Multi-dirty-vram : Support for Multiple Frame Buffers in Xen

Robert S. Phillips, Virtual Computer Inc. 2012/10/05

This document describes changes provided in the "multi-dirty-vram" patch to support multiple frame buffers in Xen. Support is provided for both shadow and hardware assisted paging (HAP) modes. For convenience this code is referred herein as "DV".

The purpose of this code is to bookkeep the set of video frame buffers (*vram*), to detect when the guest has modified any of those buffers and, upon request, return a bitmap of the modified pages. This lets other software components re-paint the portions of the monitor (or monitors) that have changed. Each monitor has a frame buffer of some size at some position in guest physical memory. The set of frame buffers being tracked can change over time as monitors are plugged and unplugged.

Code Refactoring

The code related to DV has been (as much as possible) refactored into xen/arch/x86/hvm/dirty_vram.[hc].

Data Structure Changes

The DV data structures are protected by the paging lock.

Each HVM domain points to a single *struct dv_dirty_vram*, which contains all the other related data structures:

- the head of a linked list of 'ranges'. Each range is a *struct dv_dirty_vram_range* . It contains the data related to one frame buffer.
- a count of the current number or ranges and a high-water mark for the maximum number of ranges encountered since domain creation.
- the head of a list of pages containing objects of type *struct dv_paddr_link* , described below, and a linked list of available dv_paddr_link objects.

Each range is represented by a *struct dv_dirty_vram_range* . It contains the data related to one frame buffer:

- *pl_tab ,* an array of *struct dv_paddr_link* objects. This array's size is the number of pages in the range. Each entry bookkeeps the Level 1 page table entry that maps the corresponding guest page. This is described in more detail below. Pl_tab is only used in shadow mode. It is not allocated in HAP mode.

- *nr_mappings* is the total number of mappings being managed by *pl_tab* . See below. *mappings_hwm* is the high-water mark for the maximum number of mappings created since range creation.
- *dirty_count* is a count of the number of pages in the range that have been dirtied since the last time a bit map of the dirty pages was requested by user code. This count is only maintained in HAP mode.

Various members of dv_dirty_vram_range are used (or not) depending on how page dirtiness is tracked. There are three mechanisms: Shadow, HAP and VRAM-SW-Dirty.

Shadow Mode

In shadow mode DV detects that a vram page has been dirtied by examining the level 1 page table entry (L1PTE) that maps the vram page. When memory is modified, the hardware sets the 'dirty' flag in the L1PTE.

DV bookkeeps all the L1PTEs that map frame buffer pages. When the user code requests a bitmap of dirty pages, DV scans all those L1PTEs. It looks for entries with the dirty bit set, clears the dirty bit and notes the dirty frame buffer page in the user-provided dirty bitmap.

The user code requests a dirty bitmap for a particular range by providing the frame buffer's starting guest physical page number and its length (in pages). DV scans its list of ranges looking for a match. The matching range's pl_tab contains an entry for each page in the range. Each entry points to a L1PTE. QED.

However a frame buffer page might be mapped by several L1PTEs. (For example, each guest application that can access the frame buffer will access it using its own virtual address, and so have its own mapping.) To accommodate this, each pl_tab entry can point to other pl_tab entries that map the same frame buffer page. In other words, the pl_tab entry for any frame buffer page is a dv_paddr_link object and it can be the head of a singly-linked list of such objects.

Conversely a frame buffer page might not be mapped. In that case its pl_tab entry contains an invalid value rather than a pointer to a L1PTE. Unmapped pages are deemed to be clean.

Pl_tab entries are maintained from three different points in the code.

(1) The entries for each range are initialized by scanning all L1PTEs in the domain, looking for those that map pages in the range.

(2) As shadow_set_l1() modifies L1PTEs, DV looks for the addition and removal of entries that map pages in the range.

(3) As shadow pages are deleted by delete_shadow_status() and delete_fl1_shadow_status(), entries are removed from the range.

As entries are added and removed from pl_tab, the range bookkeeps the total number of mappings and the high-water mark. If a range has all its entries removed, the range itself is deleted. This is how ranges are cleaned up. A range might fall into disuse but, sooner or later, all the shadow pages that it points to are re-claimed, at which time the range is deleted.

HAP Mode

With hardware assisted paging, the mapping from a guest physical page to a machine physical page is provided by a second level of page tables. For Intel, these are called extended page tables (EPT). For AMD, they are called nested page tables (NPT). For convenience, the following discussion assumes the Intel/EPT perspective but the concepts are the same for AMD/NPT.

(The first [traditional] level of pages tables map guest linear addresses to guest physical addresses.)

In Xen, the EPTs are provided by the P2M table. These are directly accessed by and cached in the hardware. These tables provide not only the mapping from each guest physical frame number (gfn) to a machine frame number (mfn) but also the access flags. In particular, a L1 EPT entry can mark a machine frame as read-only.

DV uses this facility to mark all frame buffer pages as “logdirty” read-only. The first time the guest tries to modify a frame buffer page, the hypervisor gets an EPT page fault. The handler recognizes this as a “logdirty” page and simply marks the page as writable. Upon retry, the write succeeds.

Later when the user code requests a dirty bitmap, DV examines all the frame buffer’s L1 EPT entries. Any that have been marked as writable indicate a dirty page. The page is re-marked as read-only and DV notes the dirty frame buffer page in the user-provided dirty bitmap.

The original code, which DV replaces, also maintained a dirty bitmap for the domain. Any EPT page fault would set the appropriate bit in addition to marking the page as read-write. The code that delivered the dirty bitmap to the user would populate it based on this domain dirty bitmap. DV still maintains the domain bitmap but does not use it to populate the user dirty bitmap. (The eliminated code was complex, buggy and unnecessary.)

The EPT caches the P2M mappings and access flags. When DV modifies an access flag in the P2M table, it must notify the EPT hardware that its cached data is out-of-date. This is the equivalent of a TLB shootdown but directed at the EPT tables rather than the regular page tables. Such shootdowns are bound to have a negative effect on performance, just as TLB shootdowns do.

Affected Functions

The following functions were added, modified or deleted for DV.

Function	Change
hvm_hap_nested_page_fault	Combined calls to mark page as dirty and to change its p2m type into a single call which performs both functions atomically. The

	previous code left a unlock window between the two calls where a page might be modified without being marked as dirty.
hap_enable_vram_tracking	Enhanced to handle multiple ranges, marking all vram pages in each range as logdirty read-only.
hap_disable_vram_tracking	Enhanced to handle multiple ranges, marking all vram pages in each range as read-write.
hap_clean_vram_tracking, hap_clean_vram_tracking_range	Disabled hap_clean_vram_tracking since it does not support multiple ranges. Replaced it with hap_clean_vram_tracking_range() which is now called directly from paging_lock_dirty_range(), whereas hap_clean_vram_tracking was called indirectly via d->arch.paging.log_dirty.clean_dirty_bitmap(d).
hap_track_dirty_vram	Creates the domain's dv_dirty_vram on demand. This is unchanged but now it does so by calling dirty_vram_alloc rather than xmalloc'ing the structure explicitly. Creates ranges on demand when some [begin_pfn:nr] is first encountered. To collect the dirty bitmask it calls paging_log_dirty_range(). Copying the dirty bitmask into guest storage is now done in this function. Code that explicitly deleted the domain's dv_dirty_vram now calls dirty_vram_free() instead.
paging_mark_dirty_hap	This is a new function. It both changes the faulting page's type from logdirty to read-write and marks the page as dirty. Both actions are performed atomically under the paging_lock, thus eliminating an erroneous unlock window that previously existed. It also records in the range's dv_dirty_vram_range that pages have been dirtied.
paging_log_dirty_range	This function has been dramatically changed. Its purpose is to return a bitmap of the dirtied pages for some range, and to re-mark all the dirtied pages as read-only. It used to call hap_clean_vram_tracking to re-mark the pages, but that function did not support multiple ranges. It then consulted the domain's dirty bitmap (which is rooted at d->arch.paging.log_dirty.top) to compute the bitmap of dirtied pages. As bits were copied into the guest dirty bitmap, they were cleared from the dirty bitmap. That code was very complex and buggy: it cleared entires pages of bits rather than just the bits that were copied. The new code does not consult the domain's dirty bitmap. It simply looks for the pages that were marked as read-write and concludes that they were dirtied. In other words it combines the re-mark operation with the dirty bitmap creation operation. It also has a quick exit which it takes if no pages in the range have been marked as dirty.
shadow_teardown	This now calls dirty_vram_free() to tear down a domain's dirty_vram structure, rather than freeing the components explicitly.
dirty_vram_alloc,	These functions encapsulate the management of the dirty_vram

dirty_vram_find_or_alloc, dirty_vram_free	structure.
dirty_vram_range_find, dirty_vram_range_find_gfn, dirty_vram_range_free, dirty_vram_range_alloc, _ dirty_vram_range_alloc, dirty_vram_range_find_or_alloc	These functions encapsulate the management of the range structure. dirty_vram_range_alloc() ensures that the new range does not overlap any existing ranges – deleting them if necessary – and then calls _dirty_vram_range_alloc to actually allocate the new range.
alloc_paddr_link, free_paddr_link	These functions encapsulate the management of the dv_paddr_link linked-list structure.
dirty_vram_delete_shadow	This is called when a page ceases to be a shadow page. The function finds and removes any paddr_links that refer to the erstwhile shadow page.
dirty_vram_range_update	This is called whenever a level 1 page table entry is modified. If the L1PTE is being cleared, the function removes any paddr_links that refer to it. If the L1PTE is being set to a frame buffer page, a paddr_link is created for that page's entry in pl_tab.
dv_find_all_vram_mappings, hash_pfn_foreach, dv_find_vram_mappings_in_l1	dv_find_all_vram_mappings is called when a new range is created to locate all the L1PTEs that map frame buffer pages in the range. The other two functions are utility functions that it calls.
shadow_scan_dirty_flags	This produces a dirty bitmap for the range by examining every L1PTE referenced by some dv_paddr_link in the range's pl_tab table. It tests and clears each such L1PTE's dirty flag.
shadow_track_dirty_vram	Creates the domain's dv_dirty_vram on demand. It creates ranges on demand when some [begin_pfn:nr] is first encountered. To collect the dirty bitmask it calls either shadow_scan_dirty_flags(). Copying the dirty bitmask into guest storage is done in this function.
delete_fl1_shadow_status, delete_shadow_status	These functions now call dirty_vram_delete_shadow. See above.
_sh_propagate	This function no longer bookkeeps dirty_vram->last_dirty. No such data member no longer exists.
shadow_vram_fix_l1e	This new function is called by (and contains code factored from) shadow_set_l1e(). It calls dirty_vram_range_update() which updates the dirty_vram structures as L1PTEs are modified so they start to (or cease to) point to frame buffer pages.

[end of document]