Virtio-trace
- Towards the flexible fast interconnection between guest and host for tracing

Tracing Summit 2012
Masami Hiramatsu <masami.hiramatsu.pt@hitachi.com>
Yoshihiro YUNOMAE <yoshihiro.yunomae.ez@hitachi.com>
Linux Technology Center
Yokohama Research Lab., Hitachi, Ltd.
Enterprise systems are moving on (private/public) cloud which uses virtualization technology and aim for system consolidation.

- Multiple servers run on one physical system
- This makes system trouble shooting harder
Troubles in Virtualized System

- A guest VM...
  - Can be affected by other VM operation
  - Can die by host OS or hypervisor’s bug

Host-Guest inter-VM tracing helps root cause analysis
Challenges of Inter-VM Tracing

- **Tracing virtualized system has following challenges**
  1. Synchronize time-stamp for each VM’s log
     - Each VM has own clock
  2. Collect guests trace-log from host without overhead
     - Too huge trace logs (# of VM times of logs)

```
+----------------+       +----------------+       +----------------+
|     VM1        |       |     VM2        |       |     Host       |
|   tracer      | ─────►|   tracer      | ─────►|   tracer      |
|               |       |               |       |               |
| Guests:        |       | Guests:        |       |               |
|               |       |               |       |               |
|               |       |               |       |               |
|               |       |               |       |               |
|               |       |               |       |               |
|               |       |               |       |               |
|     VM1        |       |     VM2        |       |     Host       |
|               |       |               |       |               |
|               |       |               |       |               |
|               |       |               |       |               |
|               |       |               |       |               |
|               |       |               |       |               |
|               |       |               |       |               |
| Trace Logs    |       | Trace Logs    |       | Trace Logs    |
```

© Hitachi, Ltd. 2012. All rights reserved.
First Prototype (~May 2012)

- Use **systemtap** as a tracer
  - Trace both of kernel and user applications
- Use **tsc_offset** to adjust guests’ time-stamp-counter (tsc)
  - This prototype use TSC for timestamp
  - Also, a VM is pinned on a CPU
- Use **IVRing** to pass the trace logs from guests to host
  - IVRing is an implementation of Inter-VM Ring buffer using IVShmem
- Use **TimeDoctor** to visualize trace data
Each guests has virtualized independent tsc

- The substitution of host’s tsc and guests’ one is called tsc_offset
- Each tracers in guests uses own tsc to record time stamp, and we can adjusting this tsc using tsc_offset to merge logs.
- Of course, this depends on “constant_tsc”
Adjusting TSC on Prototype

Save `tsc_offset` of each VMs @vmexit for each vm via systemtap’s procfs

Output the pair of qemu-kvm’s PID and its `tsc_offset`
IVRing - Fast Interconnect for Tracing

- **Guests**
  - systemtap
  - QEMU
  - Tracing Guest Events
  - ivshmem_server

- **Host**
  - systemtap
  - QEMU
  - IVRing
  - Get Host Trace data from debugfs
  - /proc/.../tsc_offset
  - Get TSC offset from procs
  - ivtrace (Merge & Analyze Host and Guests Trace data)
  - Logs
  - Time Doctor data
  - Time Doctor
  - Visualized Trace Data

© Hitachi, Ltd. 2012. All rights reserved.
IVRing on IVShmem

- A ring-buffer IVRing is constructed on IVShmem as a data path for trace data of a guest.
- IVShmem is a memory-PCI device
  - Backend memory is a posix shmem.
- IVTrace can read the data without memory copying.

**Diagram:**
- Guest
  - SystemTap
  - IVRing driver
  - IVShmem
  - IVSS
- QEMU
  - IVShmem
- Shmem is shared with IVTrace
- Host
  - POSIX shmem
  - Notify to IVTrace using eventfd
  - IVTrace
  - SystemTap driver
  - Writes data into ringbuffer
  - A ring buffer IVRing is constructed on IVShmem
  - Read trace data From shmem
The case of no contentions
### The case of no contentions

<table>
<thead>
<tr>
<th>August</th>
<th>September</th>
<th>October</th>
<th>November</th>
<th>December</th>
</tr>
</thead>
<tbody>
<tr>
<td>GS1 GS2 GS3</td>
<td>GS1 GS2 GS3</td>
<td>GS1 GS2 GS3</td>
<td>GS1 GS2 GS3</td>
<td>GS1 GS2 GS3</td>
</tr>
<tr>
<td>I/O start</td>
<td>I/O finish</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Guest1**

**Host**
The case of contentions. I/O slowdown
The case of contentions. I/O slowdown

- I/O start
- I/O finish
- Disturb Guest1’s request
Proposal of IVRing on LKML

Maintainers Don’t like it

(2012/06/06 8:22), Greg Kroah-Hartman wrote:
> On Wed, Jun 06, 2012 at 07:03:06AM +0800, Anthony Liguori wrote:
>> On 06/05/2012 09:10 PM, Borislav Petkov wrote:
>>> Yet another ring buffer?
>>> We already have an ftrace and perf ring buffer, can’t you use one of those?
>>>
>> Not to mention virtio :-)
>> Why not just make a virtio device for this kind of thing?
>
> Yeah, that’s exactly what I was thinking, why reinvent things again?

Points

- NO “yet another ring buffer” in linux kernel
  - Use ftrace and perf ring buffer for guest recording
- Use virtio instead of ivshmem
**Virtio Candidates**

- **Virtio-shmem (new device)**
  - Virtio device which provides APIs for assigning the guest pages to shared memory in host
    - Guest can assign any page to host’s shmem
    - Qemu remaps original pages with the pages on shared memory
    - Similar to the ivshmem, but no big PCI address space required

- **Virtio-serial with splice (enhancement)**
  - Virtio device which provides chardev interface for guest
    - Guest can “splice” its data into the char device
    - Qemu copies data page to host-side pipe
    - Vhost can offload the copying process
**Virtio-shmem Overview**

- **Export Ftrace ring-buffer pages directly to host**
  - How we can export per-cpu kmalloc object?

  - **Can we use perf RB too?**
  - **Can we correctly recognize readable pages?**
  - **Can we use perf RB too?**

  1. **Get the ftrace Ring Buffer Pages**
  2. **Pass the pages to host**
  3. **Allocate a POSIX shmem**
  4. **Modify shadow PT to swap given pages with shm pages**

  **Slab and per-cpu info must be shared**

© Hitachi, Ltd. 2012. All rights reserved.
To generalize interface and guarantee memory coherency
  - At first, we have simple ring-buffer for memory coherency

- Remove a page from RB
  - Ftrace reader kthread
  - Simple RB
  - Virtio-shm driver
  - Guest

- Copy the page to simpleRB with barrier
  - (1) Allocate pages for shm
  - (2) Pass the pages to host

- This still use yet another ring buffer on shm!
  - (3) Allocate a POSIX shm
    - (4) Modify shadow PT to swap given pages with shm pages

- 2 copies v.s. no interaction
  - Read and write (copy)
  - Host
    - Qemu
    - Virtio-ring
  - Reader
  - Storage
Virtio-serial opens chardev in the guest and FIFO(named pipe) in the host

- Remove a page from RB
- Ring Buffer (ftrace)
- Ftrace reader agent
- Splice to pipe (no copy)
- Splice from pipe to virtio (no copy)
- Pass the page to virtio-ring
- Copy the page to Linux pipe
- Splice to file (no copy)
- Splice multiple page at once. Per-page virtio is too heavy

Guest

Pipe

Ftrace reader agent

Virtio-console driver

Virtio-ring

Virtio-serial bus

Host

Qemu

Reader

Storage
<table>
<thead>
<tr>
<th></th>
<th>IVRing</th>
<th>Virtio-serial w/ splice</th>
<th>Virtio-shmem w/ simpleRB</th>
<th>Virtio-shmem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Qemu-Reader</strong></td>
<td>Shmem</td>
<td>Pipe (1way)</td>
<td>Shmem</td>
<td>Shmem</td>
</tr>
<tr>
<td><strong>Guest-Qemu</strong></td>
<td>Shmem(PCI)</td>
<td>virtio</td>
<td>Shmem(EPT)</td>
<td>Shmem(EPT)</td>
</tr>
<tr>
<td><strong># of Copies</strong></td>
<td>1 (reader to file)</td>
<td>1 (virtio to pipe)</td>
<td>2 (in Guest, reader to file)</td>
<td>1 (reader to file)</td>
</tr>
<tr>
<td><strong>Guest-host interaction</strong></td>
<td>No</td>
<td>Once per I/O (16pages or more)</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Supported tracer</strong></td>
<td>SystemTap</td>
<td>Ftrace, user tools</td>
<td>Ftrace, and others?</td>
<td>Ftrace only</td>
</tr>
<tr>
<td><strong>No another RB</strong></td>
<td>IVRing</td>
<td>No</td>
<td>Simple RB</td>
<td>No</td>
</tr>
<tr>
<td><strong>Use virtio</strong></td>
<td>No</td>
<td>Yes</td>
<td>Hmm...</td>
<td>Hmm...</td>
</tr>
<tr>
<td><strong>Buffer Resize</strong></td>
<td>No</td>
<td>Support</td>
<td>Support</td>
<td>Support</td>
</tr>
<tr>
<td><strong>SMP scaling</strong></td>
<td>RB w/ lock</td>
<td>Per-cpu pipes</td>
<td>Per-cpu shmem</td>
<td>Per-cpu shmem</td>
</tr>
<tr>
<td><strong>VCPU hot-add</strong></td>
<td>Support (w/lock)</td>
<td>Add channels</td>
<td>Add shmem</td>
<td>Add shmem</td>
</tr>
<tr>
<td><strong>Live migration</strong></td>
<td>No</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td><strong>Expectation</strong></td>
<td>Not scalable, not acceptable</td>
<td>Upstream Acceptable</td>
<td>Need Discussion</td>
<td>Need Discussion</td>
</tr>
</tbody>
</table>

© Hitachi, Ltd. 2012. All rights reserved.
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>IVRing</th>
<th>Virtio-serial w/ splice</th>
<th>Virtio-shmem w/ simpleRB</th>
<th>Virtio-shmem w/o simpleRB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qemu-Reader</td>
<td>Shmem</td>
<td>Pipe (1way)</td>
<td>Shmem (EPT)</td>
<td>Shmem (EPT)</td>
</tr>
<tr>
<td>Guest-Qemu</td>
<td>Shmem(PCI)</td>
<td>virtio</td>
<td>Shmem (EPT)</td>
<td>Shmem (EPT)</td>
</tr>
<tr>
<td># of Copies</td>
<td>1 (reader to file)</td>
<td>1 (virtio to pipe)</td>
<td>2 (in Guest, reader to file)</td>
<td>1 (reader to file)</td>
</tr>
<tr>
<td>Guest-host interaction</td>
<td>No</td>
<td>Once per I/O (16pages or more)</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Supported tracer</td>
<td>SystemTap</td>
<td>Ftrace, user tools</td>
<td>Ftrace, and others?</td>
<td>Ftrace only</td>
</tr>
<tr>
<td>No another RB</td>
<td>IVRing</td>
<td>No</td>
<td>Simple RB</td>
<td>No</td>
</tr>
<tr>
<td>Use virtio</td>
<td>No</td>
<td>Yes</td>
<td>Hmm...</td>
<td>Hmm...</td>
</tr>
<tr>
<td>Buffer Resize</td>
<td>No</td>
<td>Support</td>
<td>Support</td>
<td>Support</td>
</tr>
<tr>
<td>SMP scaling</td>
<td>RB w/ lock</td>
<td>Per-cpu pipes</td>
<td>Per-cpu shmem</td>
<td>Per-cpu shmem</td>
</tr>
<tr>
<td>VCPU hot-add</td>
<td>Support (w/lock)</td>
<td>Add channels</td>
<td>Add shmem</td>
<td>Add shmem</td>
</tr>
<tr>
<td>Live migration</td>
<td>No</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Expectation</td>
<td>Not scalable, not acceptable</td>
<td>Upstream Acceptable</td>
<td>Need Discussion</td>
<td>Need Discussion</td>
</tr>
</tbody>
</table>

### Chosen method
- Simple implementation
- No claiming points
- Fit to virtio framework
Prototype consists of trace-agent, virtio-console driver, virtio-pipe device, and trace-cmd

- Open percpu trace_raw and pipe
- Splice trace data to serial channel

- Export guest’s debugfs by diod + socat

- Create a channel for 9pfs control
- Create a channel for Agent control
- Create channels for trace data

- Create a channel for 9pfs control
- Create a channel for Agent control
- Create channels for trace data

- Export guest’s debugfs

- Open named pipe for 9pfs
- Open named pipe for agent
- Open named pipes for data
- Copy the page to Linux pipe

- Get event format data
  Enable/disable tracing

- Splice per-cpu buffer to the file

- Control Agent via control pipe

© Hitachi, Ltd. 2012. All rights reserved.
Compared with IVRing, virtio-trace is really fast?
- Running UnixBench with tracing on guest VM

<table>
<thead>
<tr>
<th></th>
<th>Scores</th>
<th>Overhead(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No trace</td>
<td>28807569.5</td>
<td>0.00%</td>
</tr>
<tr>
<td>virtio-trace</td>
<td>28685049.5</td>
<td>0.43%</td>
</tr>
<tr>
<td>ivring</td>
<td>28418595.5</td>
<td>1.35%</td>
</tr>
<tr>
<td>virtio-serial</td>
<td>13262258.7</td>
<td>53.96%</td>
</tr>
</tbody>
</table>

Compared with ivring, virtio-trace has lower overhead!
Maintainers accept it 😊

On Thu, 09 Aug 2012 21:30:29 +0900, Yoshihiro YUNOMAE <yoshihiro.yunomae.ez@hitachi.com> wrote:
> Hi All,
> The following patch set provides a low-overhead system for collecting kernel
> tracing data of guests by a host in a virtualization environment.

Thankyou!

I’ve applied this, and it will head into linux-next in the next few
days.

Cheers,
Rusty.

Points

- NO “yet another ring buffer” in linux kernel
  - Use virtio ring for passing data
- More generic feature (not only for tracing)
  - Maybe useful for other use, like SPICE
So, what is the next step?

- Synchronize time-stamp between guest and host
  - Tsc-based trace_clock is an option (for constant_tsc machine)
  - Or agent gives the guest’s trace_clock offset (but how?)
    - In generic, vmexit pattern matching can give us a hint.

- Consolidate 9pfs server to trace-agent
  - For simplicity and ease of use (reducing steps of setup)
  - We can prepare setup script for guest tracing

- Fix some issues on Qemu’s chardev(serial backend)
  - Hotplug issue
  - Guest blocking issue

- Live migration support
Conclusion

- IVRing is not acceptable for upstream
- Improved virtio-serial to support splice is accepted
  - Build the prototype and measure the performance
- Still under development...
  - Make setup easier
  - Time synchronizing
  - Clarify “chardev” issues on Qemu
  - Fix Qemu chardev for CPU-hotplug and non-blocking

TO BE CONTINUED...
Previous Talks

- See, IVTrace slide @ LinuxCon Japan 2012
  - Low-Overhead Ring-Buffer of Kernel Tracing
  - Tracing Across Host OS and Guest OS
Law Legal Statements

- Linux is a registered trademark of Linus Torvalds.
- UNIX is a registered trademark of The Open Group.
- All other trademarks and copyrights are the property of their respective owners.