Xen PV Performance Status and Optimization Opportunities

Author: Zhidong Yu
Contributor: Ronghui Duan, Yuan Zhou, Jian Zhang

Intel Corporation

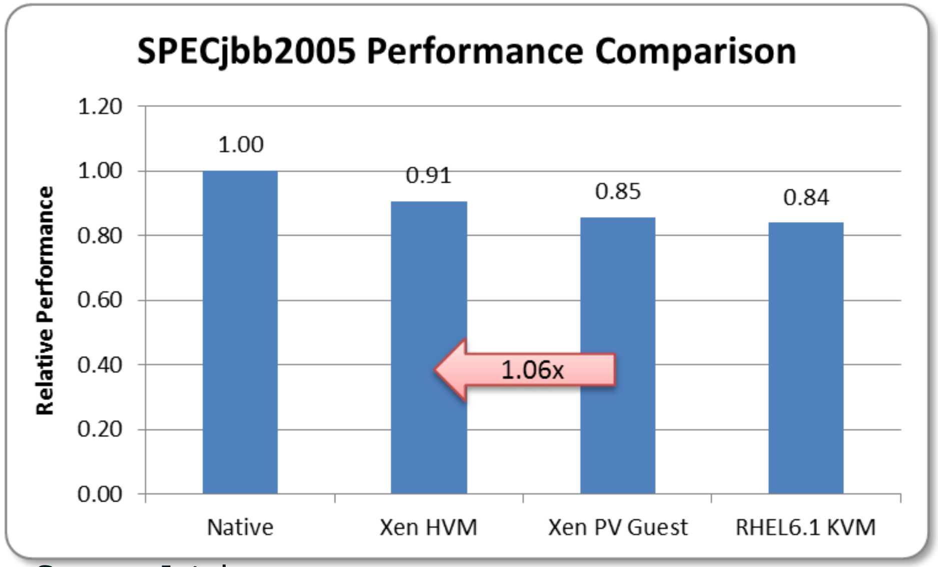
Disclaimers

Agenda

- HVM vs. PV guest comparison
- PV disk driver performance and optimizations

HVM vs. PV Guest – SPECjbb (1/2)

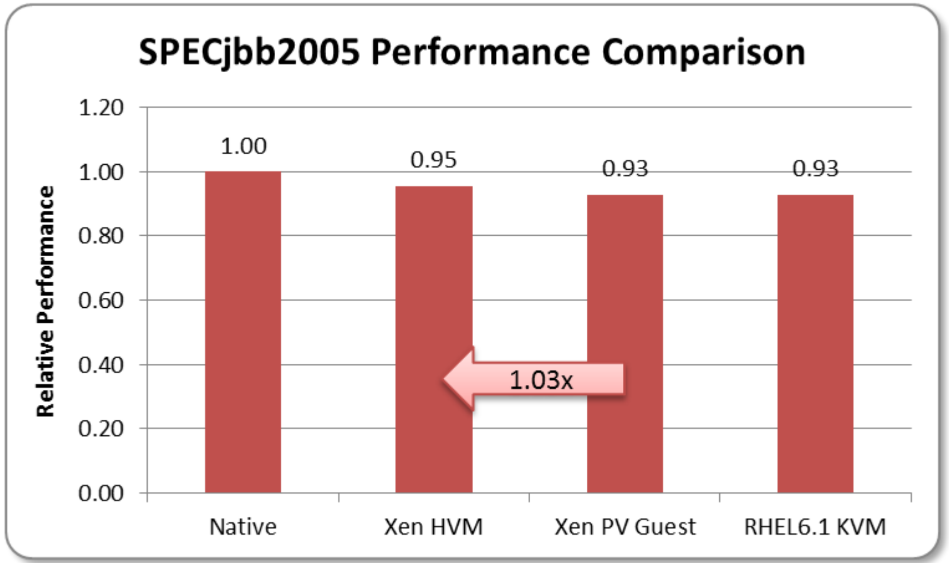
Medium VM



Source: Intel

1 VM with 4 vCPUs
floating on 4 pCPUs (2 cores w/ HT)
6GB JVM heap size

Large VM



Source: Intel

1 VM with 12 vCPUs
floating on 12 pCPUs (6 cores w/ HT)
18GB JVM heap size

- SPECjbb is a CPU and memory intensive workload.
- HVM is 3%~6% better than PV guest.

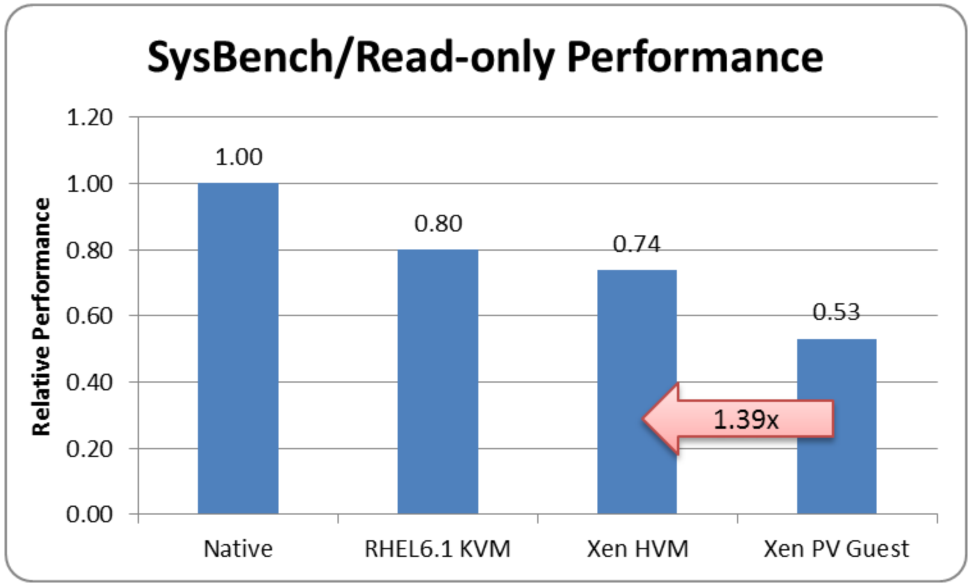
HVM vs. PV Guest – SPECjbb (2/2)

Metric	PV Guest	HVM Guest	HVM/PV
PathLength	68,473	66,408	0.97x
Guest	66,817	65,542	0.98x
VMM	1,656	866	0.52x
VMExits per second	-	12,650	
hypercalls per second	342,736	65,652	0.19x
iret	301,703	15,161	0.05x
… (the rest)	41,034	50,490	1.23x

- The major cause is that PV guest has higher Path Length (# of instructions per transaction) than HVM
  - in both guest and VMM
  - VMExits are cheaper than hypercalls in this case

Cheaper VMExits lead to less overhead in Xen

HVM vs. PV Guest – SysBench (1/2)



* Xen HVM uses PV driver

Source: Intel

HVM outperforms PV guest by 39%.

HVM vs. PV Guest – SysBench (2/2)

Metric	PV Guest	HVM	HVM/PV
Throughput (txn per sec)	1,730	2,367	1.37x
CPU %	20.9	24.2	1.16x
PathLength	4,852,609	4,821,256	0.99x
CPI	1.61	1.37	0.85x
L1 Data Cache MPKI	17.99	14.54	0.81x
L2 Cache Data MPKI	4.86	4.23	0.87x
L3 Cache Data MPKI	1.02	1.00	0.98x
DTLB Load MPKI	3.48	1.61	0.46x

There are two reasons for the higher performance of HVM:

- More CPU cycles could be utilized
- Lower CPI makes the CPU cycles be used more efficiently
  - Much lower cache misses

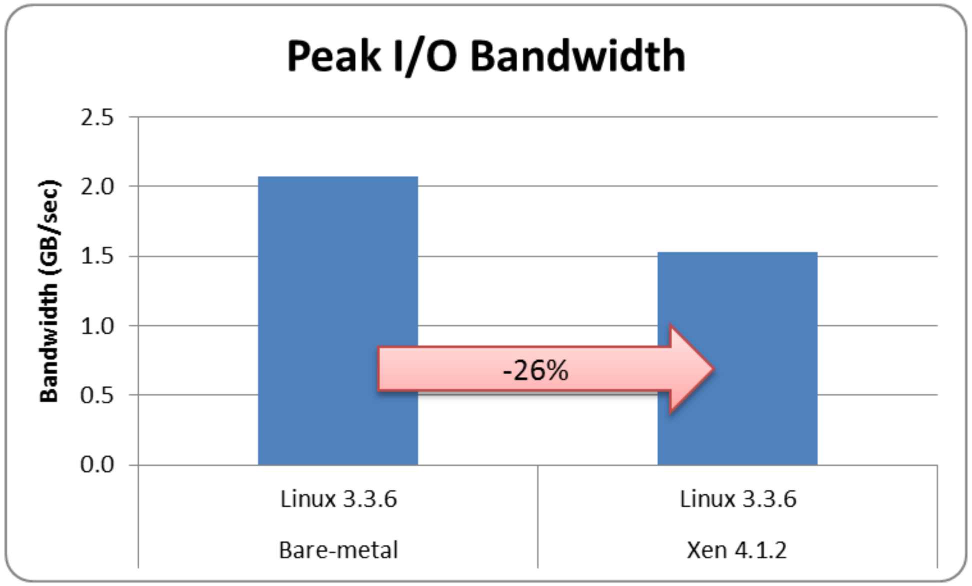
PathLength = # of instructions per transaction
CPI = Cycles per Instruction
MPKI = Misses per 1000 instructions

Better Cache Locality in HVM

Agenda

- HVM vs. PV guest comparison
- PV disk driver performance and optimizations

Big Gap between Bare-metal and PV Disk



Source: Intel

HVM+PV Disk has 26% lower bandwidth than bare-metal.

- and much higher CPU utilization.

Per-device blkif_io_lock

Multiple PV disks were used to achieve a high b/w in our test.

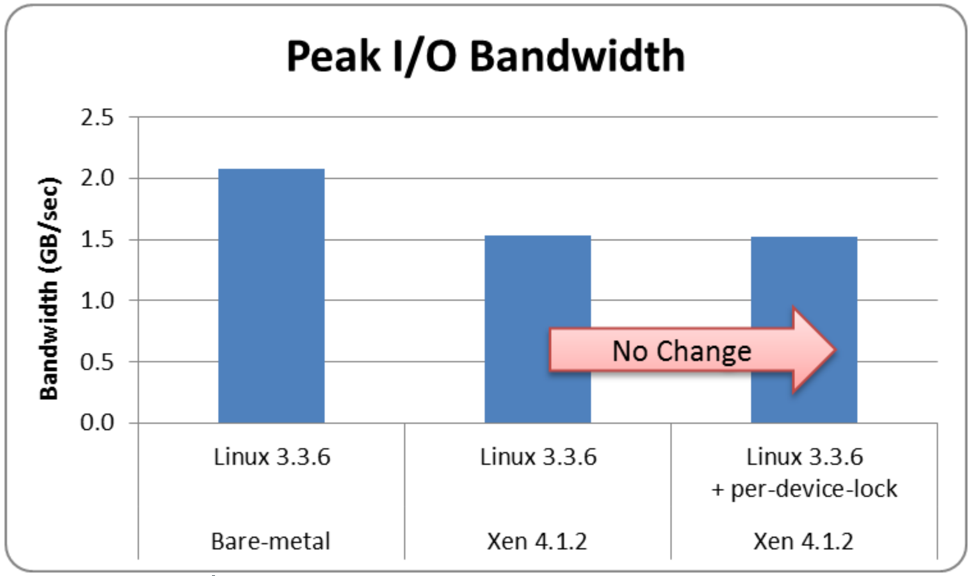
We found the **global blkif_io_lock** in the *front-end* seems to be the performance bottleneck.

- Very high CPU utilization in the guest kernel.

Luckily, this is a known issue and has been fixed already by Amazon:

- Replace the global lock by per-device lock.
- Commit (March 2012): http://git.kernel.org/?p=linux/kernel/git/axboe/linux-block.git;a=commit;h=3467811e26660eb46bc655234573d22d6876d5f9

Result of Per-device-lock



Source: Intel

However there is no performance improvement.

- Though it did reduce the overhead (CPU utilization).

I/O Merging Issue

Our further analysis showed the major issue is the I/O request cannot get adequate merging in sequential I/O case.

- A 64KB request in guest became a 44KB request plus a 20KB request in dom0.

The root cause of the problem is a pre-defined MAX limitation in the **I/O Ring** related data structures:

- `#define BLKIF_MAX_SEGMENTS_PER_REQUEST 11`
- Limits the max number of scatter-gather segments in a PV I/O request sent from front-end to back-end

Simply setting the limit to a larger value may lead to new issue:

- Too few entries accommodated by the ring, which may bring other performance issue.

Details in the next page.

BLKIF_MAX_SEGMENTS_PER_REQUEST

define a 4KB ring

```
File: linux/include/xen/interface/io/blkif.h
DEFINE_RING_TYPES(blkif, struct blkif_request, struct blkif_response);

struct blkif_request {
         uint8_t        operation;    /* BLKIF_OP_???                    */
         uint8_t        nr_segments;  /* number of segments              */
         blkif_vdev_t   handle;       /* only for read/write requests    */
         uint64_t       id;           /* private guest value, echoed in resp  */
         union {
                  struct blkif_request_rw rw;
                  struct blkif_request_discard discard;
         } u;
};

struct blkif_request_rw {
         blkif_sector_t sector_number;/* start sector idx on disk (r/w only)  */
         struct blkif_request_segment {
                  grant_ref_t gref;        /* reference to I/O buffer frame       /
                  /* @first_sect: first sector in frame to transfer (inclusive).   /
                  /* @last_sect: last sector in frame to transfer (inclusive).    /
                  uint8_t     first_sect, last_sect;
         } seg[BLKIF_MAX_SEGMENTS_PER_REQUEST];
};

#define BLKIF_MAX_SEGMENTS_PER_REQUEST 11
```

refers to

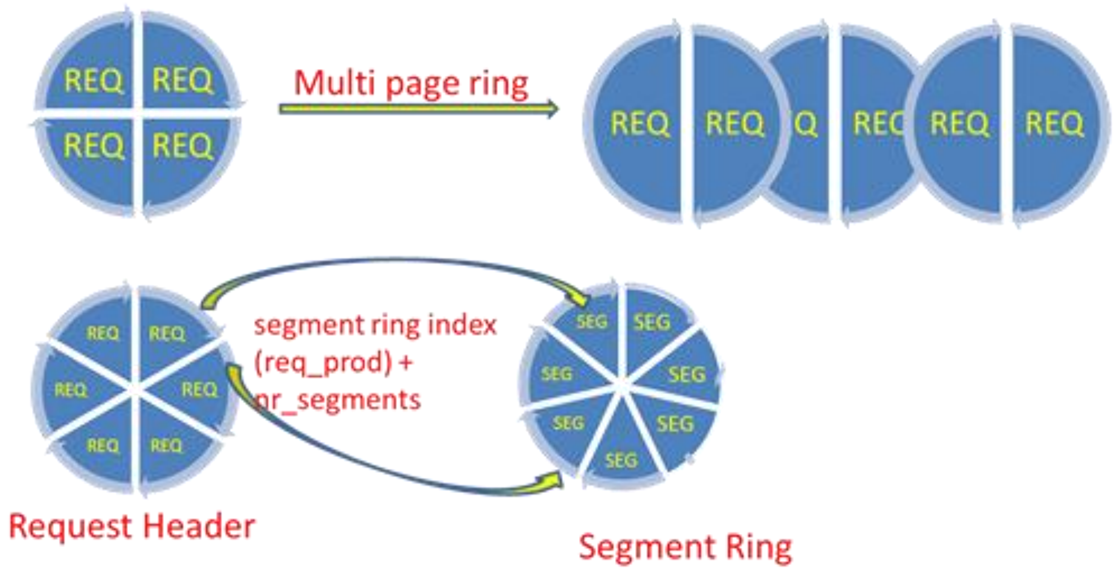
refers to

limited by

Determines how many requests a ring can hold

A segment usually be 4KB I/O block; 11 segments limits the max I/O size to be 44KB

Solution: Multi-Page-Ring



Based on Citrix’s early work.

- http://lists.xen.org/archives/html/xen-devel/2012-03/msg00388.html

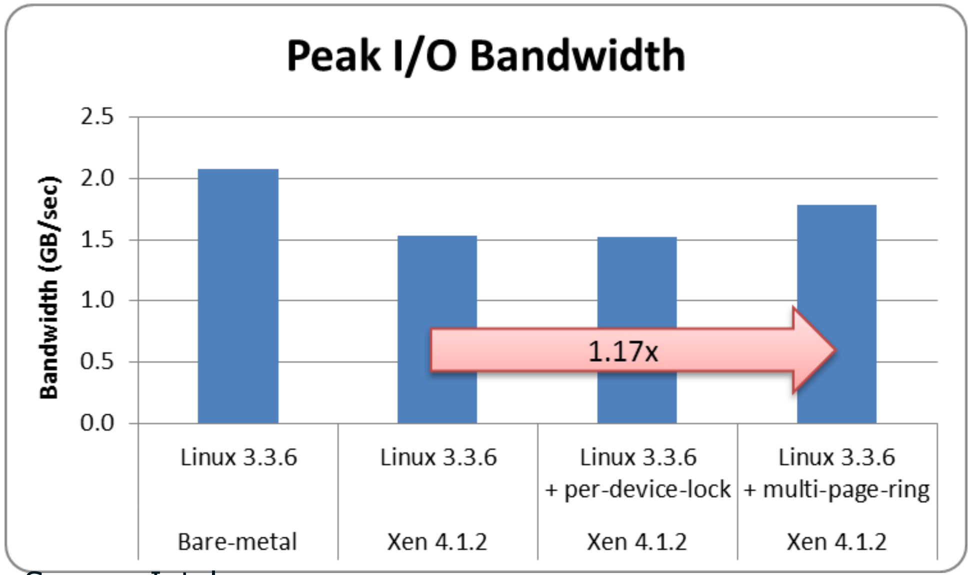
Allow to use more than one page for a ring.

- Still need to increase `BLKIF_MAX_SEGMENTS_PER_REQUEST` to support more segments in one request
- Request structure size gets increased as a result

Move `seg[BLKIF_MAX_SEGMENTS_PER_REQUEST]` out of the request structure and put them in a separate ring instead.

- More scalable

Result of Multi-page-ring

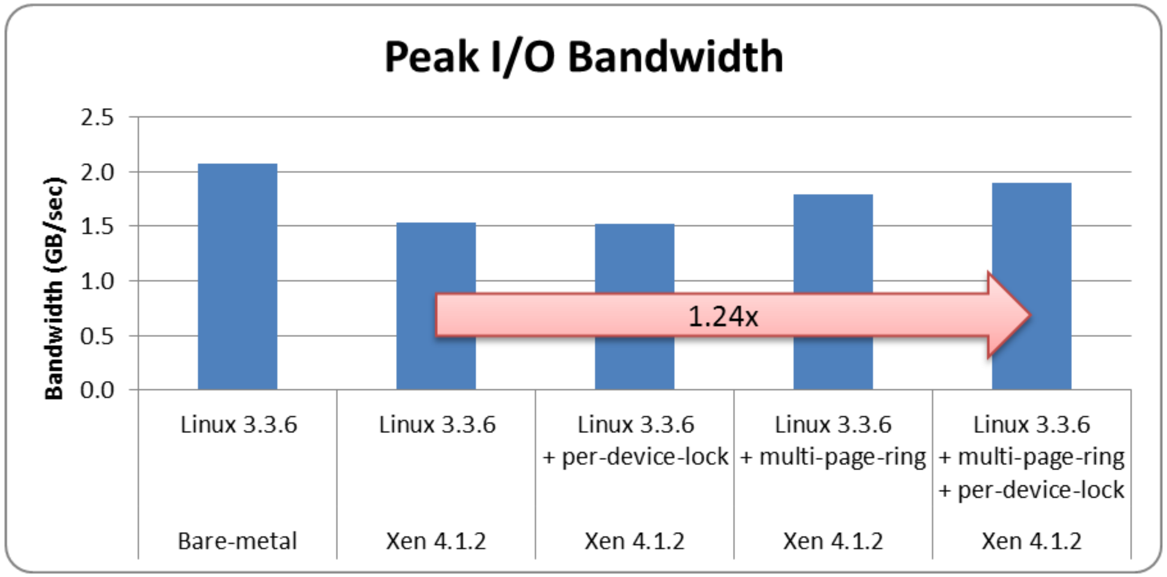


Source: Intel

Multi-page-ring improved the peak bandwidth by 17% and reduced the overhead by 29% at the same time.

- Still a big gap with the bare-metal.

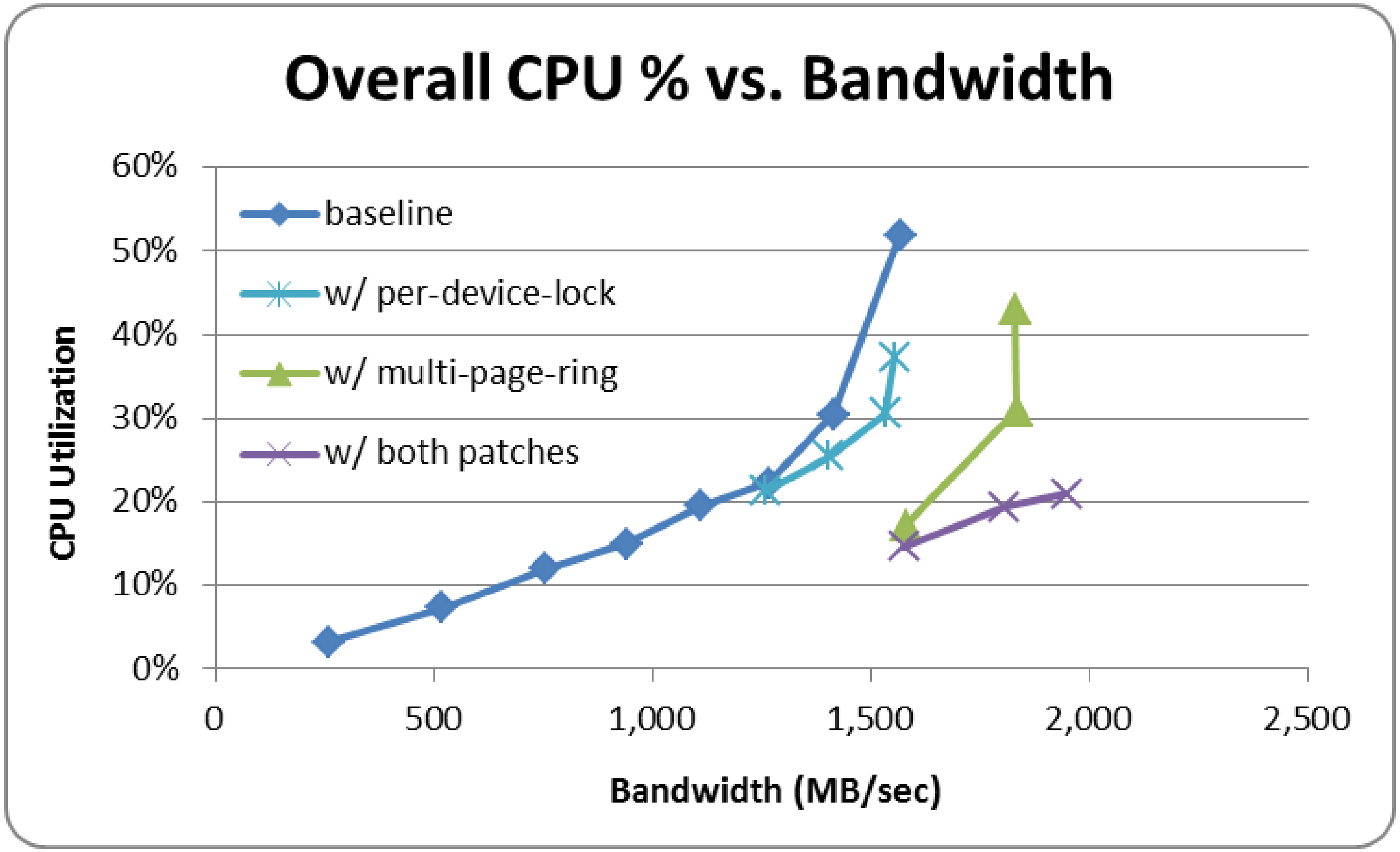
Per-device-lock Again



Source: Intel

With multi-page-ring, per-device-lock can help get even higher b/w and lower overhead.

Review the Changes



Source: Intel

Summary

HVM outperforms PV guest in both CPU/memory intensive workload and disk I/O intensive workload.

Xen PV disk performance is 26% worse than bare-metal in terms of peak bandwidth.

We proposed a multi-page-ring patch that improves the PV disk performance by 24% and reduces the gap with bare-metal to single digit.

Q&A from the Xen Summit

Q: There is another queue, the queue of the outstanding requests, did you increase the request queue?

A: Yes, the data shown in here was with increased outstanding I/O numbers which are large enough to reach peak b/w.

Q: On the PV vs. HVM work, what was the kernel that was being use for SPCjbb, and what distribution?

A: see backup pages for detailed configurations.

Q: How many guests where running?

A: single guest VM.

Q: Do you have inside into how much the hypervisor contributes to the difference between hypercall and vmexit?

A: Hypervisor should have no direct contribution to the difference. The difference is majorly from the different guest kernels.

Q: How much of these differences might be helped with APICv?

A: In this particular case (SPECjbb), APICv won't help much because SPECjbb is CPU/memory intensive and has few external interrupts. APICv may help a lot in an I/O intensive workload.

What processor was this done on?

A: see backup pages for detailed configurations.

Q: How was the data (patch lengh, CPI, cache miss rate) collected?

A: using Intel in-house tool.

Q: Was this done with ept?

A: Yes

Q: What about shadow?

A: Did not try it on shadow. Shadow and PV should have similar performance because the use the same page table format.

Q: Use huge pages with specjbb will make a big difference, because it reduces the number of page walks greatly.

A: guest hugepage was disabled to get a fair comparison between PV and HVM. See backup for configurations.

Q: Was the test run on rotating media or SSD?

A: all on SSD.

BACKUP

Configurations - SPECjbb

- H/W: 2-socket Intel® Xeon® Processor X5660 with 48GB memory (24GB on each node) running at 800MHz.
  - EIST/Turbo disabled; core c3/c6 disabled.
- VMM: Xen 4.1.0
  - Dom0: Linux 2.6.32 with 2 vCPUs and 4GB memory
- Guest O/S: RHEL 6.1
  - THP is disabled for a fair comparison, because PV guest does not support huge page.
    - Implies 4KB guest page since we did not reserve huge page explicitly.
  - With RHEL6.1's PVOPS kernel, PV disk and PV network are used for HVM
    - It does not matter much for this particular workload (SPECjbb has almost zero disk/network activities)
- VM Configurations:
  - Case 1: 4 vCPUs + 8GB memory
    - pinned to 4 pCPUs as a group (floating)
    - the 4 pCPUs share 2 cores (HT enabled) in socket 0
  - Case 2: 12 vCPUs + 19GB memory
    - pinned to 12 pCPUs as a group (floating)
    - the 12 pCPUs share the socket 0 (HT enabled)
- Workload Configurations:
  - jdk1.6.0_27/bin/java -server –Xmx<N>g –Xms<N>g -XX:+UseParallelOldGC
    - Case 1: 6GB heap size; Case 2: 18GB heap size
- Workload methodology:
  - Case 1: 3 warehouses for ramp-up and 4 warehouses for steady state.
  - Case 2: 11 warehouses for ramp-up and 12 warehouses for steady state.
- 10 minutes ramp-up, and 15 minutes steady state.
  - Used 60 minutes steady state for data collection.

Configurations - SysBench

H/W: 2-socket Intel® Xeon® Processor E5-2680 with 32GB DDR3 memory running at 1333MHz. 1x Intel SSD for MySQL data.

VMM: Xen 4.1.2

- Dom0: Linux 3.2.9

Guest O/S: RHEL 6.2

- With RHEL6.2's PVOPS kernel, PV disk and PV network were used in both HVM mode and PV guest mode.

VM Configurations:

- #vCPUs = #pCPUs; 4GB memory
- The data disk was assigned to VM in 'PHY:' mode.

Workload Configurations:

- SysBench: prepared 40 million rows ~9.2GB data
- MySQL: set innodb buffer pool size to 2GB and use O_DIRECT as flush mode.
- SysBench Read-only test: sysbench --mysql-db=sbtest --max-requests=0 --test=oltp --mysql-table-engine=innodb --oltp-table-size=40000000 --mysql-user=root --mysql-host=localhost --mysql-socket=/var/lib/mysql/mysql.sock --mysql-password=123456 --num-threads=$vCPU_num --max-time=300 --oltp-read-only=on run

Configurations – PV Disk

Hardware:

- 2-socket Intel® Xeon® Processor X5660 with 48GB memory
- HBA: LSI 9205
- Storage: 2 disk bays with 4 Intel SSD disks in each.

Software:

- Hypervisor: Xen 4.1.2
- Dom0: Linux 3.3.6 (8 vCPUs pinned to 8 pCPUs)
- Guest O/S: Linux 3.3.6 (8 vCPUs pinned to 8 pCPUs; 2GB memory)
  - Virtual disks are configured as ‘PHY:’, i.e., no caching in dom0
- Workload: aio-stress
  - Accessing disks in O_DIRECT mode, i.e, no caching in guest.
  - Using 64KB sequential I/O pattern

intel®