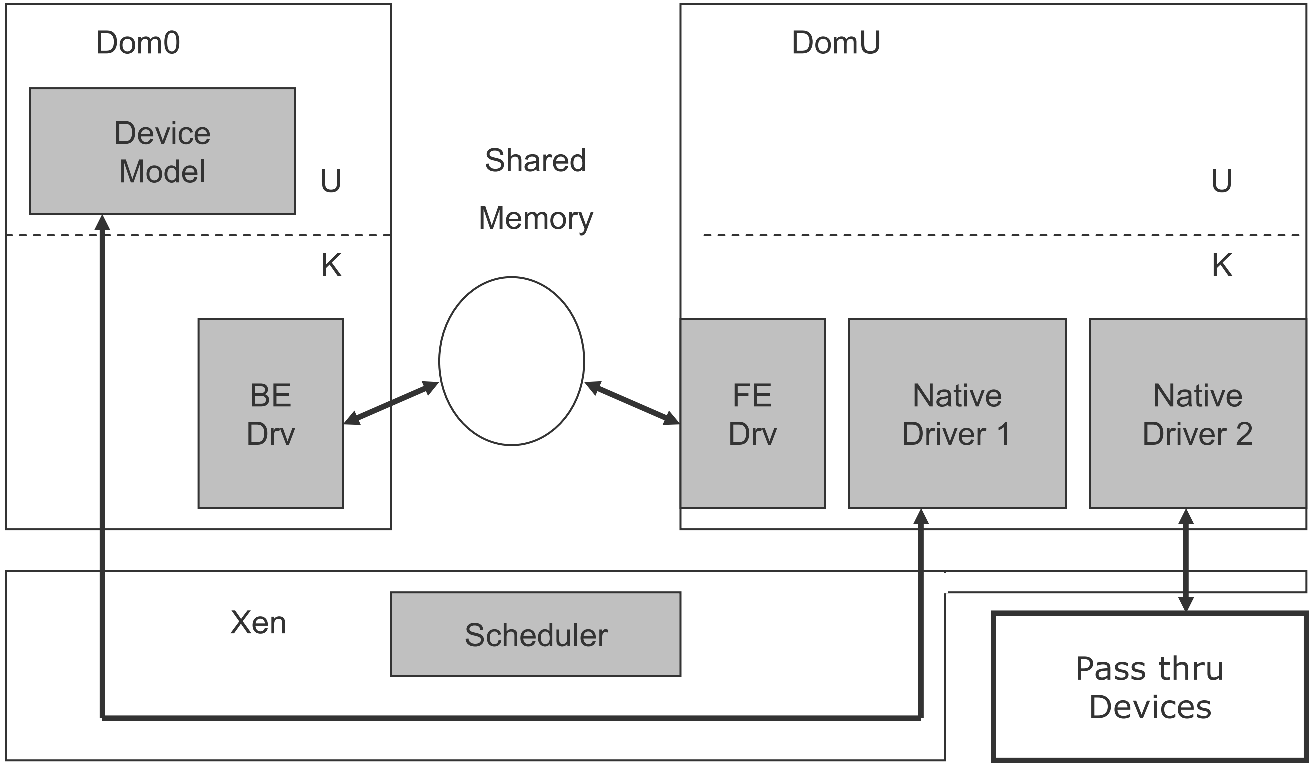
Towards High-Quality I/O Virtualization

Yaozu Dong, Jinquan Dai, Zhiteng Huang, Haibing Guan, Kevin Tian, Yunhong Jiang

IO virtualization in Xen



What is high-quality I/O virtualization

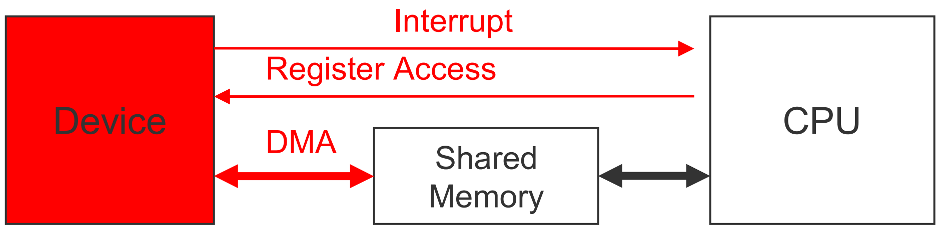
High-quality I/O virtualization

- Complete device semantics
- Full-feature set
- Close-to-native performance
- Real-time response

Gap of existing solutions

- Software approaches
  - Intrinsic virtualization overhead
  - Fail to catch up full-feature set
- Existing direct I/O solutions
  - Ignore the fact of staggering variety of PC hardware, especially for client devices
  - Lack of complete device semantics
  - Ignorant about driver virtualization hole which prevents from wide adoption
- Real time response is sacrificed

Driver (CPU) ←→ device interaction



Interaction between device and driver:

- Driver programs device through register access
- Device notifies driver through interrupt
- Device could DMA for massive data movement

High quality I/O virtualization requires above semantics to be intact

Preserving complete device semantics is a key to vast commodity devices

Preserving device semantics – State

Run-time device semantics

- Naturally preserved due to IO registers pass-through

Initial device semantics – risk of inconsistency

- A reclaimed device may have been set to an arbitrary state by previous user
- An in-fly transaction may access reclaimed memory

High quality I/O virtualization addresses inconsistency

- Initialize reclaimed device into known state as BIOS does at boot phase
- Device Function Level Reset (FLR)
  - — FLR is optional PCIe capability
- PCI link reset
  - — Upstream switch may not exist
- D0 → D3 → D0 power state transition
  - — Lead to state reset for most devices

Preserving device semantics – Interrupt

Interrupt sharing - compromise isolation

- Guest may assert/de-assert the shared interrupt line to arbitrary state, or even generate interrupt storm

High quality I/O virtualization embraces host MSI

- Dedicated vector(s) for device
- If guest is working in MSI mode
  - — Remap guest MSI capability to host MSI
- If guest is working in INTx mode
  - — Emulate virtual interrupt line state according to host MSI event. E.g:
    - Asserting when host MSI fires
    - De-asserting when EOI is issued

Preserving device semantics – Caching

Device may use 'cache-bypass DMA'

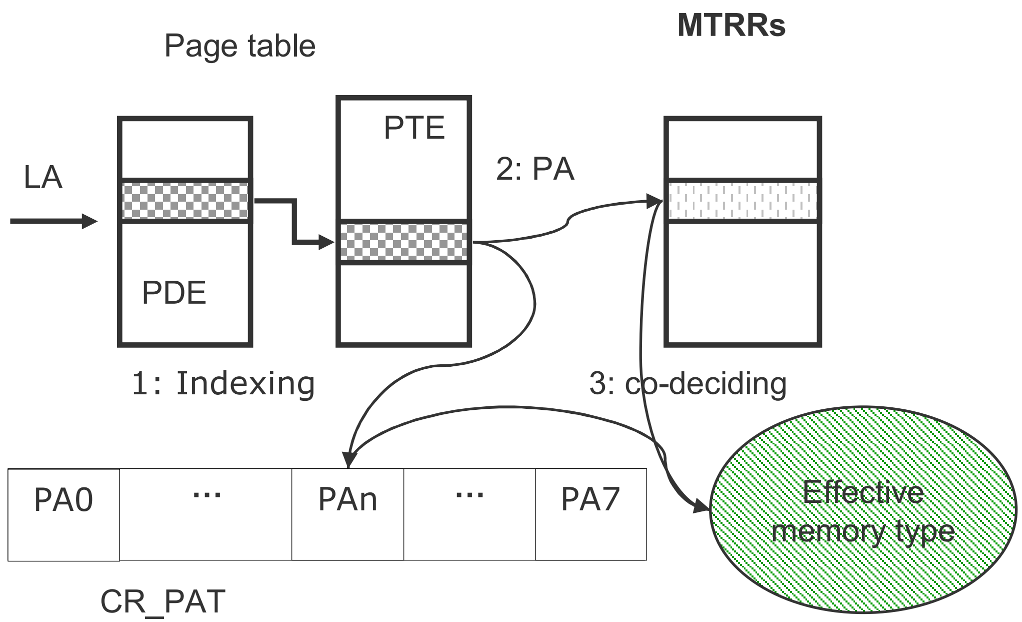
- "No Snoop" type in DMA message
- Driver ensures cache coherency
  - — Flush cache before notifying device to start DMA etc.

Incorrect cache semantics may lead to device malfunction

High quality I/O virtualization ensures strict cache semantics, by propagating guest effective memory type to host

- Derived from MTRR (indexed by physical address), and PAT (indexed by PAT/PCD/PWT bits in PTE)

Propagating guest effective memory type



- Guest effective memory type is derived from guest MTRR/PAT
- Program shadow PTE (taking effective with host MTRR) to have same effective memory type
  - — Host MTRR is not changed for performance reason

Driver virtualization hole prevents direct I/O from wide adoption

Staggering variety of PC hardware

- Build-in device is originally designed to be bound with the platform
- Different HW features such as "No-Snoopy" control may be employed in different device

Drivers originally developed for native environment never foreseeing that they would run in virtual environment

Device resource in direct I/O

Sensitive device resources (SDR)

- Defined in public specification, e.g:
  - — Standard PCI resources such as BAR and function header type etc.
  - — Platform resource such as device BDF
- VMM trap-and-emulates SDR by public defined interfaces

Non-sensitive device resources (NSDR)

- Device specific registers which VMM doesn’t need to know
- Simply pass through

Driver virtualization hole (DVH)

Drivers, accessing ***SDR*** bypassing virtualization layer, can lead to unexpected result in direct I/O

—*This is coined as driver virtualization hole for direct I/O*

Examples of DVH

- *Acquiring SDRs without using standard interface defined in relevant public specifications*
- *Using sensitive device resources for operations other than those defined in relevant public specification*
- *Accessing platform specific resource that does not belong to the device*

Acquiring SDRs

Acquiring SDRs without using standard interface

- VMM emulates SDRs by trapping at standard interface
- Acquiring SDRs using device specific knowledge won't get right information reflecting the virtual platform

```
dev_priv->fb_location = (RADEON_READ
(0x148) & 0xffff) << 16;
```

file "diver/char/drm/radeon_cp.c"

```
status = er32(STATUS);
bus->func = (status &0xC) >> 2;
```

file "driver/net/e1000e/lib.c"

Utilizing SDRs

Using SDRs for operations other than those defined in public specification

- For example, BDF is used to identify an PCI function, using it to specify MAC address of NIC could lead to mac address confliction in virtual environment

```
/* Flip last bit of mac address if
we're on second port */
if (hw->bus.func == E1000_FUNC_1)
     hw->mac.perm_addr[5] ^= 1;
```

file “driver/net/e1000e/lib.c”

Accessing platform specific resource

Accessing platform specific resource, which does not belong to the device, may lead to DVH

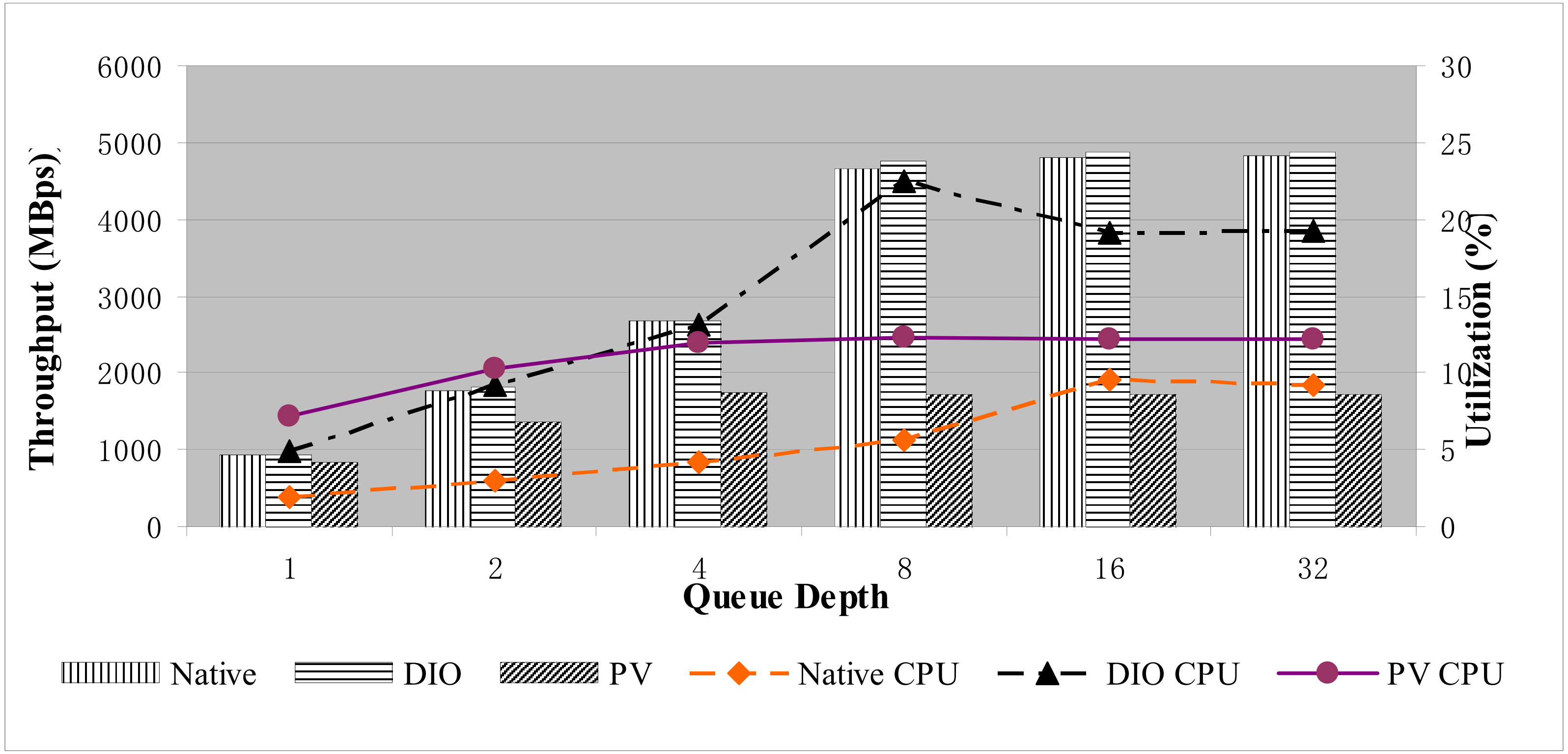
- Integrated device driver may directly access chipset specific registers
  - *Works in native environment*
  - *But prevents from running virtually as direct I/O since the guest chipset may be different from physical one*

Performance of high quality I/O virtualization

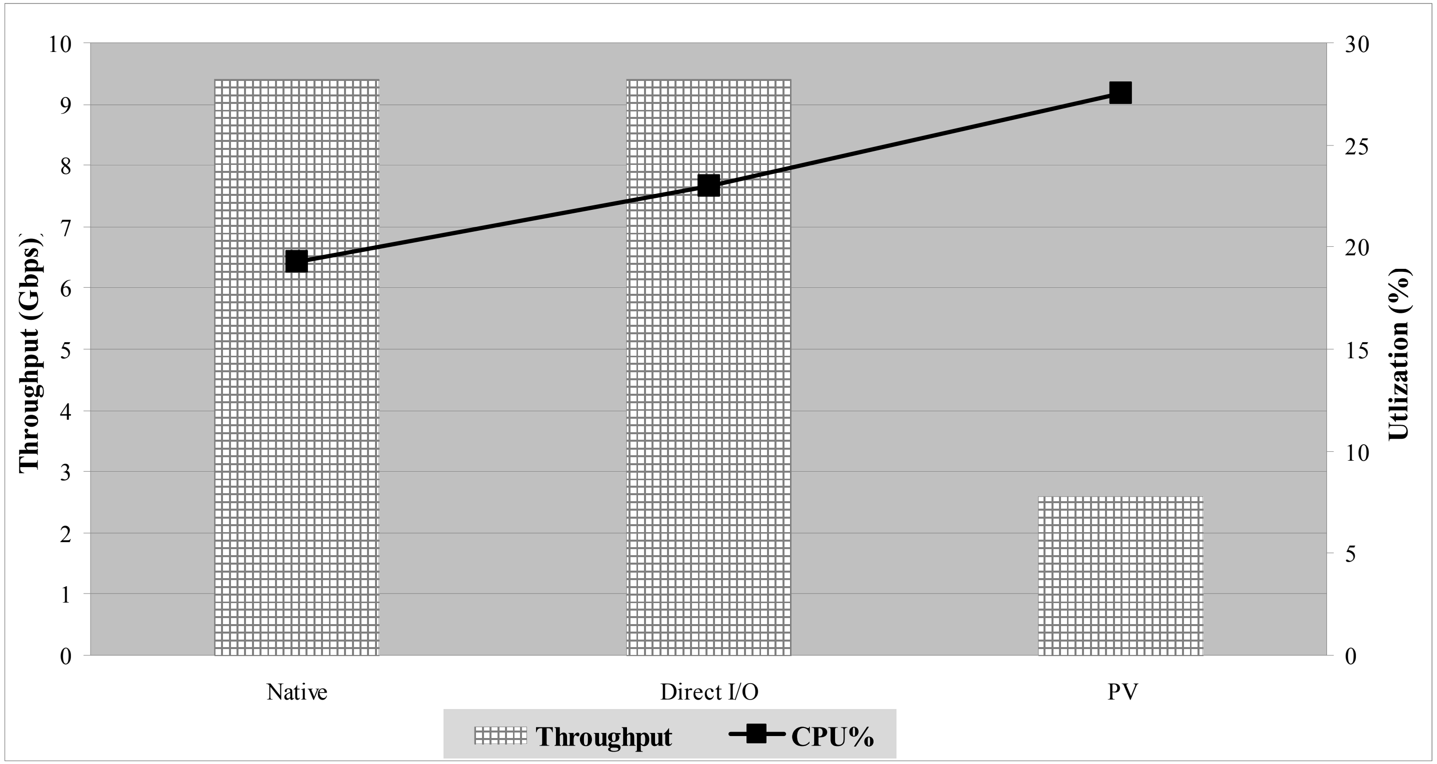
Performance of high-quality I/O virtualization

- Up to 2.86X of PV disk performance
- Up to 3.6X of PV network
  - — NIC saturates CPU at 2.6Gb/s for 10Gbit Ethernet.
  - — Utilizing VMDq technology can improve the bandwidth to 8.2Gb/s, but still suffer from CPU utilization and bandwidth.
- Within 3.76% of native for video
  - — PV graphic virtualization solution such as VMGL suffers from losing of full-feature set.

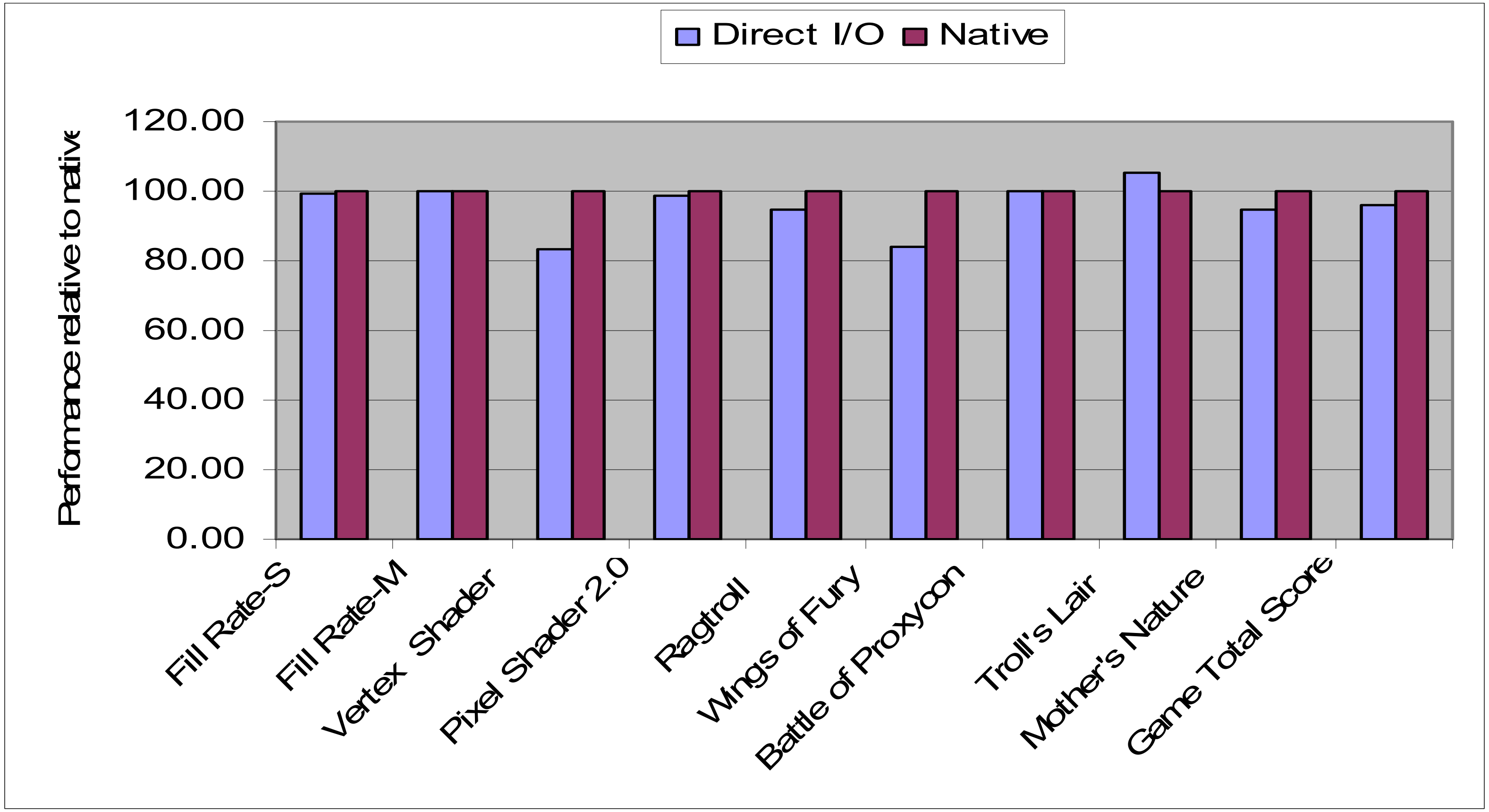
Disk direct I/O: Up to 2.86X of PV performance



Network direct I/O: Up to 3.6X of PV performance



Graphics direct I/O: Within 3.76% of native



But, how about Audio?

Direct I/O doesn’t solve all the problems without real-time response

- Buffer overruns of input stream
  - — Lost of input data
- Buffer underruns of output stream
  - — Glitch

Benchmarking audio quality

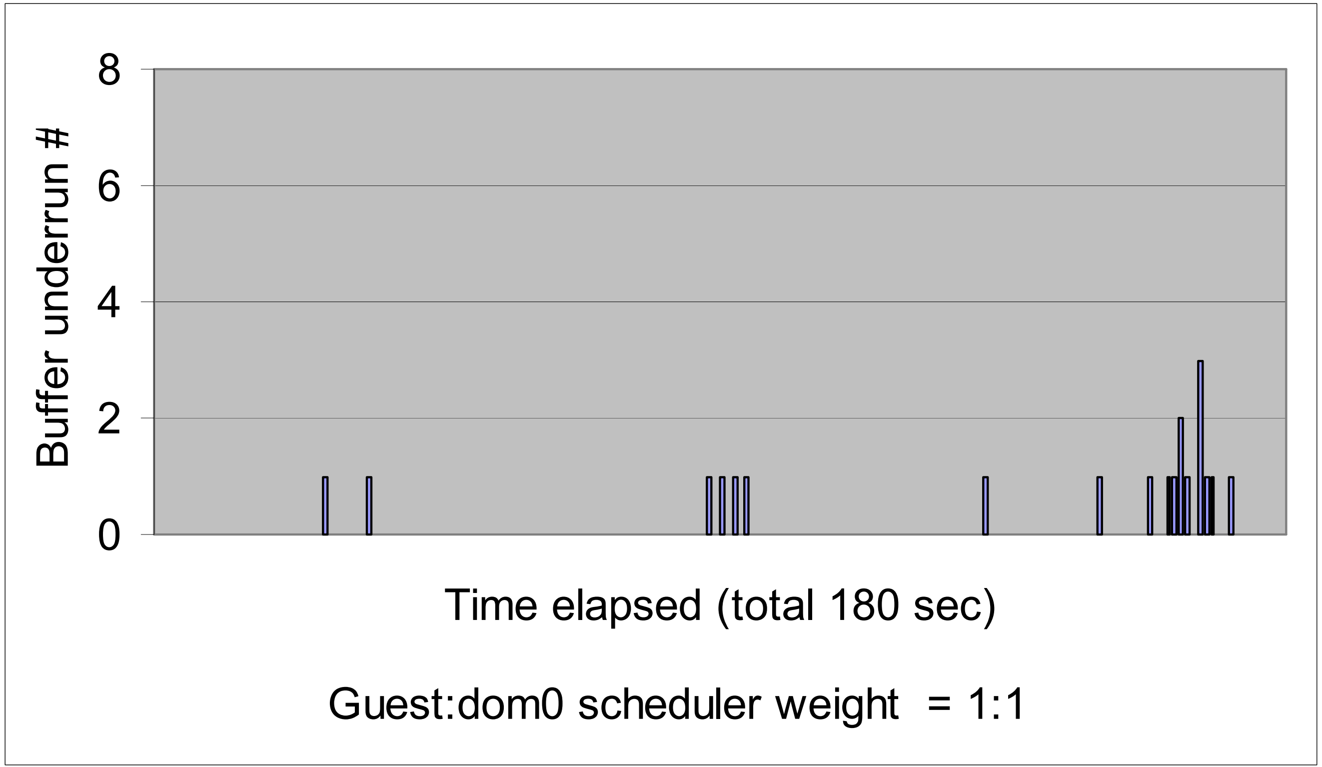
Bandwidth is not a key concern, but buffer underrun/overrun is.

- Run Amarok music player as workload
- Instrument ALSA driver to measure buffer underrun with audio direct I/O
  - Run UP guest with dom0 on top of Xen
    - VCPUs of both domains are pinned to same pCPU
    - A busy loop application in dom0 to compete CPU cycles
  - Assign audio card to guest.

Xen credit scheduler focus on fairness

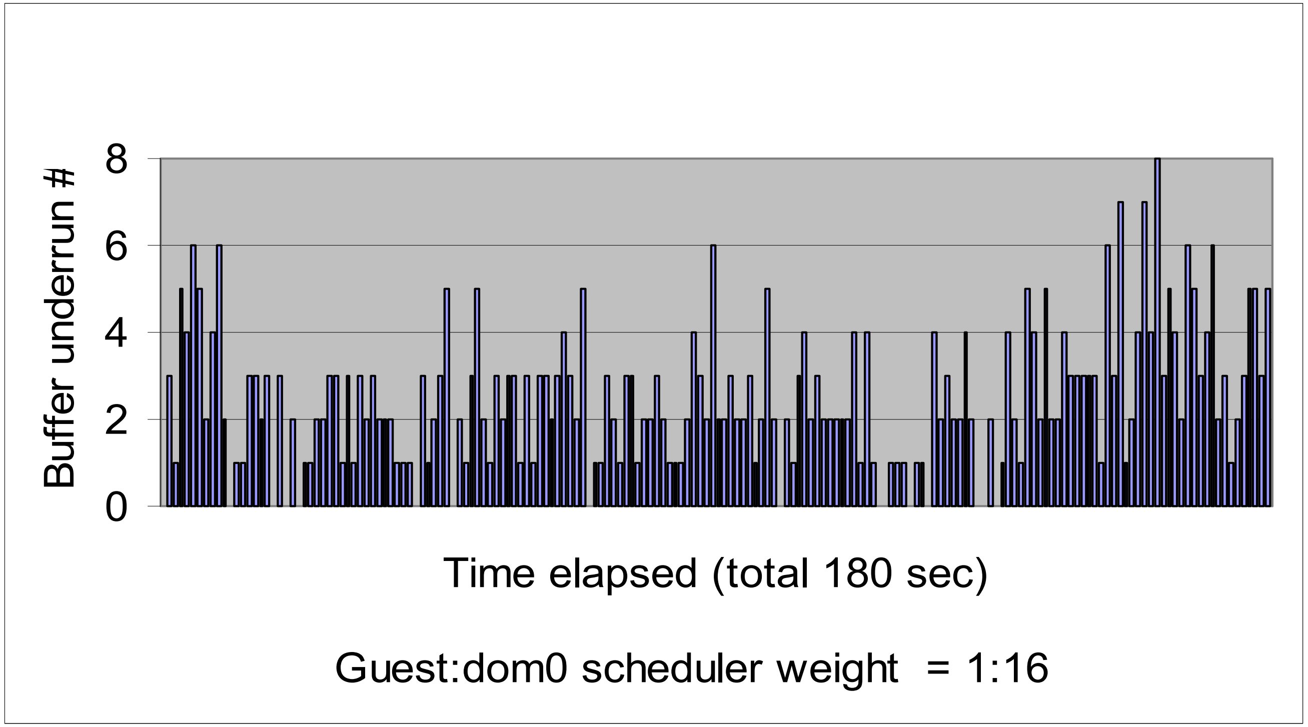
- BOOST state helps in reducing IO response latency, but not guaranteed.

With ½ (1:1) CPU reservation



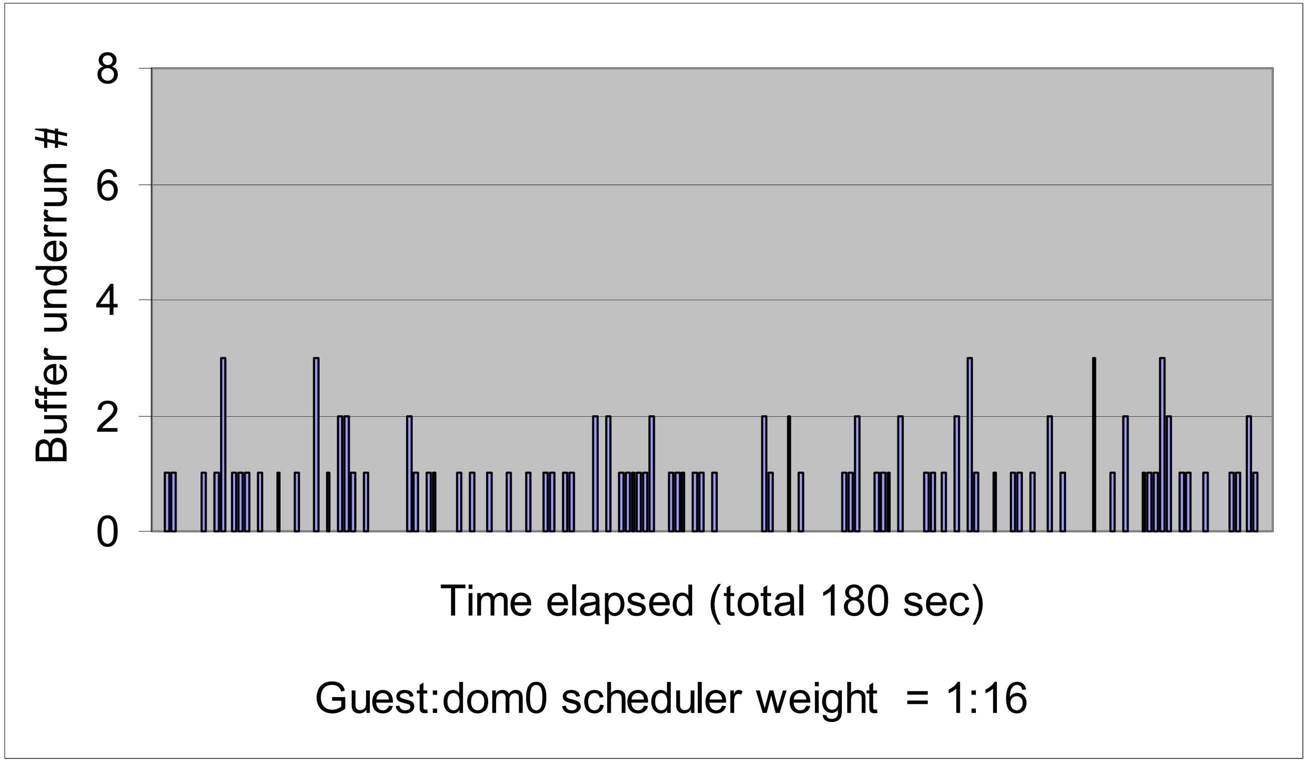
Buffer underrun is observable with ½ CPU reservation
(Xen default scheduler)

With 1/17 (1:16) CPU reservation



Frequent buffer underruns with 1/17 CPU reservation
(Xen default scheduler)

Reducing scheduler tick to 1ms

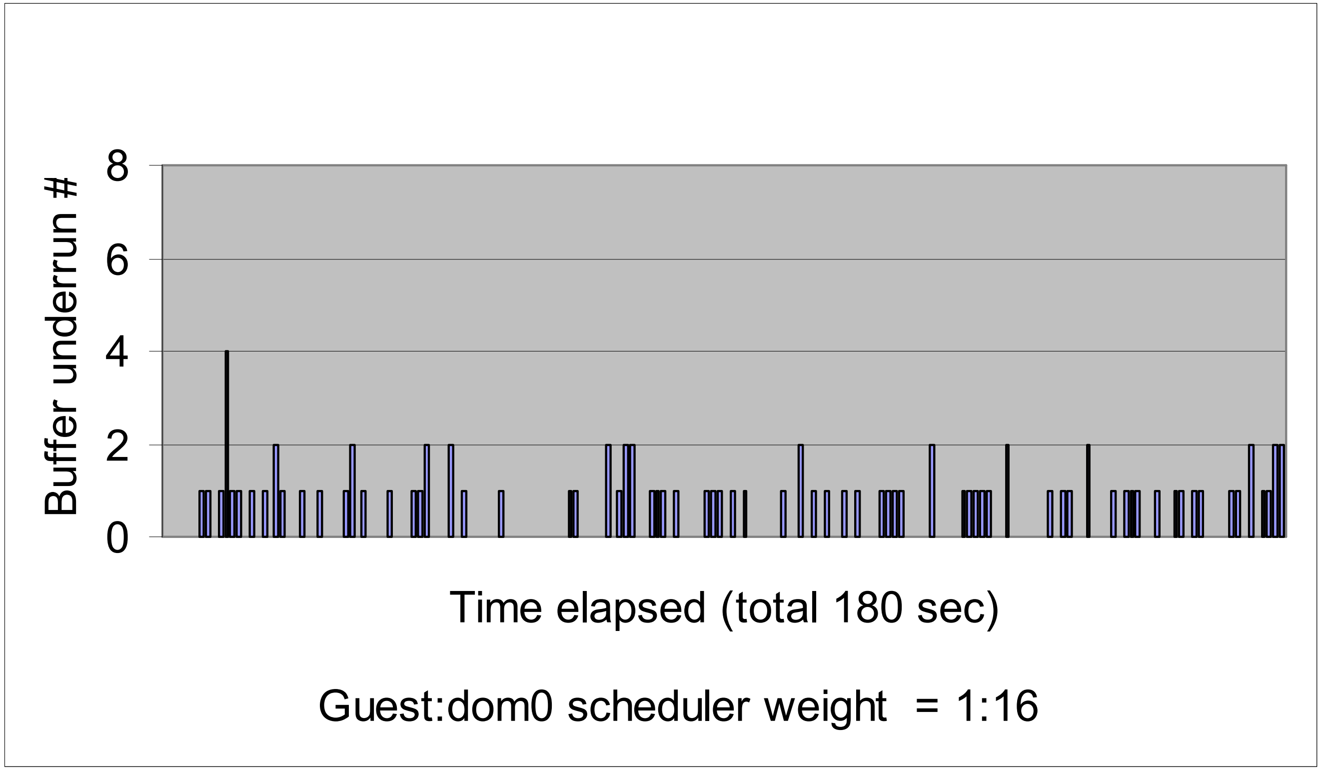


Scheduler tick, from 10ms default to 1ms, reduces average buffer underrun frequency from 2.47 per second to 0.594 for 1/17 CPU reservation

But...

- Smaller scheduler tick also means performance overhead…
- REAL_TIME VMM scheduler could meet both performance and response issue
  - — Schedule guest when the audio buffer is consumed, i.e. DMA interrupt.

REAL_TIME scheduler



Average audio buffer underrun frequency drops from average of 2.47 in default credit scheduler to 0.506 for 1/17 CPU reservation

Summarize

Our contribution toward high quality I/O virtualization:

- Preserving complete device semantics for direct I/O
- Avoiding driver virtualization hole
- Improving VMM scheduler for real-time response