMU, switch/root ports, etc.) alive
- Reconnect on resume various hardware resources with VM & passthrough devices

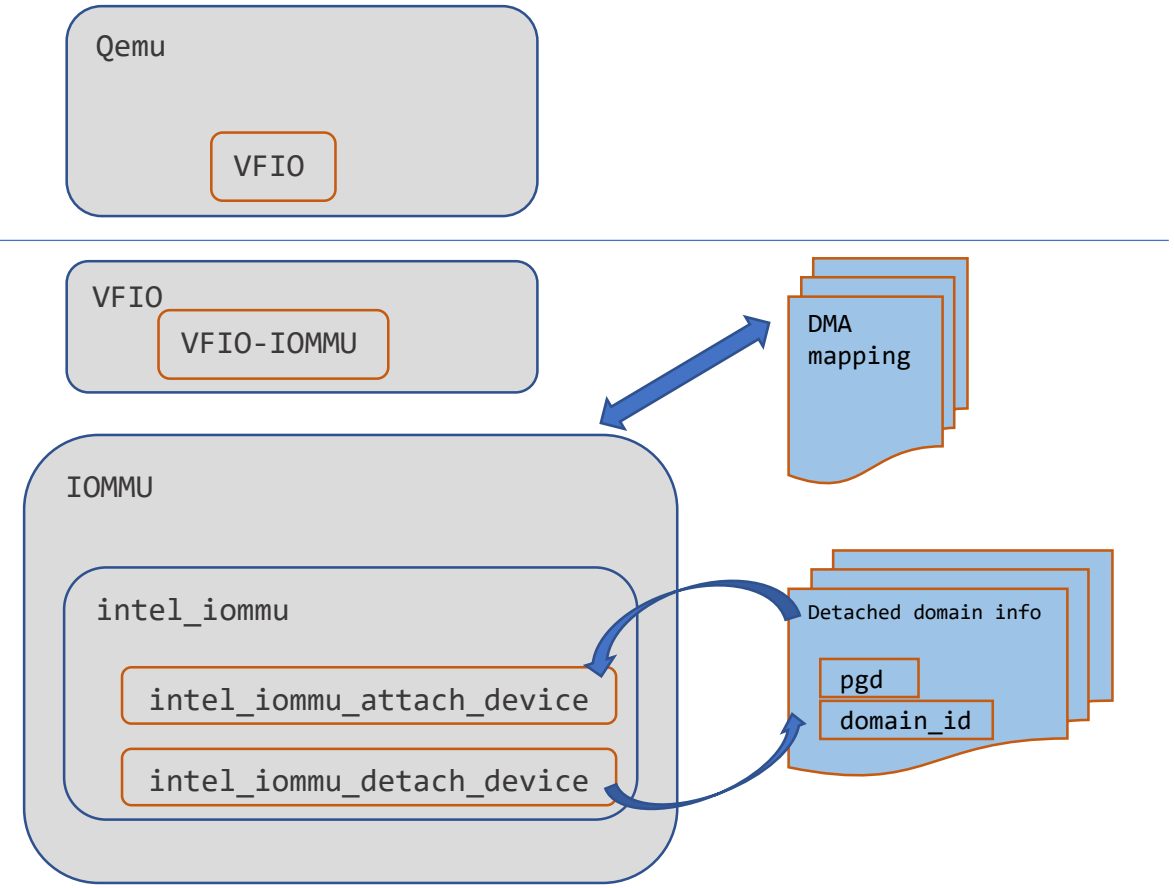
Keep Interrupt Alive

- Leverage Posted-Interrupt (PI)
- Pass PI-Desc & IR-Table to new kernel
- Hardware continues using old PI-Desc & IR-Table
  - Pending guest IRQ is recorded in PI-Desc
- New KVM/IOMMU allocates new PI-Desc & IR-table
  - Update the content based on old structure



Keep DMA Alive

- Pass IOMMU configuration to new kernel
- IOMMU HW continues using old page table in reboot
  - DMA on-going during reboot
- IOMMU driver re-allocates page table post reboot
  - Copy the mapping from old table



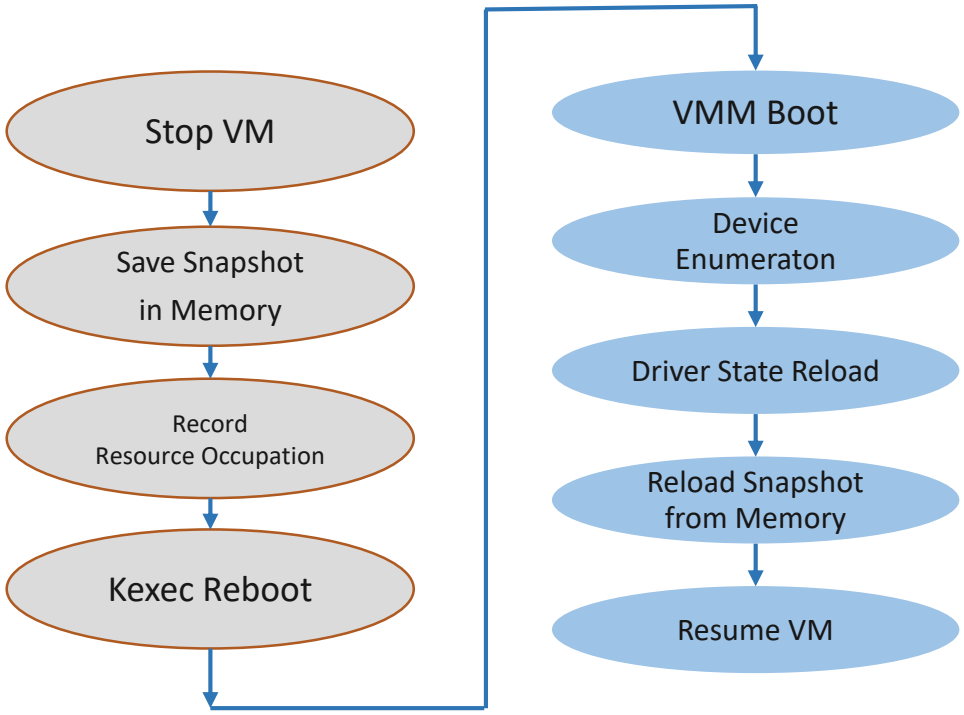
Kexec Reboot Processing

- Skip shutdown of passthru device & its upstream devices before reboot
- Data passing from old kernel to new kernel
  - Passthru device list & memory reservation
  - May need to extend boot protocol
- Special handling in device enumeration & driver matching
  - Avoid attaching of native driver for passthru devices
- Host drivers pick up hardware state (IOMMU & upstream devices)
  - Avoid clobbering hardware state

New VFIO Device Assignment Model

- VFIO driver probe/open skip reset & PCI initialization
- VFIO snapshot saving/reloading
  - Record VFIO internal states, including MSI/MSIX state
- Complete re-assignment in 2 steps
  - vfio_realize() reattach group/domain etc., w/o hardware clobbering
  - Reload VFIO snapshot to re-connect DMA/IRQ state
- New ioctl cmd/flags to indicate re-assignment
  - Or VFIO handles automatically by recognizing in passthru device list

High Level Flow



- Save Snapshot in Memory (in old Linux) & VM Quit
  - DAX filesystem in DRAM-as-PMEM
  - Don’t free HW resources (IRTE, etc.)
- Record Resource Occupation
  - Device list, memory, etc.
- KEXEC Reboot
  - No hardware clobber in driver shutdown
- VMM Boot (new Linux)
  - Reserve resources
- Device Enumeration
  - No hardware clobber in PCI enumeration
  - No native driver attaching
- Driver State Reload
  - IOMMU driver reload state
- Reload Snapshot from Memory (in new Linux)
  - Re-enable DAX filesystem in DRAM-as-PMEM
- Resume VM
  - Reload DMA mapping
  - Re-enable MSI/MSIX

Opens

- How to handle trusted boot?
- Handling memory resizing with DRAM-as-PMEM?

Status & Plan

- Milestones:
  - VMM fast restart w/o passthru devices
    - Done
  - Qemu fast restart w/ passthru devices, but w/o host reboot
    - Done
  - VMM fast restart w/ passthru devices, w/ host reboot
    - On-going
- Next Step: Upstreaming

Questions? Thanks!