Paravirt Ops on Linux IA64

Isaku Yamahata
VA Linux Systems Japan K.K.
yamahata@valinux.co.jp

Abstract

The Xen/IA64 community has been working on IA64 paravirt ops with a view to merging the Xen/IA64 changes into the Linux upstream tree. Following the method used for x86, paravirt_ops was implemented and then Xen was modified to run on top of paravirt_ops. In this document firstly paravirt_ops will be reviewed and then paravirt_ops on IA64 will be discussed. In particular, why it is necessary on IA64, differences between the x86 and IA64 implementations, why these difference exist, what challenges exist, and the current approaches to those challenges. To conclude the current status of IA64 paravirt ops will be summarized and the future plan will be discussed.

1 Introduction

The Xen/IA64 community has been working on IA64 paravirt ops with a view to merging the Xen/IA64 changes into the Linux upstream tree. Following the method used for x86, paravirt_ops was implemented and then Xen was modified to run on top of paravirt_ops.

1.1 What is paravirt_ops?

paravirt_ops (pv_ops for short) was developed as a method of supporting virtualization on the Linux kernel on x86. It has been developed as an API, not an ABI. It allows each hypervisor to override operations which are important for that hypervisor at the API level. And it allows a single kernel binary to run on all supported execution environments including the native machine. One significant difference from usual function pointer table is that it allows optimizations with binary patches.

The operations of paravirt_ops are classified into three categories.

(A) Simple indirect call
These operations correspond to high level functionality and thus the overhead of an indirect call isn't very important.

(B) Indirect call which allows optimization with a binary patch
Usually these operations correspond to low level instructions. They are called frequently and are performance critical. Thus low overhead is very important.

(C) A set of macros in hand-written assembly code
Hand-written assembly code (.S files) also need paravirtualization because they include sensitive instructions or performance critical code paths.

Table 1: Old History

date	subject	author	version
30 Jun, 2005	Xen and the Art of Linux/IA64 Virtualization	Dan Magenheimer	2.6.12
28 Oct, 2005	virtualization hooks patch for IA64 2.6.15	Dan Magenheimer	2.6.15
03 Jun, 2006	[RFC 0/2] Xen/IA64 kernel changes 2.6.17-rc5	Alex Williamson	2.6.17-rc5

2 paravirt_ops/IA64

2.1 Old History

The history of the merge of Xen/IA64 paravirt_ops into upstream Linux is shown in table 1. At this time things have moved on. In particular:

- x86 pv_ops has been merged
- Xen/x86 has been merged

This has made work on merging Xen/IA64 paravirt_ops a lot easier.

Table 2: IA64 pv_ops

name	description	type
pv_info	general info	-
pv_init_ops	initialization	A
pv_cpu_ops	privileged instruction	B
pv_cpu_asm_ops	macros for assembly code	C
pv_iosapic_ops	iosapic operations	A
pv_irq_ops	irq related operations	A
pv_time_op	steal time accounting	A

2.2 Implementation

Linux/IA64 has the IA64 machine vector functionality which allows the kernel to switch implementations (e.g. initialization, ipi, dma api...) depending on executing platform. One approach to implementing paravirt_ops/IA64 would be to enhance the machine vector. However, we adopted a pv_ops approach instead. The figure 1 shows the relationship of pv_ops to the machine vector and the rest of kernels. IA64 domU is implemented as a combination of xen domU machine vector and xen domU pv_ops.

2.2.1 paravirt_ops/IA64

Because of architectural differences between IA64 and x86, the resulting set of functions is very different from x86 pv_ops as shown in table 2

2.2.2 pv_cpu_asm_ops

To paravirtualize hand written assembly code (i.e. .S files), the approach of single source code and compile multiple times with different macros definitions was adopted for maintenance purpose.

2.2.3 pv_cpu_ops

Currently this class of functions corresponds to a subset of IA64 intrinsics that is a subset of privileged instructions.

2.2.4 Binary Patching

At the time of writing these hooks are defined as C indirect function pointers, but in order to support binary patch optimization they will be changed using GCC extended inline assembly code. However, it is not possible to describe all the clobbered registers for an IA64 C function call in cases. So this is not a generic solution.

In the first phase, the binary patching feature was abandoned because it was felt that it was difficult to reach an agreement on the calling convention.

3 Merge

3.1 Merging Strategy

To make the merge easier, the x86 strategy was followed. For the first merge:

- minimize modifications
- postpone optimization where possible

As a later phase, domU optimization and dom0 support (though dom0 on x86 isn't supported yet) will be addressed.

3.2 Resulting Patches

Table 3 shows that pv_ops/IA64 relies heavlily on hand written assembly code. The number of pv_cpu_ops shows that IA64 uses fewer function pointers than x86. This is mainly because the on privileged registers are represented by the "mov" instruction and thus it corresponds to a single function pointer on IA64. In contrast, on x86 such opperations are represented by many function pointers.

Another reason that pv_ops/IA64 has smaller number of hooks than x86 is that the current pv_ops/IA64 set was determined only by the requirements of Xen/IA64 as is currently the only user of pv_ops/IA64. While pv_ops/x86 is utilized by many virtualization technologies including Xen, VMWare, lguest and KVM.

Xen/IA64 fully virutualizes MMU so that it has no pv_mmu_ops.

3.3 Current Status and Future Plans

Currently the paravirt_op/IA64 development is at early phase as per table 4. The first patches were merged into the Linux/IA64 test branch and minimal Xen/domU patches are waiting for the next merge window because some necessary patches are in the x86/Xen tree so it's necessary to sync-up Linux/IA64 with that.

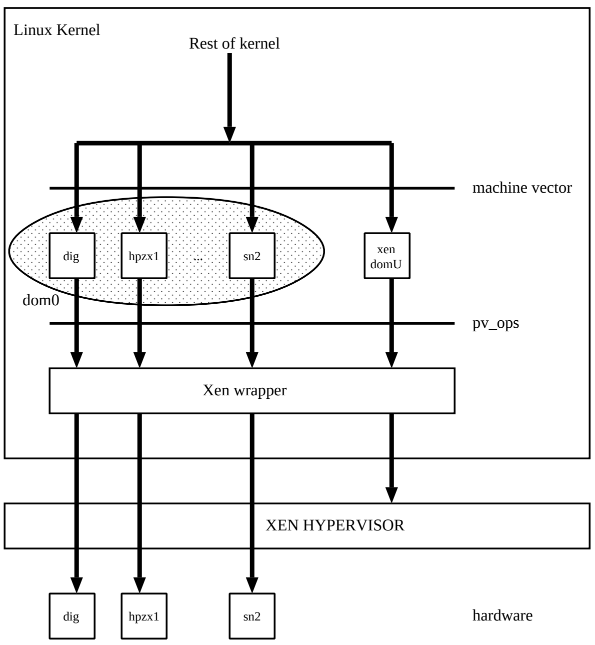

Figure 1: the relation to machine vector

Table 3: the number of hooks in pv_ops

items	x86_32(2.6.26-rc6)	IA64
pv_init_ops	5	6
pv_cpu_asm_ops	7 macros	32 macros 6 asm labels
pv_cpu_ops	32	14
pv_mmu_ops	29	-
others	13	18

Table 4: IA64 pv_ops status

items	status
minimal domU	
pv_ops	in linux IA64 test
Xen/domU	W.I.P.
domU optimization	
dom0	

4 Acknowledgements

I'd like to thank those who have contributed to paravirt_ops/IA64 development. Especially Eddie Dong, Alex Williamson (the previous Xen/IA64 maintainer), Akio Takebe, Qing He, and Simon Horman. I'd also like to give thanks to the Linux/Xen maintainer, Jeremy Fitzhardinge and the Linux IA64 maintainer, Tony Luck. Finally special thanks to the Xen/IA64 initiator, Dan Magenheimer.

References

[1] Rusty Russel, lguest: Implementing the little Linux hypervisor In Preedings of the Linux Symposium, July 2007.

[2] J. Nakajima and A.K. Mallick Hybrid-Virtualization – Enhanced Virtualization for Linux, in Proceeding of the Linux Symposium, July 2007

[3] Daniel J. Magenheimer, Thomas W. Christian, vBlades: Optimized paravirtualization for the Itanium processor family, Proceedings of the Thrd Virtual Machine Research and Technology, May 2004.

[4] Havard K. F. Bjerke, HPC Virtualization with Xen on Itanium, July 2005. http://openlab-mu-internal.web.cern.ch/openlab-mu-internal/Documents/Reports/Technical/Thesis_HarvardBjerke.pdf

[5] Joshua LeVasseur, Volkmar Uhlig, Matthew Chapman, Peter Chubb, Ben Leslie, Gernot Heiser Pre-Virtualization: Slashing the Cost of Virtualization In Fakultat fur Informatik, Universit?t Karlsruhe (TH), Technical Report 2005-30, November 2005

[6] Joshua LeVasseur, Volkmar Uhlig, Matthew Chapman, Peter Chubb, Ben Leslie, and Gernot Heiser Pre-Virtualization: Soft Layering for Virtual Machines In Technical Report 2006-15, Fakultat fur Informatik, Universitet Karlsruhe, July 2006