Towards a HVM-like Hardware Domain for Xen

Roger Pau Monné
Citrix Systems R&D,
royger@FreeBSD.org

Abstract

Xen is a hypervisor using a microkernel design that allows running multiple concurrent operating systems on the same hardware. One of the key features of Xen is that it is OS agnostic, meaning that any OS (with proper support) can be used as a host. Xen has a long history going back to the 90s when it was designed and the early 2000s when it was released. As a consequence of this, many of the assumptions and virtualization techniques backed into it are now superseded by new hardware features, that make virtualization more transparent from an OS point of view.

This paper provides an analysis of the current interface provided by Xen to host OSes (hardware domain), it's limitations and the current work in order to improve it.

1 Introduction

Hypervisors can be divided into two categories: type 1 - those that run directly on bare metal and are in direct control of the hardware, and type 2 - hypervisors that are part of an operating system. Common type 1 hypervisors are VMware ESX/ESXi and Microsoft Hyper-V, while BHyVe and VirtualBox are clear examples of type 2 hypervisors.

Xen is a type 1 hypervisor with a twist — its design resembles a microkernel in many ways. Xen itself only takes control of the CPUs, the local and IO APICs, the MMU, the IOMMU and a timer (either HPET or PIT). The rest is taken care of by the hardware domain (sometimes also called Dom0), a specialized guest granted elevated privileges by the Hypervisor. This allows Dom0 to manage all other hardware in the system, as well as all other guests running on the Hypervisor. It is also important to realize that Xen contains almost no hardware drivers, preventing code duplication with the drivers already present in OSes. Figure 1 contains a high-level diagram of the architecture of a typical Xen deployment.

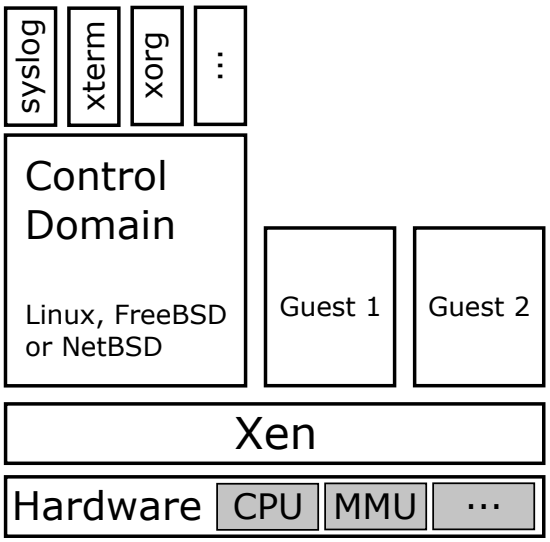

Figure 1: Xen architecture diagram

2 Current hardware domain interfaces

Due to the nature of the Xen architecture, where Xen takes ownership of some devices while leaving others for the hardware domain to manage, a slightly different interface from the one present in bare metal is needed. This interface presents dif-

ferences in the way the hardware description is provided to the hardware domain, and also affects how it interacts with physical devices.

The following is a list of the most notable interfaces that differ from bare metal:

- Page tables and memory management.
- CPU detection/enumeration and bringup.
- Setup and delivery of interrupts from physical devices.
- Bare metal ACPI tables.

Although it would seem that this list of not very big, these differences can cause a lot of impact on OSes, since their implementation is done at the core level of an OS, and not in pluggable drivers or optional components.

Page tables and memory management: traditional Xen PV guests lacked access to the MMU because it's owned by Xen, and hardware didn't provide a virtualized MMU back when PV was designed. In order to cope with this, PV guests used hypercalls in order to manage page-tables. This requires extensive modifications to core OS interfaces, since the x86 architecture only has a single MMU interface, and Xen was introducing another one.

CPU detection/enumeration and bringup: on modern ACPI systems, boot time CPU detection and enumeration is done by using the local APIC (or local x2APIC) structures found in the Multiple APIC Descriptor Table (MADT)[1]. Bringup of secondary processors (APs) is done using the local APIC. Since the description found in ACPI tables provided to the hardware domain is not accurate (we will elaborate on this later), the boot time enumeration of available CPUs needs to be done using hypercalls, so that Xen can properly tell the hardware domain about the number of CPUs. And due to the lack of a local APIC, secondary CPU bringup also needs to be done using an out-of-band method, that also relies on hypercalls.

Setup and delivery of interrupts from physical devices: on x86 systems interrupts are delivered from the local APIC to the CPU. There are several kinds of interrupts on bare metal systems:

- Legacy (PCI) interrupts: are implemented using side-band signals, and are delivered to the IO APIC, which then forwards those to the local APIC(s).
- MSI/MSI-X interrupts: implemented using in-band signals, these interrupts are delivered directly to a specific local APIC.

What type of interrupt delivery is going to be used depends on the capabilities of both the OS and the device itself. Some devices only support legacy interrupts using the IO APIC, while others support MSI and MSI-X, there are also OSes that lack MSI or MSI-X support, so only legacy interrupts can be used in that case. On bare metal the configuration of interrupts from devices is done through the PCI configuration space, so that the OS can choose which interrupt delivery to use and configure it. Since on Xen the hardware domain lacks access to a local APIC or IO APIC, interrupt configuration is done using out-of-band methods, that involve using certain hypercalls. It is also important to notice that the configuration of the device is split between Xen and the hardware domain. Xen will configure the interrupt delivery of the device on behalf of the hardware domain, while the hardware domain itself will configure other parameters of the device.

Due to the mentioned lack of native interrupt controllers in the hardware domain, delivery of interrupts from physical devices also need to be done using out-of-band methods, this involves using a Xen-specific interface called event channels, that replaces the functionality of an interrupt controller. Figure 2 contains a high-level diagram of the interrupt delivery flow from physical devices on Xen systems.

This requires a non-trivial amount of Xen code in each OS in order to perform this out-of-band configuration, that's completely different from the native approach, which also increases the burden of maintainership.

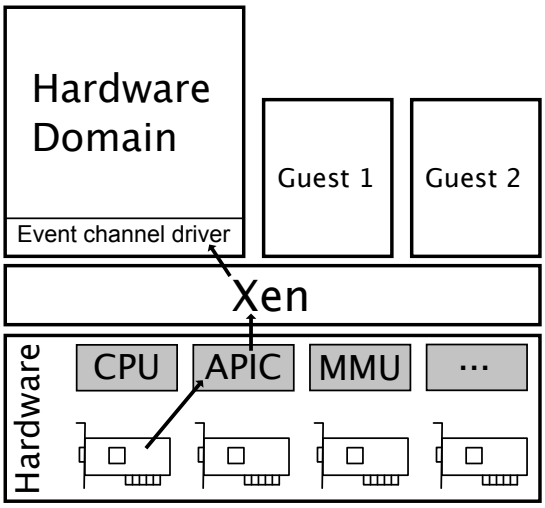

Figure 2: Xen interrupt delivery diagram

Bare metal ACPI tables: there are two very different kind of ACPI tables that can be found as part of a system description. The first ones are called static tables, and are a simple binary structure in memory, that can be directly mapped to a C structure with different fields. The content of this tables is completely static, and are generally used in order to contain information that doesn't change during system runtime and that's needed early on boot.

Parsing those tables doesn't require much ACPI knowledge, and can be done by Xen without problems. Creating this kind of tables also doesn't require much effort, since it's just a C structure mapped at a certain position.

However, there is also another kind of ACPI tables, usually called dynamic tables, that are described using ACPI Machine Language (AML). In order to access the contents of this tables an AML parser is required. There are two tables that contain AML code, the DSDT and the SSDT, and are used in order to describe the physical devices available on the system. This includes devices on the PCI bus, processors, and other platform devices, like the HPET.

All the ACPI tables are provided as-is to the hardware domain, without any change, which means that the OS needs to be aware that some of the information in those ACPI tables doesn't match the environment where it is currently running. For example, it's quite likely that the number of CPUs found in ACPI doesn't match the real number of CPUs available to the hardware domain. This requires the OS to use Xen specific out-of-band methods in order to get it's systems description, which is both cumbersome and hard to maintain.

3 Current Xen limitations

There are several limitations in the current hardware domain interface presented from Xen to the hardware domain OS, that force OS developers to use out-of-band methods in order to fetch the hardware description.

First of all, and as said above, Xen has limited capacity to process all the ACPI information, the lack of an AML parses means that Xen cannot get the full hardware description by itself, and requires the hardware domain OS cooperation in order to perform some tasks, like system shutdown, CPU C state discovery or physical CPU hotplug. All this information resides in dynamic ACPI tables, thus preventing Xen from fetching it by itself.

Apart from this, Xen is also incapable of modifying dynamic ACPI tables, since that would require an AML decompiler, an ACPI Source Language (ASL) parser/modifier and an AML compiler in order to pack the modified code. The most common tool used by UNIX like OSes in order to perform this tasks is the ACPICA tools suite[2], that contains and AML parser, decompiler and compiler. The approximate number of lines of code in ACPICA is more than 200000, while Xen itself only contains around 150000 lines of code, so there's always been a big push back into integrating any kind of AML parser/decompiler/compiler into Xen itself due to it's huge size.

Another problem that arises from the usage of ACPI, is that according to the ACPI spec there can only be one OS executing ACPI methods, shared usage of ACPI methods by two different OSes is not supported, so even if Xen had the capability to parse dynamic ACPI tables it wouldn't be allowed to execute any ACPI method, since the hardware domain OS should be the entity that interacts with

ACPI, because it's the one that contains the drivers for most of the devices present on the system.

All this limitations has lead Xen to provide many out-of-band methods in order to do hardware discovery and configuration, which involves adding a fair amount of code to each OS that wants to support running as a hardware domain. This also increases the chances of bugs in OSes, and the burden of maintainership.

4 Designing a new hardware domain interface

Due to all the issues and limitations mentioned above, it was decided that since a new guest type was being added, we could take advantage of that opportunity to introduce a new interface for the hardware domain in order to do hardware discovery and configuration. The aim of this new interface is to be as close as possible to the native one, and only resort to hypercalls or other out-of-band methods when there is no other option. Using such interface should also allow the hardware domain to take benefit of the current hardware virtualization extensions, that are commonly available even on commodity x86 hardware nowadays.

This should also help reducing the amount of Xen specific code in OSes, thus reducing the burden of maintainership and allowing developers to instead focus on other projects, or on Xen features itself.

4.1 Memory management

With the introduction of hardware virtualization extensions in the x86 architecture, now x86 processors are able to make use of second stage translations for virtual machine memory maps. This allows Xen to provide guests with a virtual memory map, and also allows them to use the hardware virtualized MMU. Thanks to this, all the Xen specific MMU code can be removed, and the guest can use the native MMU code in order to deal with memory management. Avoiding the use of hypercalls also gives the guest better performance when performing page-table operations. Figure 3 contains a diagram of the functionality provided by the second stage translation mechanism present in current x86 CPUs.

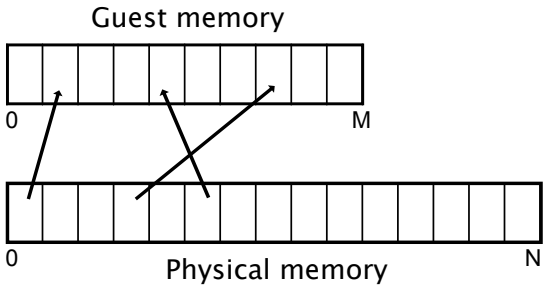

Figure 3: Second stage memory management

4.2 Interrupt management

In order to use the same interface as bare metal, the hardware domain will be provided with a set of emulated local APICs, and as many emulated IO APICs as present in the bare metal system, which might vary. Figure 4 shows a high-level diagram of the proposed solution. This is important for two reasons, for once it will prevent the hardware domain from having to implement and support a specific interface when running on Xen, thus allowing OSes to take advantage of their already available code to deal with interrupts. Secondly it will enable Xen and the hardware domain to take advantage of advanced hardware virtualization technologies, like local APIC hardware virtualization and Posted Interrupts[3]. Using such technology allows interrupts generated from physical devices to be directly injected to guests (the hardware domain in this case) without Xen intervention, which reduces interrupt latency and increases interrupt delivery rate.

The configuration of interrupts is also going to be done using native methods, which involves using the PCI configuration space. Xen is going to setup traps in order to detect access to the PCI configuration space (which resides in IO space), and any extended configuration space areas (which resides in memory areas), in order to detect attempts from the hardware domain to setup interrupts and properly

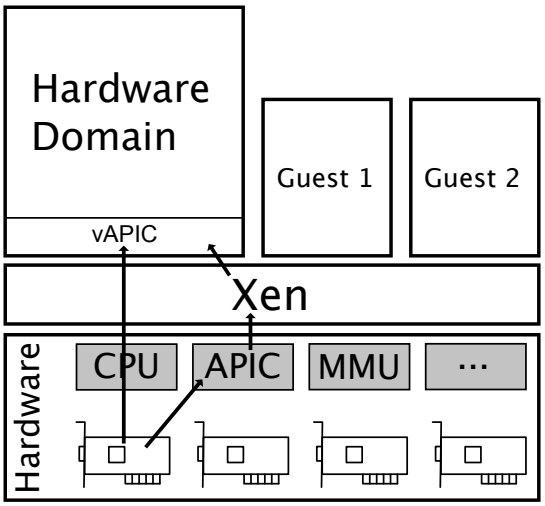

Figure 4: Proposed interrupt injection

react on them, like it's currently done for HVM domains. Xen will have to setup an emulation layer for any MSI or MSI-X areas reported by the PCI config space for each device that is made available to the hardware domain. The MSI configuration is performed by accessing certain registers on the PCI configuration space itself, while MSI-X configuration is performed by accessing a memory region, which is reported as a Base Address Register in the PCI header of each device. Configuration of legacy PCI interrupts is going to be detected by the emulated IO APIC itself, which is emulated inside of Xen.

4.3 Fixup of ACPI tables

A more problematic topic is the fixup of ACPI tables presented to the hardware domain. As explained above, Xen used to simply pass the native ACPI tables to the hardware domain, and then the hardware domain using some Xen-specific exceptions would decide which devices to use, and which part of the information present in ACPI tables is actually describing the system it is currently running on. The end goal is to provide a set of ACPI tables to the hardware domain that correctly describe the hardware that's available to it's hardware container and not the full hardware present on the system.

In order to do this, Xen will have to craft a new version of the MADT, so that it correctly accounts for the number of CPUs available to the hardware domain, and the correct address of the emulated local APIC. This doesn't involve a lot of work, since the MADT is a static table that can easily be created by Xen. The problem arises from the fact that each local APIC struct in the MADT also has a processor object in the DSDT table, which sadly it's a dynamic table and Xen cannot modify at all.

The proposed solution to workaround the fact that Xen cannot modify dynamic ACPI tables relies on using the ACPI Status Override Table (STAO)[4]. This table was introduced by the Xen on ARM team, and allows to hide devices described in the dynamic tables without having to modify them. By using it Xen can hide the physical processor objects found in the DSDT from the hardware domain, preventing it from accessing them in any way.

Then and extra SSDT table is going to be appended to the hardware domain, that contains processor objects for virtual CPUs, so that all the local APIC structs in the MADT have corresponding processor objects in the ACPI namespapce. This is required for doing CPU hotplug of virtual CPUs, that is implemented using a GPE device block and the return value of the _STA and _MAT methods of processor objects[5].

5 Conclusions

The end goal of this work is to provide an interface as similar as possible to bare metal for all Xen guests. This is done in order to both reduce the Xen specific code in Xen guests, so that less maintainership is required, and in order to expand the number of OSes with Xen support. By presenting such interface, and removing the need to perform any out-of-band Xen specific configuration the Xen Project expects to expand the base of Xen-enlightened OSes, that are capable of acting as both guests and hosts (hardware domains).

References

[1] Unified EFI, Inc. *Advanced Configuration and Power Interface Specification*, http://www.uefi.org/sites/default/files/resources/ACPI_6_1.pdf

[2] Intel Corp. *ACPI Component Architecture*, https://acpica.org

[3] Feng Wu. *XenSummit 2014: Intel Virtualization Technology for Directed I/O (VT-d) Posted Interrupts* https://www.xenproject.org/presentations-and-videos/video/xpds14-intel-r-virtualization-technology-for-directed-i-o-vt-d-posted-interrupts-feng-wu.html

[4] Al Stone, Graeme Gregory, Parth Dixit *ACPI Specification for Status Override Table* https://lists.xen.org/archives/html/xen-devel/2016-08/pdfYfOWKJ83jH.pdf

[5] Roger Pau Monné *CPU hotplug support for PVH* https://lists.xen.org/archives/html/xen-devel/2017-01/msg01039.html