paravirt_ops/IA64

Isaku Yamahata <yamahata@valinux.co.jp>
VA Linux Systems Japan K.K.
Introduction
Goal

- Merge xenLinux/IA64 modification to Linux upstream. Both domU/dom0 support.
Old History

• There have already been several unsuccessful merge efforts

• So we are trying again...

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
<th>Author</th>
<th>Kernel Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>30 Jan, 2005</td>
<td>Xen and the Art of Linux/IA64 Virtualization</td>
<td>Dan Magenheimer</td>
<td>2.6.12</td>
</tr>
<tr>
<td>28 Oct, 2005</td>
<td>virtualization hooks patch for IA64 2.6.15</td>
<td>Dan Magenheimer</td>
<td>2.6.15</td>
</tr>
<tr>
<td>03 Jun, 2006</td>
<td>Xen/IA64 kernel changes 2.6.17-rc5</td>
<td>Alex Williamson</td>
<td>2.6.17-rc5</td>
</tr>
</tbody>
</table>
What's Different This Time?

- The x86 paravirt_ops (pv_ops for short) has been merged
  - After several approaches to virtualization support were proposed, finally the paravirt_ops approach won
  - Consensus on virtualization via source-code API

- (Minimal) Xen/x86 has been merged
  - The Xen common code is already there
  - Though, portability patches would be necessary
Merging Strategy

- The first merge is the most difficult
- Basically follow the x86 approach
  - Virtualization infrastructure via source code API, i.e. paravirt_ops.
  - Minimize modifications at first step
    - Make patch size and the number of patches as small as possible for code review.
  - Postpone optimization where possible
  - Dom0 support is also postponed
What is paravirt_ops

• The indirect layer for virtualization
  - Virtualization support via source-code API
  - It allows a single kernel binary to run on all supported execution environments including bare-metal hardware.
  - It is implemented as C function pointer.
  - But, it supports special optimization, binary patching.
    • Indirect calls can be transformed into a direct call or in-place execution.
  - A set of macros for assembly code
    • Hand-written assembly code also needs paravirtualization.
Challenges

- IA64 machine vectors
- Privileged instruction (para)virtualization
- Binary patching
- Hand-written assembly code paravirtualization
IA64 Machine Vector

• C function pointers for platform support.
  - Initialization, IPI, DMA API...

• One possible approach is to enhance machine vector.

• But paravirtualization needs more.
  - Initialization at very early stage of the boot sequence.
  - Binary patch optimization.

  • Paravirtualization overhead on bare metal hardware should be as small as possible.
Linux kernel

The rest of kernel code

machine vector

paravirt_ops

Xen Wrapper

Xen Hypervisor

hardware

dig
hpzx1...
sn2

Xen domU

dom0

dig
hpzx1...
sn2
Instruction (Para)Virtualization

• The main issues for IA64 is privileged instruction paravirtualization.

• Some approaches were discussed.
  - Pre-Virtualization and after-burning
  - Hybrid Virtualization. Rely on hardware support (VT-i).
  - Privify. (vBlades approach)
  - paravirt_ops.
## (Para)Virtualization (cont.)

<table>
<thead>
<tr>
<th>Method</th>
<th>Hypervisor Change</th>
<th>Linux Change</th>
<th>Performance in Theory</th>
<th>Single Binary?</th>
</tr>
</thead>
<tbody>
<tr>
<td>previrtualization</td>
<td>yes</td>
<td>No</td>
<td>Low</td>
<td>No</td>
</tr>
<tr>
<td>hybrid</td>
<td>yes</td>
<td>No</td>
<td>Low</td>
<td>Yes</td>
</tr>
<tr>
<td>privify</td>
<td>yes</td>
<td>No</td>
<td>Low</td>
<td>No</td>
</tr>
<tr>
<td>paravirt_ops</td>
<td>No</td>
<td>Yes</td>
<td>High</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- With the preliminary benchmark results (the next slides), paravirt_ops approach was adopted.
- The re-aim-7 which was used is CPU intensive benchmark.
Preliminary benchmark done by Alex Williamson


<table>
<thead>
<tr>
<th></th>
<th>Max JPM</th>
<th>% of native</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>242347.19</td>
<td>100.00%</td>
</tr>
<tr>
<td>Para-virt</td>
<td>218694.95</td>
<td>90.24%</td>
</tr>
<tr>
<td>Full-virt</td>
<td>58289.32</td>
<td>24.05%</td>
</tr>
</tbody>
</table>
Binary Patch

• IA64 intrinsics (privileged instructions used by C code.) needs binary patch
  - They are performance critical.
  - e.g. mask/unmask interrupts.

• But not all the hooks needs binary patch
  - The hooks for high level functionalities accepts C indirect call overheads.
  - Unlike x86, we don’t support binary patch for all the hooks because...
Binary Patch (cont.)

- To allow binary patch, the call instruction needs to be annotated. But there is no way to write C calling conversion with GCC extended inline assembly code.

- So other calling convention has to be used.
  - Non-banked static registers (xen/ia64 like)
  - Banked Static registers. (PAL call like)
  - C function call like convention.

- Anyway those functions can't be written in C.
caller_func(in0, ... inL)
  loc0, ... locM
out0, ... outN

bp_func(out0, out1)
other_func(out0, out1, ... outN)

asm("... ::: "out0, "out1") is bad.
Here out0, ... outN are clobbered.
No way to specify out0, ... outN
as clobbered registers where N is unknown.

func(in0,..., inL)
r0-r31
M, N are determined by GCC

Static registers
in0...inL
loc0...locM
out0, out1...outN

Clobbered registers

bp_func(in0, in1)
r0-r31
M’, N’ are determined by GCC

Static registers
in0, in1
loc0...locM’
out0...outN’
Hand-Written Assembly Code

- .S files
- Kernel entry (fault handler, system call) / exit and context switch.
- Stack is not always usable.
  - For example, only several registers can be used on fault.
- They are well tuned so that it’s difficult to touch them without performance degradation.
Hand Written Assembly Code (cont.)

- Single source file, compile multiple times using CPP macros and Makefile tricks.

- Most of those macros 1:1 correspond to single instruction

Example Patch

\[
\begin{align*}
(p8) \text{ mov cr.itir} &= r25 \\
+ \quad \text{MOV_TO_ITIR}(p8, r25, r24)
\end{align*}
\]

Native

\[
\begin{align*}
\text{#define MOV_TO_ITIR}(\text{pred, reg, clob}) & \quad \backslash \\
\text{(pred) mov cr.itir} &= \text{reg} & \quad \backslash \\
& \quad \text{CLOBBER(clob)}
\end{align*}
\]

Xen

\[
\begin{align*}
\text{#define MOV_TO_ITIR}(\text{pred, reg, clob}) & \quad \backslash \\
\text{(pred) movl clob} &= \text{XSI_ITIR;} & \quad \backslash \\
& \quad ;; & \quad \backslash \\
& \quad \text{(pred) st8 [clob]} &= \text{reg}
\end{align*}
\]

Extra clobberable registers are found by careful code reading.
Switching Hand-Written Assembly Code

- Fault handler is pointed by cr.iva register.
- Other path is switched by indirect call.
  - Clobberable registers for branch are found by careful code reading.
PV Checker

- .S paravirtualization is fragile.
  - It’s very easy to write raw instruction.
  - Enforce people to use paravirtualized instructions.

- Simple checker is implemented.
  - Doesn’t cover all the breakage.
  - Most of easy breakage can be detected.

Example: cover instruction case
#define COVER nop 0
#define cover .error "cover should not be used directly."
Current Status and Future Plans
Current IA64 pv_ops

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pv_info</td>
<td>general info</td>
<td>-</td>
</tr>
<tr>
<td>pv_init_ops</td>
<td>initialization</td>
<td>normal C function pointers</td>
</tr>
<tr>
<td>pv_cpu_ops</td>
<td>privileged instructions</td>
<td>Normal C function pointers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Needs binary patch</td>
</tr>
<tr>
<td>pv_cpu_asm_ops</td>
<td>macros for hand-written assembly</td>
<td>asm macros</td>
</tr>
<tr>
<td>pv_iosapic_ops</td>
<td>iosapic related operations</td>
<td>normal C function pointers</td>
</tr>
<tr>
<td>pv_irq_ops</td>
<td>irq related operations</td>
<td>normal C function pointers</td>
</tr>
<tr>
<td>pv_time_ops</td>
<td>steal time accounting</td>
<td>normal C function pointers</td>
</tr>
</tbody>
</table>

- The resulting set of hooks is completely different from x86’s.
- That’s because of architecture difference between x86 and IA64.
This figures shows paravirt_ops/IA64 issues are in cpu instruction paravirtualization.
## Comparison with x86 pv_ops

<table>
<thead>
<tr>
<th>items</th>
<th>X86_32 (2.6.26-rc6)</th>
<th>IA64</th>
</tr>
</thead>
<tbody>
<tr>
<td>pv_init_ops</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>pv_cpu_asm_ops</td>
<td>7 macros</td>
<td>32 macros/6 asm labels</td>
</tr>
<tr>
<td>pv_cpu_ops</td>
<td>32</td>
<td>14</td>
</tr>
<tr>
<td>pv_mmu_ops</td>
<td>29</td>
<td>n/a</td>
</tr>
<tr>
<td>others</td>
<td>13</td>
<td>18</td>
</tr>
</tbody>
</table>

- Again this figure shows that paravirt_ops/IA64 relies on hand-written assembly code paravirtualization.

- The number of IA64 pv_cpu_ops is smaller than x86's.
  - On IA64 privileged register operation is done by 'mov' instructions which is replaced by 2 function pointers.
  - On the other hand on x86 many function pointers are defined. e.g. load_cr3()

- Xen/IA64 fully virtualizes MMU so that pv_mmu_ops is unnecessary.
## Activity

<table>
<thead>
<tr>
<th>Date</th>
<th>subject</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 Jan, 2008</td>
<td>Time for hybrid virtualization?</td>
<td>discussion</td>
</tr>
<tr>
<td>07 Feb, 2008</td>
<td>forward ported to linux ia64 upstream</td>
<td>single jumbo patch</td>
</tr>
<tr>
<td>17 Feb, 2008</td>
<td>paravirt_ops suport in IA64</td>
<td>paravirt_ops discussion</td>
</tr>
<tr>
<td>21 Feb, 2008</td>
<td>ia64/xen domU paravirtualization</td>
<td>split patches</td>
</tr>
<tr>
<td>24 Feb, 2008</td>
<td>ia64/xen: paravirtualization of hand written assembly code</td>
<td>discussion</td>
</tr>
<tr>
<td>26 Feb, 2008</td>
<td>RFC: ia64/xen TAKE 2: paravirtualization of hand written assembly code</td>
<td></td>
</tr>
<tr>
<td>28 Feb, 2008</td>
<td>RFC: ia64/pv_ops: ia64 intrinsics paravirtualization</td>
<td></td>
</tr>
<tr>
<td>05 Mar, 2008</td>
<td>ia64/xen take 3: ia64/xen domU paravirtualization</td>
<td>started pv_ops</td>
</tr>
<tr>
<td>09 Apr, 2008</td>
<td>RFC: ia64/pv_ops take 4: ia64/xen domU take 4</td>
<td>Fully converted into pv ops</td>
</tr>
<tr>
<td>01 May, 2008</td>
<td>ia64/pv_ops take 5, ia64/xen domU take 5</td>
<td>pv_ops part was merged to linux</td>
</tr>
<tr>
<td>19 May, 2008</td>
<td>ia64/pv_ops take 6, ia64/xen domU take 6</td>
<td>ia64 test branch</td>
</tr>
<tr>
<td>10 Jun, 2008</td>
<td>ia64/xen domU take 7</td>
<td>preliminary for save/restore</td>
</tr>
</tbody>
</table>
## Current status and future plan

<table>
<thead>
<tr>
<th>Items</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>minimal domU</strong></td>
<td></td>
</tr>
<tr>
<td>pv_ops</td>
<td>in Linux IA64 test branch</td>
</tr>
<tr>
<td>Xen/domU</td>
<td>W.I.P. (working patch, under review)</td>
</tr>
<tr>
<td><em>balloon</em></td>
<td></td>
</tr>
<tr>
<td>save/restore</td>
<td>Preliminary support. Needs more paravirtualization</td>
</tr>
<tr>
<td>free unused pages</td>
<td></td>
</tr>
<tr>
<td>revise boot protocol</td>
<td></td>
</tr>
<tr>
<td>domU optimization</td>
<td></td>
</tr>
<tr>
<td>more .S paravirtualization</td>
<td></td>
</tr>
<tr>
<td>binary patch</td>
<td>Experimental patch existed</td>
</tr>
<tr>
<td>balloon</td>
<td></td>
</tr>
<tr>
<td><strong>domO</strong></td>
<td></td>
</tr>
<tr>
<td>DMA API</td>
<td></td>
</tr>
<tr>
<td>kexec/kdump</td>
<td></td>
</tr>
<tr>
<td>PMU/PMD/PMC(xenoprof)</td>
<td></td>
</tr>
<tr>
<td>mca</td>
<td></td>
</tr>
<tr>
<td>and more?</td>
<td></td>
</tr>
</tbody>
</table>
Acknowledgements

• Thanks to
  - Eddie Dong(some of slides are cited from his slides with modification)
  - Alex Williamson(The previous Xen/IA64 maintainer)
  - Akio Takebe
  - Qing He
  - Simon Horman
  - Jeremy Fitzhardinge(The Linux xen maintainer)
  - Tony Luck(The Linux IA64 maintainer)
  - Dan Magenheimer(The Xen/IA64 initiator)
Thank you