August 07, 2019

Live migration of cloud clients that have PCIe devices

Benjamin Serebrin

Grigory Makarevich

Eric Northup

Live migration of cloud clients that have PCIe devices

ABSTRACT

Cloud computing service providers use live migration to transparently move guests from one host to another. During live migration, guest access to certain memory regions is temporarily restricted. Such access restrictions are possible with CPUs; however, PCIe devices, e.g., GPUs or other accelerators, typically do not provide the facility to restrict memory access even temporarily. Thus, at present, guests that use PCIe devices cannot be live-migrated without device-specific mechanisms. This disclosure describes techniques to perform live migration for guests with PCIe devices by using hardware transactional memory to perform an atomic transfer of a range of guest memory from a source host to a destination host. PCIe-originated traffic targeted at a memory page under transition between hosts is not paused.

KEYWORDS

- Cloud computing
- Virtual machine
- Live migration
- Peripheral connect interface (PCI)
- PCI-express (PCIe)
- Hardware transactional memory
- Atomic memory transfer

BACKGROUND

Cloud computing service providers use live migration to transparently move clients (guests) from one host to another. Doing so relies on the ability to temporarily restrict guest access to certain memory regions as the guest is being moved from the original host to the new host. For example, for CPU-only guests, the hypervisor is able to move without inconsistency pages that have been touched during the move (by using dirty page tracking). Similarly, blocking access to certain pages on the new host enables guests to be restarted during transfer of the guest memory. For example, if the guest accesses a page of memory that has not moved yet, the guest is paused until the desired page is ready. This pausing relies on a property of CPUs: any page of memory can be as marked not-present or not-writeable, and any access to such a page is paused while the host performs the appropriate maintenance to make the page available. The host restarts the paused access which resumes cleanly.

However, assigned PCIe devices such as GPUs and other accelerators lack clean, general purpose techniques to pause and resume operation. Thus, at present, guests that use PCIe devices cannot be live-migrated.

DESCRIPTION

Per the techniques of this disclosure, guests with PCIe devices are live-migrated using hardware transactional memory to perform an atomic transfer of a range of memory from a source host to a destination host. The atomic transfer includes copying the full range of guest memory and updating mappings in the IOMMUs (input-output memory management units) of the hosts to ensure that there are no undetected orderings of events that cause the original and copied pages to become inconsistent.

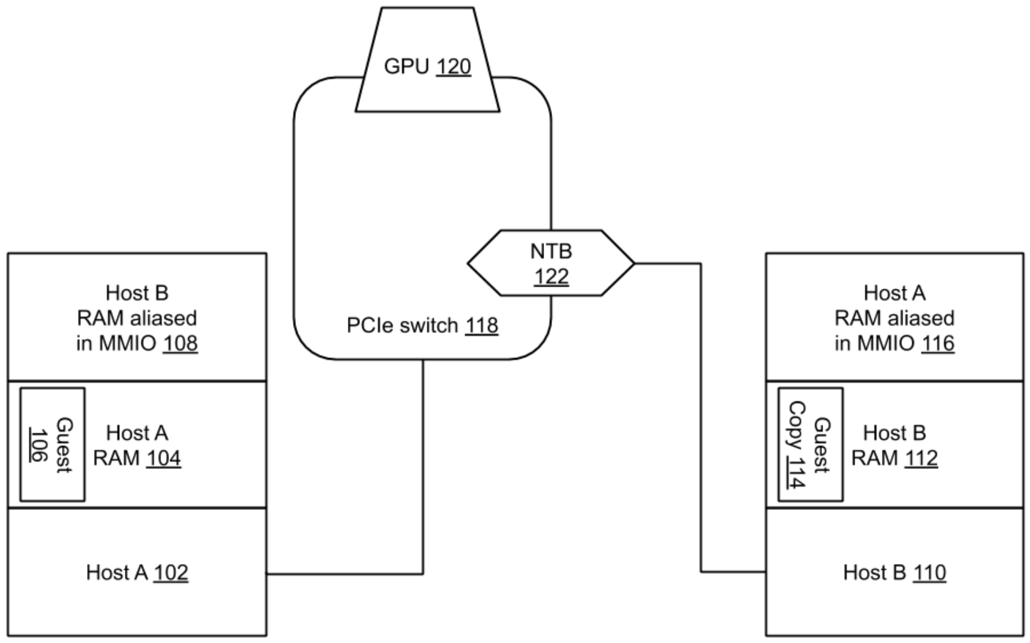

Fig. 1: Live migration of cloud clients that have PCIe devices

Fig. 1 illustrates live migration of cloud clients that have PCIe devices, per techniques of this disclosure. Host A (102) is a cloud-based machine with RAM (104), a section of which is occupied by a guest (106) of host A. Similarly, Host B (110) is a cloud-based machine with RAM (112), a section of which may be occupied by guests of host B (114). Host A is coupled to a PCIe switch (118), which provides access to PCIe devices, e.g., GPU 120. Both hosts A and B are connected to a common PCIe chassis. Some of the RAM and devices of host B are aliased via MMIO (memory-mapped input-output, or memory-mapped devices) (108) such that these are accessible to host A. Similarly, some of the RAM and devices of host A are aliased via MMIO (116) such that these are accessible to host B. A non-transparent bridge (NTB) (122) links the two hosts. Each NTB endpoint has one or more apertures such that a write to an aperture at one end is mirrored to memory (including memory-mapped devices) on the other end. A live migration of a guest is sought from host A to host B.

Devices on a PCIe typically form a small, local network of acyclic topology, e.g., tree. The root of the tree is the CPU complex. The disjoint PCIe trees of the two hosts are connected at the NTB. While this disclosure refers to use of NTB, multi-root IOV (MR-IOV) and advanced PCIe switch fabrics, that exhibit similar behavior, can also be used.

The IOMMU (input-output memory management unit) is located between the PCIe hierarchy and the memory hierarchy of the CPU and performs address translation for devices. The host sets up the IOMMU to map the virtual memory addresses of its guest to physical memory addresses on the RAM of the host.

Per techniques of this disclosure, the PCIe hierarchy is not migrated along with the guest. Rather, with both hosts being connected to one PCIe system, the guest is moved from the source host to the destination host while retaining the same PCIe hierarchy. Memory translations in the NTB and the IOMMU are adjusted such that memory access requests that went to the source host go to the destination host instead.

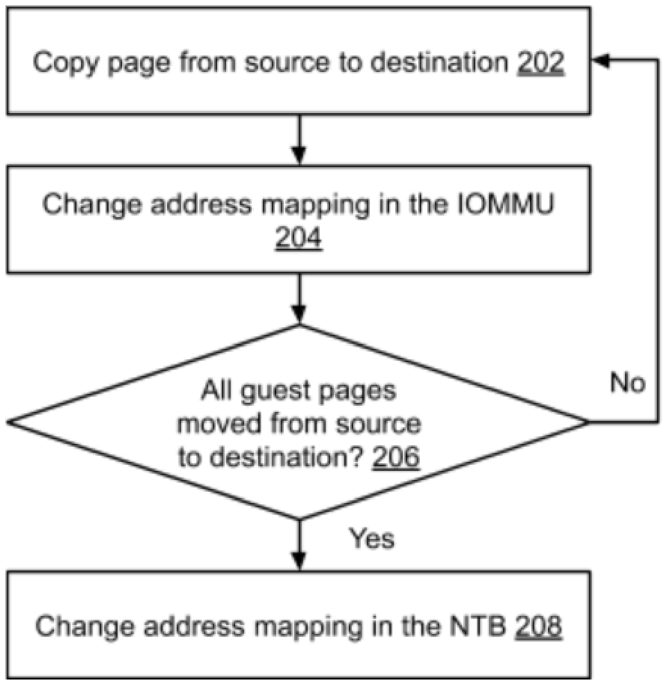
Copy page from source to destination 202
Change address mapping in the IOMMU 204
All guest pages moved from source to destination? 206
No
Yes
Change address mapping in the NTB 208

Fig. 2: Moving pages from source host to destination host

Fig. 2 illustrates the moving of pages from a source host to a destination host to achieve live migration of guests. A page of memory is typically 4096 bytes and can include memory-mapped devices. A page of memory is copied from source to destination (202). The address mapping in the IOMMU is changed to point to the new page location in the destination host (204). By virtue of the NTB, the new location is physically on a different (destination) host but is addressable by the source host. When all guest pages are moved from source to destination (206), a one-time address translation is performed inside the NTB (208) such that PCIe traffic is sent directly to the destination host, rather than bouncing off the source host before reaching the destination. In this manner, a move from source to destination host is effected such that PCIe devices that, unlike CPUs, cannot handle page faults, are granted memory access through and beyond the duration of live migration.

Transitioning of a memory page from source to destination is atomic, e.g., a read or write access to the page during transition either results fully in the old page (prior to it being touched by a PCIe device) or results fully in the new page (after it has been updated by the PCIe). Atomic transitioning of memory pages is achieved using principles of hardware transactional memory as follows. A first CPU or device that performs a transaction comprising a read operation on a set of data items caches such data items in a read set. The first device or CPU enters, executes, and exits the transaction. Upon exit, it checks if any other device or CPU has written cache lines read by the first device or CPU. If another device or CPU did write the same cache lines, then the first device or CPU aborts its transaction, e.g., rejects the data items read. If no other device or CPU wrote on the same cache lines during the transaction, the first device or CPU commits the transaction.

Similarly, a first CPU or device that performs a transaction comprising a write operation on a set of data items caches such data items in a write set. The first device or CPU enters, executes, and exits the transaction. Upon exit, it checks if any other device or CPU has read or written cache lines read by the first device or CPU. If another device or CPU did read or write the same cache lines, then the first device or CPU aborts its transaction, e.g., leaves the data items unchanged. If no other device or CPU read or wrote on the same cache lines, the first device or CPU commits the transaction.

As explained above, a transaction that occurs during a page move modifies an IOMMU page table entry. If the IOMMU entry isn't cached in the TLB (translation lookaside buffer) and an access has not occurred to the corresponding entry in the page table, e.g., by another device or CPU, then the physical memory that is the target of the IOMMU page table entry hasn't been accessed. In this manner, the techniques of this disclosure detect that a PCIe device has touched a page that was transitioning between hosts, and if detected, the copy of the page being transitioned is deleted.

	CPU0	**CPU1**	**IOMMU**

*t*1	`Begin`
	`// Fill the read set -- this is needed only if the VCPUs are still able to access the page`
	`for (i = 0; i < 4096; i+=CACHELINE_SIZE)`
	`  Dummy += SOURCE_VA[i]`
	`PTE = FINAL_PTE`
	`Send signal 1`
		`Wait on signal-1`	`TLB may contain IOVA->SOURCE_PA mapping`

*t*2	`Wait on signal-2`	`Invalidate IOMMU TLB for IOVA`
		
		`memcpy(DESTINATION_A_VA, SOURCE_VA, 4096)`
		
		`Send signal-2`	

*t*3	`End`		

Table 1: Timeline for transitioning a page between hosts

Table 1 illustrates a timeline, including pseudo-code to transition a page between hosts. In this table, `Begin` and `End` signify respectively the beginning and the end of a transaction on a CPU thread. `IOVA` is the PCIe address the GPU uses to talk to the memory page under transition. Its numeric value does not change. The page under transition has the following aliases: `SOURCE` is a source host RAM address (`SOURCE_VA` and `SOURCE_PA` are source-side addresses, where `_VA` indicates virtual address used by CPUs and `_PA` indicates physical address in RAM); `DESTINATION` is a RAM address in the destination host that aliases to the same page, and `DESTINATION_A_{VA,PA}` are the Host A-side virtual and physical address aliases of that page; `FINAL_PTE` is Host A-side page table entry value that encodes the A-side physical address of host B's RAM page, `DESTINATION_A`.

In a first time slot *t*1, the source host (CPU0) edits the IOMMU page table entry. The IOMMU page table entry that used to point to a certain physical memory location in the source host is made to point to a new physical memory location in the destination host. Also, the memory page, which in this example has a size of 4096 bytes, is copied between host physical address `SOURCE` to destination address `FINAL_PTE`. The page table entry (PTE), which maps from `IOVA` to `SOURCE` is set to `FINAL_PTE`. This change is not visible to the IOMMU, PCIe hierarchy, or other CPU because the transaction is not yet committed.

In the meantime, the destination host (CPU1) is waiting on a signal-1 from the source host. The receipt of signal-1 by the destination host signifies the start of time-slot *t*2, during which the destination host invalidates the IOMMU TLB for `IOVA`, and performs a memory copy from `SOURCE` to `FINAL_PTE`. This memory copy is done over the NTB. The destination host sends a signal-2 to the source host, which signifies the end of time slot *t*2. The source host commits the transaction using an `Xend` command. Upon committing the transaction, the page table entry becomes visible to all devices and CPUs.

If the page table is walked between `Xbegin` and `Xend`, e.g., a device accessed the memory page under transition, the transaction is aborted, and the memory page transition of table 1 is attempted again. If the transaction is not aborted, then the page update has occurred without external modification, and the IOMMU page table entry update has committed without external observation.

The page moving process can accommodate larger page sizes, at the risk of increased rate of retries due to device contention. This per-page moving process is repeated for all pages within the guest's RAM. IOMMU mappings on host B are configured to have a parallel mapping from IOVA to guest RAM locations as the pages are moved. When all pages are moved, the NTB or PCIe hierarchy settings are updated so all traffic flows directly to host B.

Signals during a transaction

Some definitions of hardware transactional memory disallow traditional signaling into and out of a transaction; typically all writes within a transaction can be either invisible to the system or simultaneously visible, which limits the ability of the transaction to signal out to a helper thread. Similarly, spinning on a lock variable is not possible because the first transaction read brings the lock variable into the read set, and the helper thread's clearing of the lock variable then aborts the transaction. This section describes how signals can be built within the restrictions of transactional memory.

Before the start of a transaction, the source and destination hosts agree on a schedule that is based on a common (synchronized) clock. For signals into a transaction, e.g., signal-2 of Table 1, the transaction involves the reading by one CPU of a series of status fields that the other CPU writes. The reading CPU consumes, for example, one status field every 10,000 ticks of a high-precision counter (time-stamp counter, or TSC), and the writing CPU asserts the signal by writing the next status field to be read, but does so before the reader CPU will read the field. If, in the course of reading status fields, a “done” indicator is found, then the reader determines that the signal was asserted. If all fields are consumed without finding a “done” indicator, then the reader determines that the signal was never asserted within the specified time frame and aborts the transaction. Thus, when CPU1 is done, it examines the current TSC and writes the next status field that will be read by CPU0, e.g., a status field that is not yet in the read set so that writing it won’t abort the transaction.

For signals out of a transaction, e.g., signal-1 of Table 1, the pre-arranged schedule is such that a given phase of the transaction is to be done by a certain time. If the phase isn’t completed by then, the transaction is aborted. This works in cases where the non-transaction phase comprises idempotent operations, e.g., operations whose effect is the same no matter how many times they are repeated (so long as the operations occur prior to a commit). In this case, CPU1’s invalidation and copy are harmless if done spuriously; CPU1 consumes signal-1 by waiting until the prearranged time has passed.

CONCLUSION

This disclosure describes techniques to perform live migration for guests with PCIe devices by using hardware transactional memory to perform an atomic transfer of a range of guest memory from a source host to a destination host. PCIe-origin traffic targeted at a memory page under transition between hosts is not paused.