Xen VGA passthrough

Author: Yuri Schaeffer BSc, yuri.schaeffer@os3.nl
System & Network Engineering, Univeriteit van Amsterdam
Supervisor: Mendel Mobach, Systemhouse Mobach bv

Abstract

After a small introduction of PCI passthrough and Xen we will discuss VGA passthrough. We will try to cover the most important steps needed to successfully implement the passthrough and analyse why these steps are needed. Afterwards we will reflect on the project and look into the future of VGA passthrough.

1 Introduction

Virtualization of hardware is hardly anything new in computer science. Decades ago large mainframes already ran multiple operating systems simultaneously. Not only full (software) virtualization but also hardware aided virtualization and paravirtualization. On consumer graded products (e.g. i386 compatible hardware) such hardware support was not available up until recent years and virtualization was always implemented in software. This implies a significant performance penalty.

Nowadays, mainstream hardware has support for virtualization and notably paravirtualization. The introduction of technologies as VT-x and more specific VT-d (*Virtualization Technology for Directed I/O*) enables PCI passthrough. The open source virtualization software Xen can use this functionality to assign a specific piece of PCI(e) hardware exclusively to a domU machine [4]. The hypervisor maps memory and interrupts directly to the domU machine. Other machines, including dom0, do not have any access to the device anymore. Since the machine communicates directly with the hardware it is able to use the native drivers and its full capabilities.

1.1 VGA passthrough

PCI passthrough is mainly used for devices such as network interfaces and USB ports but all modern hardware on the PCI bus should be able to support this feature. One of the exceptions to this are graphics controllers. Graphics controllers have some special inherited legacy from the past. For example not all memory is allocated dynamically and the card carries its own BIOS that needs to be (re-)executed.

Although hard to implement, VGA passthrough could be very useful in certain cases. Imagine a user working in an unprivileged domain, possibly unaware of any virtualization going on. This user might be able to do anything he could do on a normal computer including using full capabilities of his graphics card. Underneath, domain 0 has control over the virtualized OS. Being able to guard the OS against the big bad world.

2 Xen architecture

Prior to analysing what is needed to implement VGA passthrough we need to understand Xen's internal architecture. This chapter will show us how Xen is logically organised, how memory is allocated and the basics of PCI passthrough.

2.1 Structure

Xen consists of two main parts: the hypervisor and a management module. Xen's hypervisor is the first thing that boots but it does not have any direct interface to the user. In fact, the hypervisor is a very limited piece of software, it has for example no notion of devices or disks. All it does is scheduling of processes and memory management.

The first action that the hypervisor takes (visible for the user) is bootstrapping a privileged domain 0, dom0 for short. Dom0 usually is a Linux distribution with a modified kernel to support Xen. This operating system has the privilege to directly address hardware with their native drivers. On top of dom0 runs a management program, DM&C, to control Xen and administer unprivileged virtual machines (domU).

When using paravirtualization (PV), the OS is aware it is virtualized and may not address hardware directly. Instead, it uses special drivers without any hardware capability. These *front-end drivers* communicate with a *back-end driver* which sits on top of the native driver in dom0, see figure 2.1. The communication takes place in a shared memory space for dom0 and the domU, when data is set ready the virtual machine can invoke an interrupt for the other machine to pick up the data. An important advantage for this method is that no hardware needs to be emulated which reduces the performance hit.

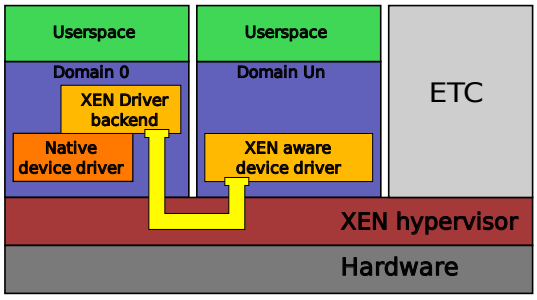

Fig. 1: A PV domain is aware of virtualization and makes no attempt to directly control hardware.

Another option is to run a virtual machine as *hardware virtual machine* (HVM). This time the guest OS is not aware of any virtualization and will run as it normally would do, including 'real' device drivers. The hardware seen by the HVM guest OS however is not related to any physical hardware present but is completely emulated. In this case the dom0 machine will run a daemon, *qemu-dm*, that performs this emulation and facilitates translation to real devices. The QEMU project is an existing processor emulator which is patched for use in Xen. The fully virtualized OS thinks to control some well supported generic hardware such as a Cirrus Logic video card and a Realtek network interface.

2.2 Memory layout and passthrough

To understand what is happening with PCI passthrough we need a grasp at how the machines memory is build up when running Xen. Figure 2 shows a rough sketch of the memory of a computer running Xen. Besides the BIOS, there is a range reserved for communication with devices. This area is used for example by network interfaces or disk controllers.

Then there is Xen and the virtualized machines, each virtual machine is layout the same way as any normal computer would be. The VM it self knows no better than its memory starts at address 0, it is Xen and the CPU that facilitate the translation to physical addresses. Similarly, the VM has a BIOS available. Because of its static nature this is just a one on one copy of the original BIOS.

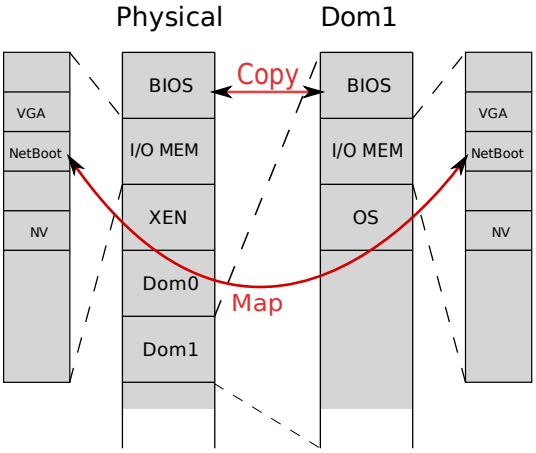

Fig. 2: Schematic overview of possible memory usage while running Xen.

When applying PCI passthrough, certain memory areas of the physical machine are mapped to the VM. Figure 2 shows this with NetBoot. The memory area in the VM occupied by netboot is actually living outside the VM's memory range. When the guest OS writes to one of those memory addresses, Xen will make sure it is actually written at the appropriate address. This implicates that no other VM is able to make use of that device.

3 VGA passthrough

Although VGA passthrough is far from widely available, at least one attempt has been made to achieve native VGA in an unprivileged domain. In May of 2008 Jean Guyader presented a patch for Xen to allow the passthrough [2]. Although his code is not very generic it serves well as a proof of the concept. This chapter we will analyse the steps the above patch takes, try to grasp the meaning and comment on the why and how of each part.

According to Guyader's patch notes four important steps are made:

1. Map VGA framebuffer to guest
2. Copy VGA BIOS to guest
3. Map VGA I/O ports
4. Disable Xen's VGA code

3.1 Map VGA framebuffer to guest

A standard VGA adapter can have memory up to 256 KiB. 128 KiB of this memory is used in normal VGA/EGA, monochrome and CGA display modes (In the old days multiple modes could be used simultaneously when having more than one adapter) [3]. This memory is called the framebuffer and is normally mapped to the machines main memory at address 0xA0000 (i.e. at 640 KiB, well above what anyone will ever need) to 0xC0000. This memory range must be mapped to the VM's memory in order for the OS to address the video adapter.

PC's have a mechanism to report which memory locations are in use by the BIOS and which are free for the OS to claim, dubbed e820 (The hexadecimal value a register needs to be set in order to get this information). The VGA memory and VGA BIOS are not reported by e820 but are at a fixed location in memory, thus every OS has the locations hard coded.

This has two implications. First the VGA memory can not be located inside the physical VM's memory. When the VM's OS tries to read or write to that specific memory range, the logical addresses should *not* be translated by Xen to physical addresses in the VM's memory space. This requires an extra check specially for the VGA address range. Second, since the VGA adapter does not specify it's address range dynamically Xen can not use it's normal passthrough mechanism. The VGA addresses must always be hard coded.

Most of this work is done in the *tools/libxc/xc_hvm_build.c* and *tools/libxc/xc_linux.c* files. The VGA BIOS is copied to the guests memory (see section 3.2) and there's a mapping to the VGA memory.

3.2 Copy VGA BIOS to guest

As with the framebuffer memory the VGA BIOS can be found at a fixed range, 0xC0000 – 0xCAFFF. The BIOS provides a set of functions to control the VGA adapter, it must be copied to the memory space of the guest and then executed.

The patch introduces a check in *tools/firmware/hvmloader.c* on vendor and device ID of the graphics card to decide which type of adapter is present and copy range of memory for the appropriate BIOS. In order to accommodate this, the memory location of etherboot is shifted with 8 KiB. Also, Xen seemed to use the memory range of the VGA BIOS as a temporary space for BIOS related things, this area is now relocated elsewhere. The files responsible for this change is *tools/firmware/config.h*.

At last the BIOS instructions need to be executed to initialise the graphics adapter. This takes place in the *tools/ioemu/hw/pc.c* file.

3.3 Map VGA I/O ports

To perform communication between software and the VGA adapter memory mapped i/o is used. The adapter virtually occupies a part of the machines memory. To pass a message to the adapter one would simply write a value to the appropriate memory address. The adapter would pick the address and the value up from the memory bus and behave accordingly.

Since the hardware is listening to a specific memory address Xen must facilitate a mapping from the logical to the physical range. Or rather make an exception not to translate operations to that memory area.

The file *tools/libxc/xc_hvm_build.c* is modified by the patch to map the memory range from address 0x3C0 to 0x3E0 to the guest OS.

3.4 Disable Xen's VGA code

If the video is passed through to any unprivileged domain, Xen must prevent dom0 to grab and use the VGA adapter. The patch notes imply to modify a *hvm.c* to disable the VGA code, this is however not implemented by the patch. A little research did however reveal a source file *xen/arch/x86/hvm/hvm.c* that included a call to the a video initialisation function.

3.5 miscellaneous

The last thing that Guyaders patch does is reroute the mouse and keyboard input from dom0 to domU. This last step could also be implemented in a more generic way as PCI passthrough. Individual USB controllers can very well be passed through to an unprivileged domain without any special software modifications.

3.6 Limitations

Although the above patch looks promising and is relatively small it appears to be quite rigid. Since all memory locations are hard coded in to the program it is not very flexible. A short e-mail conversation with Jean Guyader revealed that currently the patch only works correct on systems with an Intel chipset and an Intel integrated graphics card. The problem is that for the patch to work is that the video BIOS needs to be executed again. This trick does not seem to work reliable on other combinations of brands of hardware.

Another problem is that it only works with the principle graphics card. It is not possible to use an other adapter for the passthrough or use multiple cards.

4 Future

The patch gives us a reasonable proof of concept. The very first thing to do of course is try to make the patch compatible with other adapters and chipsets. While most important, other interesting related projects could be thought of. We'd like to envision a few cases which might not be currently possible but could be researched later on:

1. Allow multiple video cards in one system. Either pass them to a single OS or distribute them. e.g. dom1 gets adapter1 and dom4 gets adapter2. This would for one thing require reinitialising more than one VGA BIOS.

2. Might it be possible to use one card in *dual-head* setup for more VM's simultaneously? Some graphics adapters present themselves as two separate PCI devices. Maybe with a little extra effort both heads could be assigned to another VM.

3. Switch video card live between two or more VM's. Could it be possible to have two separate OS's use the same graphics adapter? Xen could switch which of the VM's will get to output to the adapter.

At least for the first proposal multiple BIOSes need to be executed. Having difficulty with one already this could proof to be quite though for Xen. A possible approach could be to initialise one adapter, then unmap BIOS, memory and I/O, remap them for a second one and do the initialisation again. The XFree86 project is known [1] for a similar approach to support multiple cards simultaneously.

In order to have more VM's using the same monitor, Xen must be able to remap PCI devices between VM's. The decoupled OS will still try to control the VGA adapter. For a smooth transition Xen must reroute the I/O to an other memory location and might need to implement some dummy adapter to keep the decoupled OS happy.

5 Alternatives

An other big player in virtualization, VMware has the VGA problem tackled in a different way. In order to enable graphics acceleration dom0 uses the native drivers for the graphics card and boots in graphics mode. Instead of just presenting the guest OS with a simple generic VGA adapter, it will pretend it has directX abilities. When directX functions are called VMware will recompile the instructions to openGL which then is executed on the host OS. A downside on this would be that the host OS is not hidden for the user and needs to support the hardware. A likely scenario would be that the host is Linux based and not supported by the card's manufacturer.

6 Conclusion

The current patch shows that VGA passthrough is doable with relatively simple code and without breaking existing functionality in Xen. Although it is still far from flexible at the moment it seems possible to have generic VGA passthrough in Xen in the future.

While doing this research we get the feeling very few people are actively participating on this part of Xen. For this project to catch on a few things need to change. The patch is not available for any current version of Xen. It was made in May 2008 and the author told us that it was not developed in the open source tree of Xen. Looking at the current source code reveals that Xen is a very vivid project with still big source changes. This makes it hard for any one else to implement the patch. Second, the patch is not very verbose on what it precisely does. If possible, it is even worse documented then Xen's own source code at current time. Finally, PCI passthrough requires relatively new hardware which might not be available for everyone interested.

Even so, we are sure that many with us will find the project interesting enough to support. If this concept becomes mature and available for other hardware combinations in Xen's main tree we can not imagine what new exiting purposes people can come up with.

References

[1] Li-Ta Lo, Gregory R. Watson, Ronald G. Minnich *"FreeVGA: Architecture Independent Video Graphics Initialization for LinuxBIOS"* http://www.coreboot.org/data/vgabios/vgabios.html

[2] Jean Guyader *"[PATCH] Pass-through a graphic card"* http://article.gmane.org/gmane.comp.emulators.xen.devel/51194, Xen-devel mailinglist

[3] Various *"Video Graphics Array"* Wikipedia, http://en.wikipedia.org/w/index.php?title=Video_Graphics_Array&oldid=267743817, 01:42, 1 February 2009

[4] Yuji Shimada *"Development of I/O Pass-through: Current Status & the Future"* Presentation Xen Tokyo Summit, Nov 21 2008, http://www.xen.org/xensummit/xensummit_fall_2008.html

[5] Xen *"Xen Architecture Overview"* February 13, 2008, version: 1.2