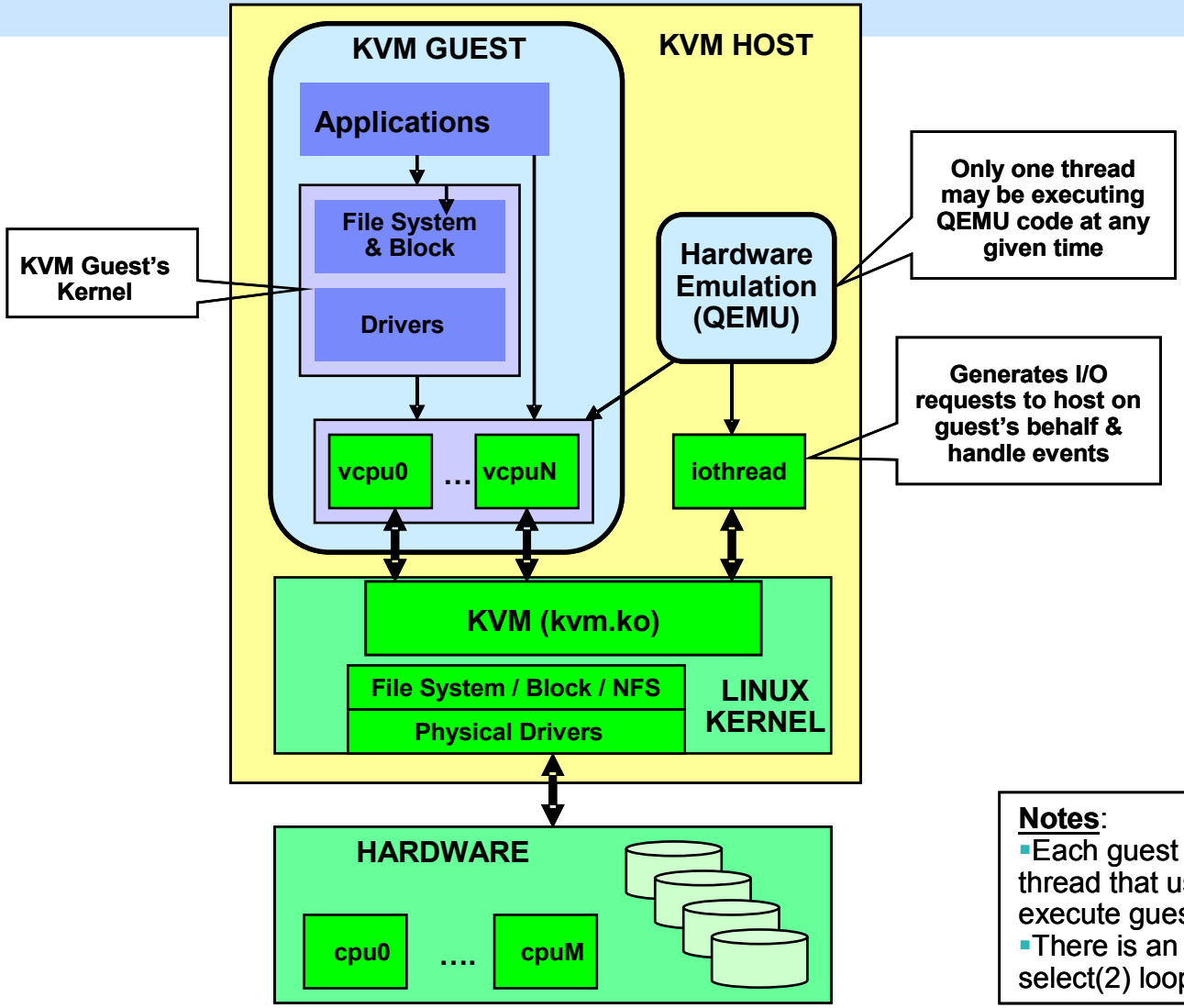
Exploiting The Latest KVM Features For Optimized Virtualized Enterprise Storage Performance

Dr. Khoa Huynh (khoa@us.ibm.com)
IBM Linux Technology Center

Agenda

- Overview – KVM I/O architecture
  - Key performance challenges for storage I/O in virtualized environments
- Solving key performance challenges
  - Solutions & prototypes
  - Performance results
  - Performance recommendations
- Other recent performance features
- Recap

KVM I/O Architecture … at 10,000 Feet



- Applications and guest OS run inside **KVM Guest** just like they run bare metal
- Guest interfaces with emulated hardware presented by **QEMU**
- **QEMU** submits I/Os to host on behalf of guest
- **Host kernel** treats guest I/Os like any user-space application

Notes:
- Each guest CPU has a dedicated vcpu thread that uses kvm.ko module to execute guest code
- There is an I/O thread that runs a select(2) loop to handle events

So … What Are The Key Performance Challenges?

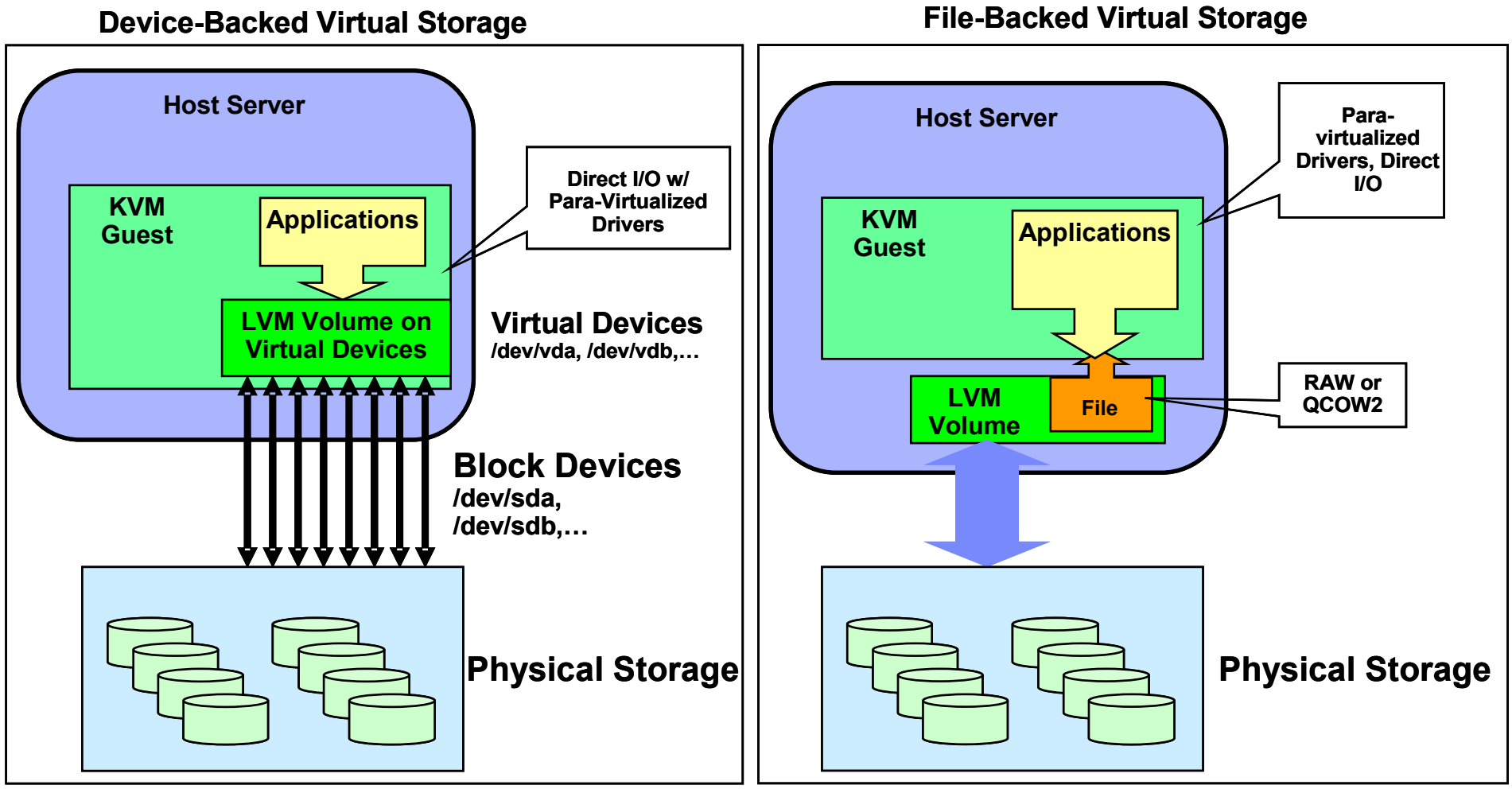
- **Low throughput / high latencies (compared to bare metal)**
  - Generally less than 30% of bare metal → configuration issue(s)
  - Between 30% and 60% of bare metal → performance tuning
  - With proper configuration + performance tuning → 90% or more
- **Low I/O rates (IOPS)**
  - Some enterprise workloads require 100Ks I/Os Per Second (IOPS)
  - Most difficult challenge for I/O virtualization!
  - Current KVM tops out at ~147,000 IOPS
  - VMware claimed vSphere v5.1 could achieve 1.06 million IOPS for a single guest

Issue – Low Throughput / High Latencies

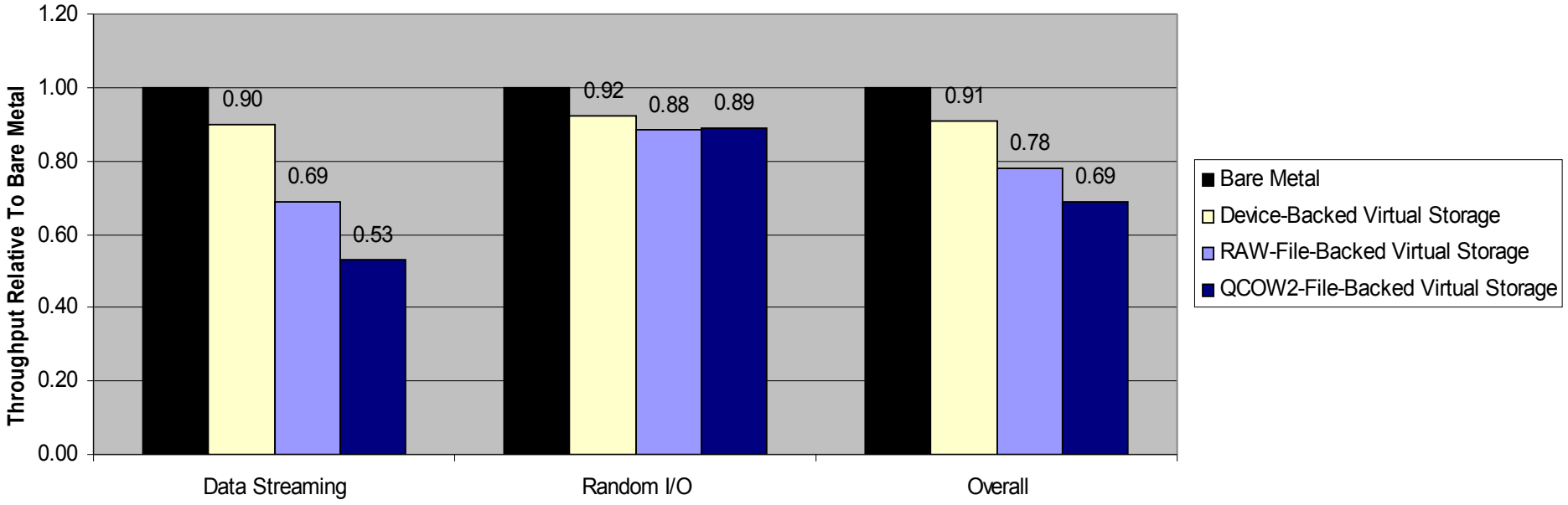
I/O Virtualization Approaches

- **Device assignment** (pass-through)
  - Pass physical device directly to guest
  - High performance
  - No device sharing among multiple guests
  - Difficult for live migration
  - PCI device limit (8) per guest
- **Full virtualization** – IDE, SATA, SCSI
  - Good guest compatibility
  - Bad performance (many trap-and-emulate operations), does not scale beyond 1 thread ← ***Not recommended for enterprise storage***
- **Para-virtualization** – virtio-blk, virtio-scsi
  - Efficient guest ↔ host communication through virtio ring buffer (virtqueue)
  - Good performance
  - Virtualization benefits (e.g. device sharing among guests, etc.)

Virtio-blk Storage Configurations



Storage Performance Results



Notes:

- FFSB = Flexible File System Benchmark (http://sourceforge.net/projects/ffsb/)
- Data Streaming (sequential reads, sequential writes) with block sizes of 8KB and 256KB
- Random Mixed Workloads = random reads, random writes, mail server, mixed DB2 workloads (8KB block size)

Storage Performance Recommendations

Proper Configuration & Performance Tuning make BIG difference!

- Use VirtIO drivers (IDE emulation does NOT scale beyond 1 thread)
- Virtual Storage
  - If possible, pass whole partitions or block devices to guest (device-backed virtual disks) instead of host files
  - PCI pass-through is better than device-backed virtual disks, but cannot easily migrate
- Virtual Image Format
  - RAW offers the best performance, but QCOW2 performance has **improved significantly** since RHEL 6.1 and upstream
- Guest Caching Mode
  - cache = writethough is the default & provides data integrity in all cases (disk cache not exposed to guest)
  - cache = writethrough is great for read-intensive workloads (host's page cache enabled)
  - cache = none is great for write-intensive workloads or workloads involving NFS remote storage (host's page cache disable and disk write cache enabled)
- Guest I/O Model
  - Linux AIO support (aio=native) provides better performance in many cases, especially with multiple threads
- I/O Scheduler
  - Deadline I/O scheduler yields the best performance for I/O-intensive workloads (much better than the default CFQ scheduler)
- Miscellaneous
  - Enable x2APIC support for guest → 2% to 5% performance gain for I/O-intensive workloads
  - Disable unnecessary features: delay accounting (nodelayacct kernel boot parameter), random entropy contribution (sysfs)
  - Avoid memory over-commit in KVM host (much worse than CPU over-commit)

Storage Performance Recommendations (cont'd)

- KVM Guest Configuration:
  - Use **2 or more virtual CPUs** in the guest for I/O-intensive workloads
  - Allocate enough memory in the guest to avoid memory overcommit
  - Do **not** format virtual disks with ext4 on RHEL5
    - RHEL5 does not have performance optimizations for ext4
  - Use **deadline I/O scheduler** in the guest
  - Use Time Stamp Counter (TSC) as clocksource (if host CPU supports TSC)
  - 64-bit guest on 64-bit host has the best I/O performance in most cases
- NFS Remote Storage
  - If RHEL5.5 (or earlier version) is used on KVM host, need ***kvm-83-164.el5_5.15*** or later
    - Reduce number of *lseek* operations → up to 3X improvement in NFS throughput and 40% reduction in guest's virtual CPU usage
    - Patches are included in RHEL5.6 and later (and upstream QEMU)
  - If RHEL6.0 or 6.1 is used on KVM host, need **an errata package** for RHEL6.1 (http://rhn.redhat.com/errata/RHBA-2011-1086.html)
    - RHEL6 QEMU uses vector I/O constructs for handling guest I/O's, but NFS client (host) does not support vector I/O, so large guest I/O's are broken into 4KB requests
    - This issue was fixed in RHEL6.2 (NFS client now supports vector I/O)
  - Use Deadline I/O scheduler
  - Ensure NFS read / write sizes are sufficiently large, especially for large I/O's
  - Ensure NFS server(s) have enough NFS threads to handle multi-threaded workloads
  - NFS exports = sync (default), w_nodelay
  - For distributed parallel file-system with large block sizes used in cloud's storage systems, need ability to avoid reading/writing multiple blocks for each I/O that is smaller than block size

Issue – Low I/O Rates (IOPS)

Some Background

- Many enterprise workloads (e.g. databases, ERP systems, low-latency financial trading applications, etc.) require very high I/O rates
  - Some demand well over 500,000 I/Os Per Second (IOPS)
- KVM can typically handle only up to ~147,000 IOPS
  - Very high I/O rates present significant challenge to virtualization
  - *Primary reason why many enterprise workloads have not been migrated to virtualized environments*
- Challenge from other hypervisors
  - **(2011) VMware vSphere 5.0** – 300,000 IOPS for single VM; 1 million IOPS for 6 VMs @ 8KB I/O size
  - **(2012) VMware vSphere 5.1** – 1 Million IOPS for a single VM @ 4KB I/O size
  - **(2012) Microsoft Hyper-V** – 700,000 IOPS for a single host @ 512-byte I/O size

Possible Paths To Very High I/O Rates (1 million IOPS)

- **PCI Pass-through (PCI Device Assignment)**
  - Currently stands at ~800,000 IOPS @ 8KB I/O size, ~950,000 IOPS @ 4KB I/O size
  - Limit of 8 PCI devices per VM (guest)
  - No virtualization benefits; difficult for live migration
- **Virtio-blk Optimization**
  - Virtio-blk can traditionally support up to ~147,000 I/O Operations Per Second (IOPS)
  - Performance profiling → ***Big QEMU Lock***
    - Allows core QEMU components to ignore multi-threading (historically)
    - Creates scalability problems
  - ***Need to bypass or relieve Big QEMU Lock as much as possible***

Bypassing Big QEMU Lock in virtio-blk

- **Vhost-blk**
  - Initially coded by Liu Yuan, new prototype by Asias He
  - Submits guest I/Os directly to host via *kernel* threads (similar to vhost_net for network)
  - *Drawbacks*:
    - Involves the kernel (ring 0 privilege, etc.)
    - Cannot take advantage of QEMU features (e.g. image formats, etc.)
    - Cannot support live migration
- **"Data-Plane" QEMU**
  - Coded by Stefan Hajnoczi (~1500 LOC)
  - Submits guest I/Os directly to host in *user space* (one user-space thread per virtual block device)
  - Will become *default* mode of operations – eventually
  - No kernel change is required

Both approaches have comparable performance in our testing!

“Data-Plane”

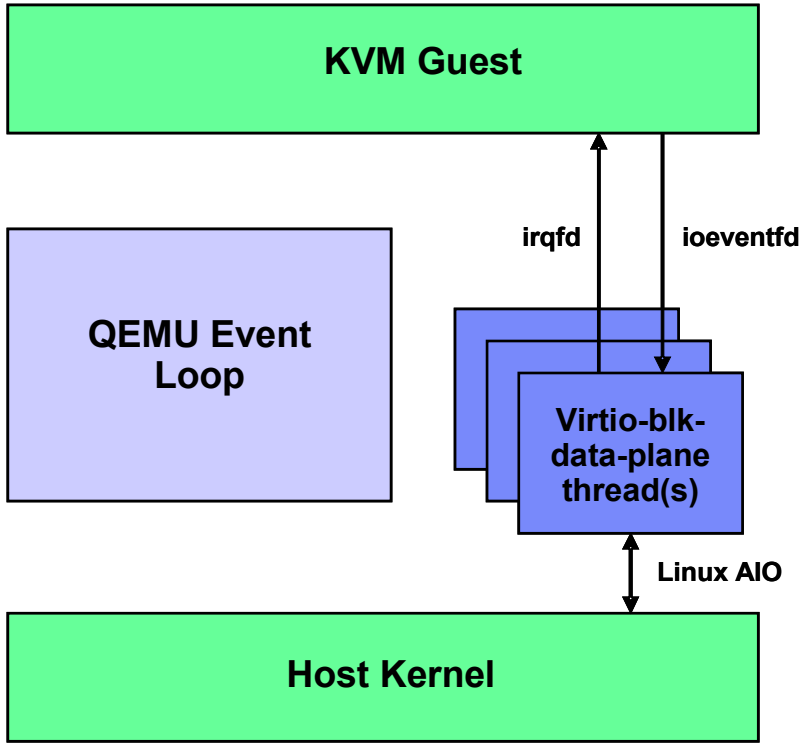
“Most exciting development in QEMU in the last 5 years!”
- Anthony Liguori (QEMU Maintainer)

Virtio-blk-data-plane:

- Accelerated data path for para-virtualized block I/O driver
- Using per-device dedicated threads and Linux AIO support in the host kernel for I/O processing – without going through QEMU block layer
  - *No need to acquire big QEMU lock*

Availability

- Accepted upstream (*qemu-1.4.0* or later)
- Available as Technology Preview in ***Red Hat Enterprise Linux 6.4*** and ***SUSE Linux Enterprise Server 11 SP3***



How To Enable Virtio-blk-data-plane?

Libvirt Domain XML

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
...
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw' cache='none' io='native'/>
      <source file='path/to/disk.img'/>
      <target dev='vda' bus='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
    </disk>
...
  <qemu:commandline>
    <qemu:arg value='-set'/>
    <qemu:arg value='device.virtio-disk0.scsi=off'/>
  </qemu:commandline>
  <!-- config-wce=off is not needed in RHEL 6.4 -->
  <qemu:commandline>
    <qemu:arg value='-set'/>
    <qemu:arg value='device.virtio-disk0.config-wce=off'/>
  </qemu:commandline>
  <qemu:commandline>
    <qemu:arg value='-set'/>
    <qemu:arg value='device.virtio-disk0.x-data-plane=on'/>
  </qemu:commandline>
<domain>
```

QEMU command-line

```
qemu -drive if=none,id=drive0,cache=none,aio=native,format=raw,file=path/to/disk.img \
     -device virtio-blk,drive=drive0,scsi=off,config-wce=off,x-data-plane=on
```

BUT ....

- **Current limitations**
  - Only raw image format is supported
    - Other image formats depend on QEMU block layer
  - Live migration is not supported
  - Hot unplug and block jobs are not supported
  - I/O throttling limits are ignored
  - Only Linux hosts are supported (due to Linux AIO usage)
- **On-going work (for upcoming QEMU releases)**
  - Patches have recently been submitted upstream to convert *virtio-net* to use “data-plane”
  - Reduce the scope of big QEMU lock → moving to RCU (Read Copy Update)

Performance Results – “Data-Plane”

- *Highest virtualized storage I/O rates ever reported for a single virtual machine (guest)*
- Red Hat Enterprise Linux 6.4
  - 1.20 million IOPS @ 8KB I/O size for a single guest
  - 1.58 million IOPS @ 4KB I/O size for a single guest
- Upstream QEMU / SLES 11 SP3
  - 1.37 million IOPS @ 8KB I/O size for a single guest
  - 1.61 million IOPS @ 4KB I/O size for a single guest
  - More efficient memory-mapping infrastructure (in QEMU)
  - Approaching bare-metal limit of our storage setup
- *50% higher than the closest competing hypervisor (VMware vSphere 5.1)*
- Low latencies and consistent throughput for storage I/O requests

First Thing First

- Needed a storage setup capable of delivering > 1 Million IOPS

Host Server: **IBM® System x3850 X5**

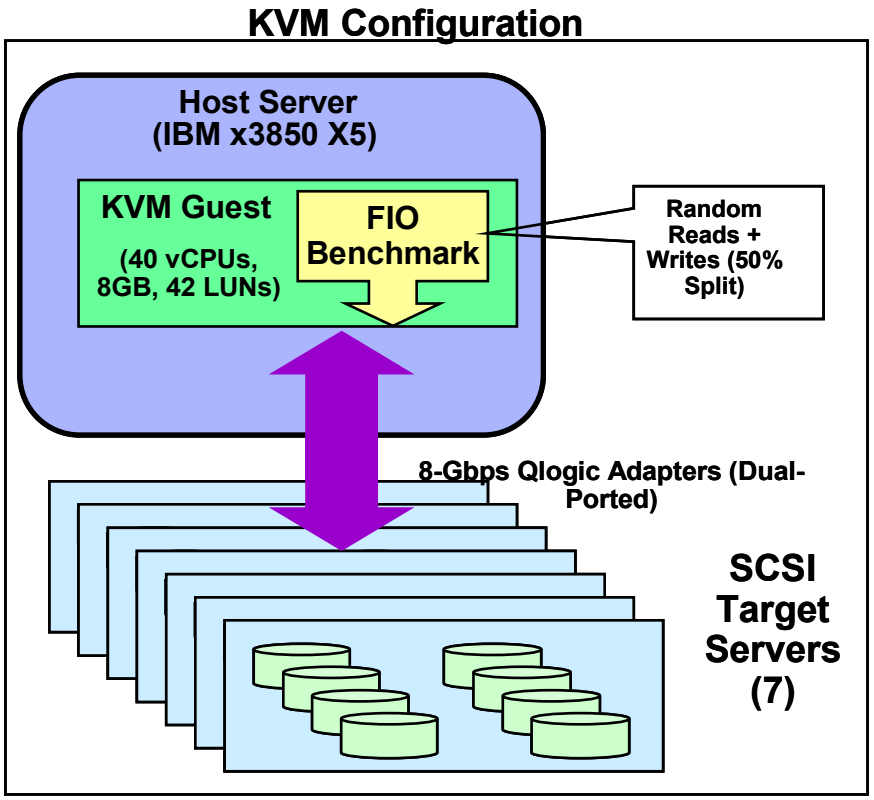
- 4 x E7-4870 sockets (40 Cores @ 2.40 GHz), 256GB Memory (Total)

Storage:

- 7 x 8-Gbps, dual-ported QLogic® HBA's hooked to 7 SCSI target servers
- Each SCSI target server is backed by RAM disks and provides 2 PCI devices (ports) and 8 LUNs (56 LUNs total)

Virtual Machine (Guest):

- 40 vCPUs, 8 GB memory, 42 LUNs
- No host cache (cache=none)
- FIO workload:
  - 1 job per LUN
  - Queue depth = 32
  - Direct I/O's
  - Engine = libaio



Qemu-kvm Command Line For “Data-Plane” Testing

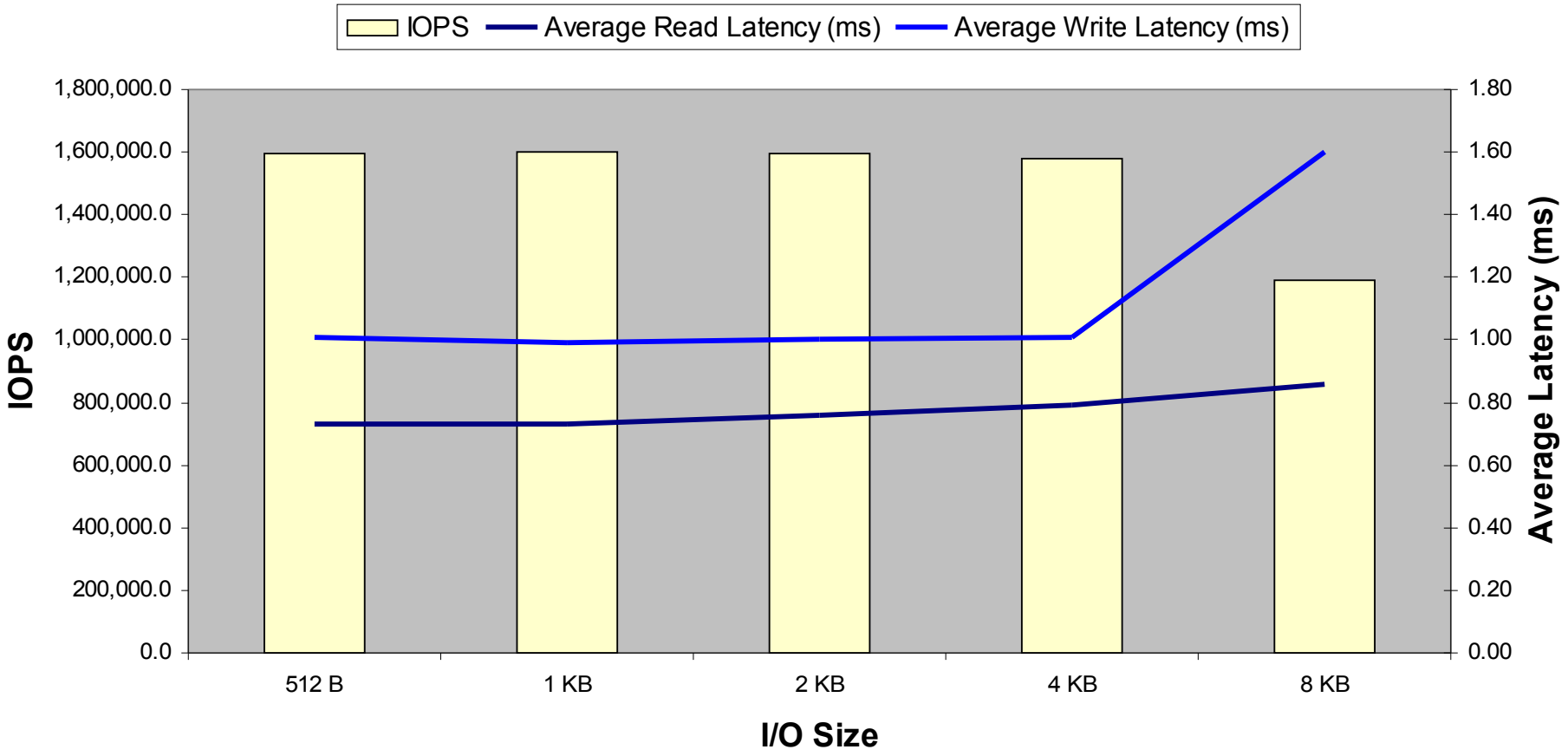
```
qemu-kvm -M pc -mem-path /hugepages -m 7168 -cpu qemu64,+x2apic -smp 40 -name guest1 -uuid 1c047c62-c21a-1530-33bf-185bc15261d8 -boot c
-drive if=none,id=root,file=/khoa/images/if1n1_sles11.img -device virtio-blk-pci,drive=root -drive if=none,id=drive-virtio-
disk1_0,file=/dev/sdc,cache=none,aio=native -device virtio-blk-pci,scsi=off,config-wce=off,x-data-
plane=on,addr=5.0,multifunction=on,drive=drive-virtio-disk1_0,id=virtio-disk1_0 -drive if=none,id=drive-virtio-
disk1_1,file=/dev/sdd,cache=none,aio=native -device virtio-blk-pci,scsi=off,config-wce=off,x-data-
plane=on,addr=5.1,multifunction=on,drive=drive-virtio-disk1_1,id=virtio-disk1_1 -drive if=none,id=drive-virtio-
disk1_2,file=/dev/sde,cache=none,aio=native -device virtio-blk-pci,scsi=off,config-wce=off,x-data-
plane=on,addr=5.2,multifunction=on,drive=drive-virtio-disk1_2,id=virtio-disk1_2 -drive if=none,id=drive-virtio-
disk2_0,file=/dev/sdg,cache=none,aio=native -device virtio-blk-pci,scsi=off,config-wce=off,x-data-
plane=on,addr=6.0,multifunction=on,drive=drive-virtio-disk2_0,id=virtio-disk2_0 -drive if=none,id=drive-virtio-
disk2_1,file=/dev/sdh,cache=none,aio=native -device virtio-blk-pci,scsi=off,config-wce=off,x-data-
plane=on,addr=6.1,multifunction=on,drive=drive-virtio-disk2_1,id=virtio-disk2_1 -drive if=none,id=drive-virtio-
disk2_2,file=/dev/sdi,cache=none,aio=native -device virtio-blk-pci,scsi=off,config-wce=off,x-data-
plane=on,addr=6.2,multifunction=on,drive=drive-virtio-disk2_2,id=virtio-disk2_2 -drive if=none,id=drive-virtio-
disk3_0,file=/dev/sdk,cache=none,aio=native -device virtio-blk-pci,scsi=off,config-wce=off,x-data-
plane=on,addr=7.0,multifunction=on,drive=drive-virtio-disk3_0,id=virtio-disk3_0 -drive if=none,id=drive-virtio-
disk3_1,file=/dev/sdl,cache=none,aio=native -device virtio-blk-pci,scsi=off,config-wce=off,x-data-
plane=on,addr=7.1,multifunction=on,drive=drive-virtio-disk3_1,id=virtio-disk3_1 -drive if=none,id=drive-virtio-
disk3_2,file=/dev/sdm,cache=none,aio=native -device virtio-blk-pci,scsi=off,config-wce=off,x-data-
plane=on,addr=7.2,multifunction=on,drive=drive-virtio-disk3_2,id=virtio-disk3_2 -drive if=none,id=drive-virtio-
disk4_0,file=/dev/sdo,cache=none,aio=native -device virtio-blk-pci,scsi=off,config-wce=off,x-data-
plane=on,addr=8.0,multifunction=on,drive=drive-virtio-disk4_0,id=virtio-disk4_0 -drive if=none,id=drive-virtio-
disk4_1,file=/dev/sdp,cache=none,aio=native -device virtio-blk-pci,scsi=off,config-wce=off,x-data-
plane=on,addr=8.1,multifunction=on,drive=drive-virtio-disk4_1,id=virtio-disk4_1 -drive if=none,id=drive-virtio-
disk4_2,file=/dev/sdq,cache=none,aio=native -device virtio-blk-pci,scsi=off,config-wce=off,x-data-
plane=on,addr=8.2,multifunction=on,drive=drive-virtio-disk4_2,id=virtio-disk4_2 -drive if=none,id=drive-virtio-
disk5_0,file=/dev/sds,cache=none,aio=native -device virtio-blk-pci,scsi=off,config-wce=off,x-data-
plane=on,addr=9.0,multifunction=on,drive=drive-virtio-disk5_0,id=virtio-disk5_0 -drive if=none,id=drive-virtio-
disk5_1,file=/dev/sdt,cache=none,aio=native -device virtio-blk-pci,scsi=off,config-wce=off,x-data-
plane=on,addr=9.1,multifunction=on,drive=drive-virtio-disk5_1,id=virtio-disk5_1 -drive if=none,id=drive-virtio-
disk5_2,file=/dev/sdu,cache=none,aio=native -device virtio-blk-pci,scsi=off,config-wce=off,x-data-
plane=on,addr=9.2,multifunction=on,drive=drive-virtio-disk5_2,id=virtio-disk5_2 -drive if=none,id=drive-virtio-
disk6_0,file=/dev/sdw,cache=none,aio=native -device virtio-blk-pci,scsi=off,config-wce=off,x-data-
plane=on,addr=10.0,multifunction=on,drive=drive-virtio-disk6_0,id=virtio-disk6_0 -drive if=none,id=drive-virtio-
disk6_1,file=/dev/sdx,cache=none,aio=native -device virtio-blk-pci,scsi=off,config-wce=off,x-data-
plane=on,addr=10.1,multifunction=on,drive=drive-virtio-disk6_1,id=virtio-disk6_1 -drive if=none,id=drive-virtio-
disk6_2,file=/dev/sdy,cache=none,aio=native -device virtio-blk-pci,scsi=off,config-wce=off,x-data-
plane=on,addr=10.2,multifunction=on,drive=drive-virtio-disk6_2,id=virtio-disk6_2 -drive if=none,id=drive-virtio-
disk7_0,file=/dev/sdaa,cache=none,aio=native -device virtio-blk-pci,scsi
```

Single KVM Guest

RHEL 6.4 with virtio-blk-data-plane

FIO Benchmark, Direct Random I/Os (50% Reads, 50% Writes)

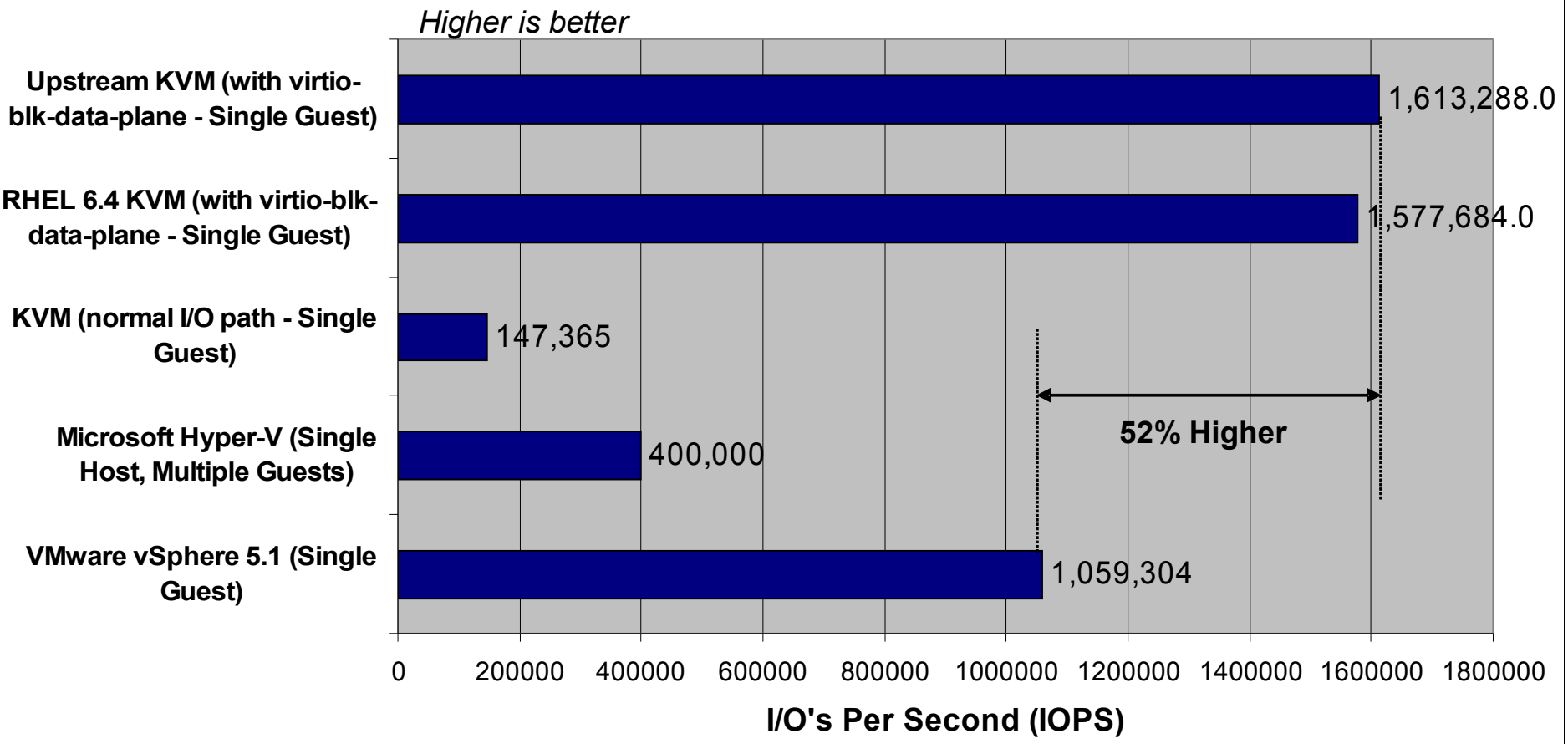
KVM Host = IBM x3850 X5 (Intel E7-8870@2.4GHz, 40 Cores, 256GB)



KVM vs. Competing Hypervisors

Direct Random I/Os at 4KB Block Size

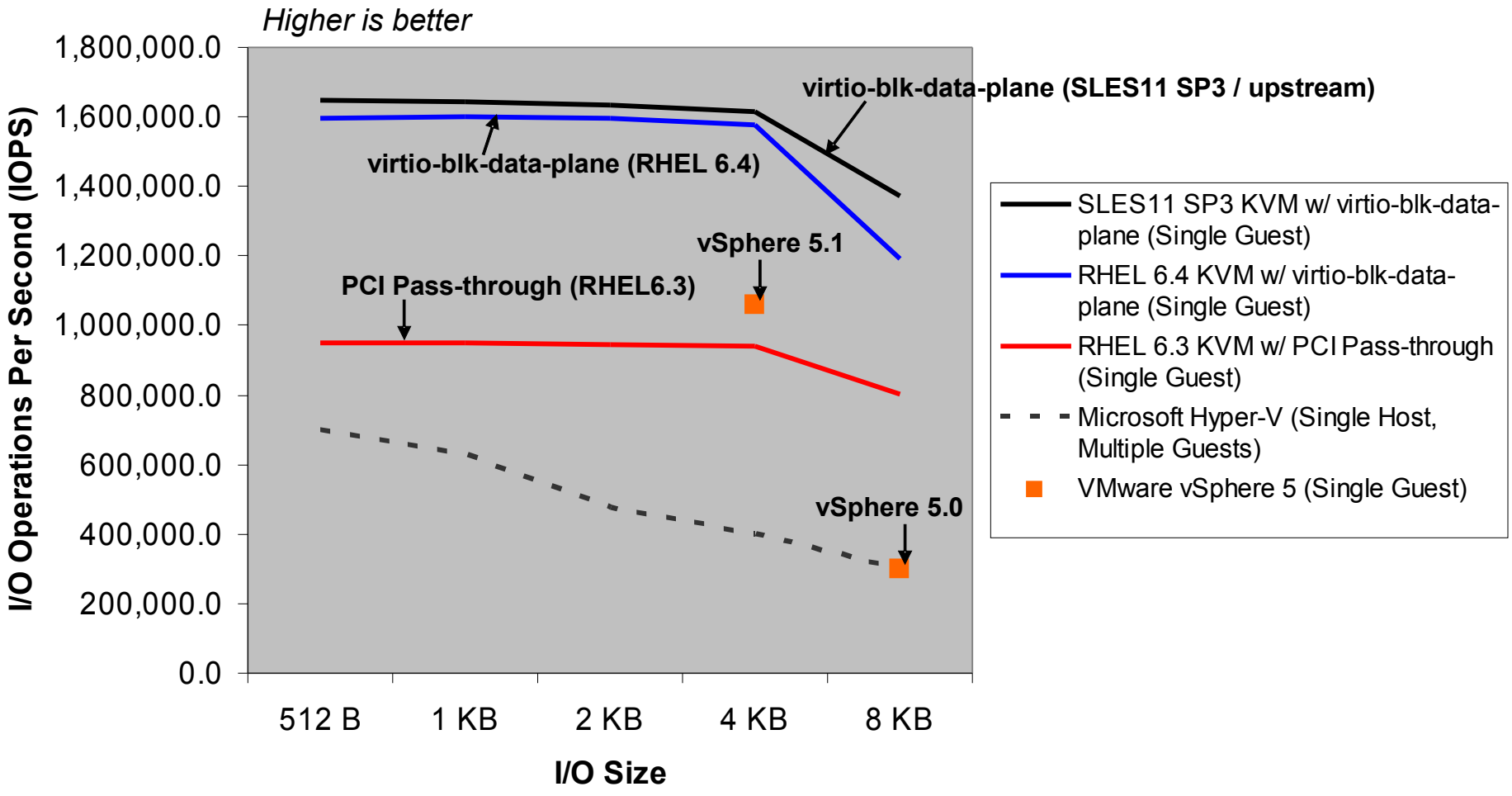
Host Server = Intel E7-8870@2.4GHz, 40 Cores, 256GB



KVM vs. Competing Hypervisors

Direct Random I/Os Across Various Block Sizes

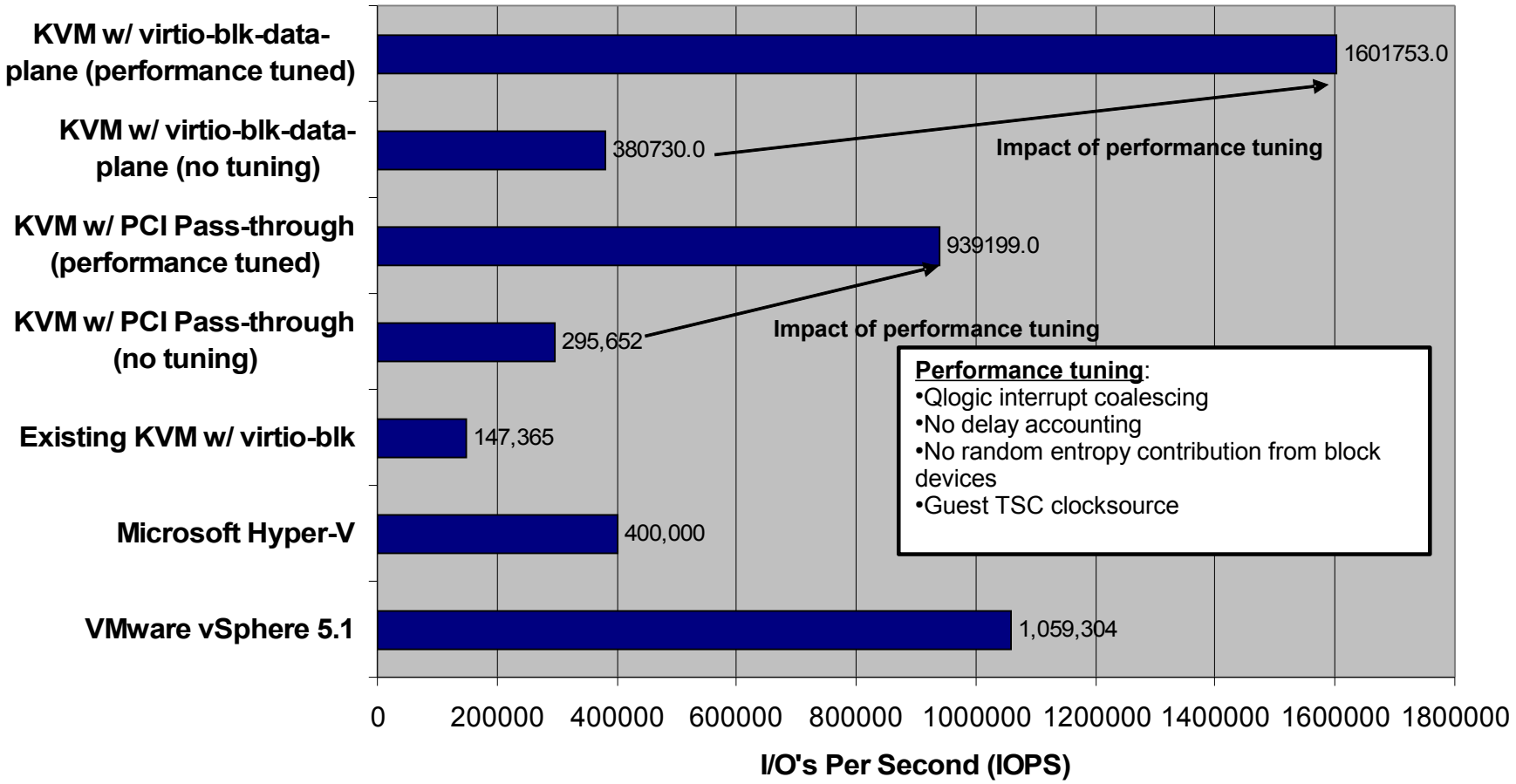
Host Server = Intel E7-8870@2.4GHz, 40 Cores, 256GB



Single Virtual Machine

Direct Random I/Os at 4KB Block Size

Host Server = Intel E7-8870@2.4GHz, 40 Cores, 256GB



Other Recent Performance Features

Para-Virtualized End-Of-Interrupts (PV-EOI)

- **Improve interrupt processing overhead**
  - Reduce number of context switches between KVM hypervisor and guests
    - Less CPU utilization
      - Up to 10%
    - Ideal for workloads with high I/O rates
      - High storage I/O rates
      - High incoming network traffic
    - Enabled by default in guest Linux operating systems
    - Availability
      - Red Hat Enterprise Linux 6.4 (KVM guests)

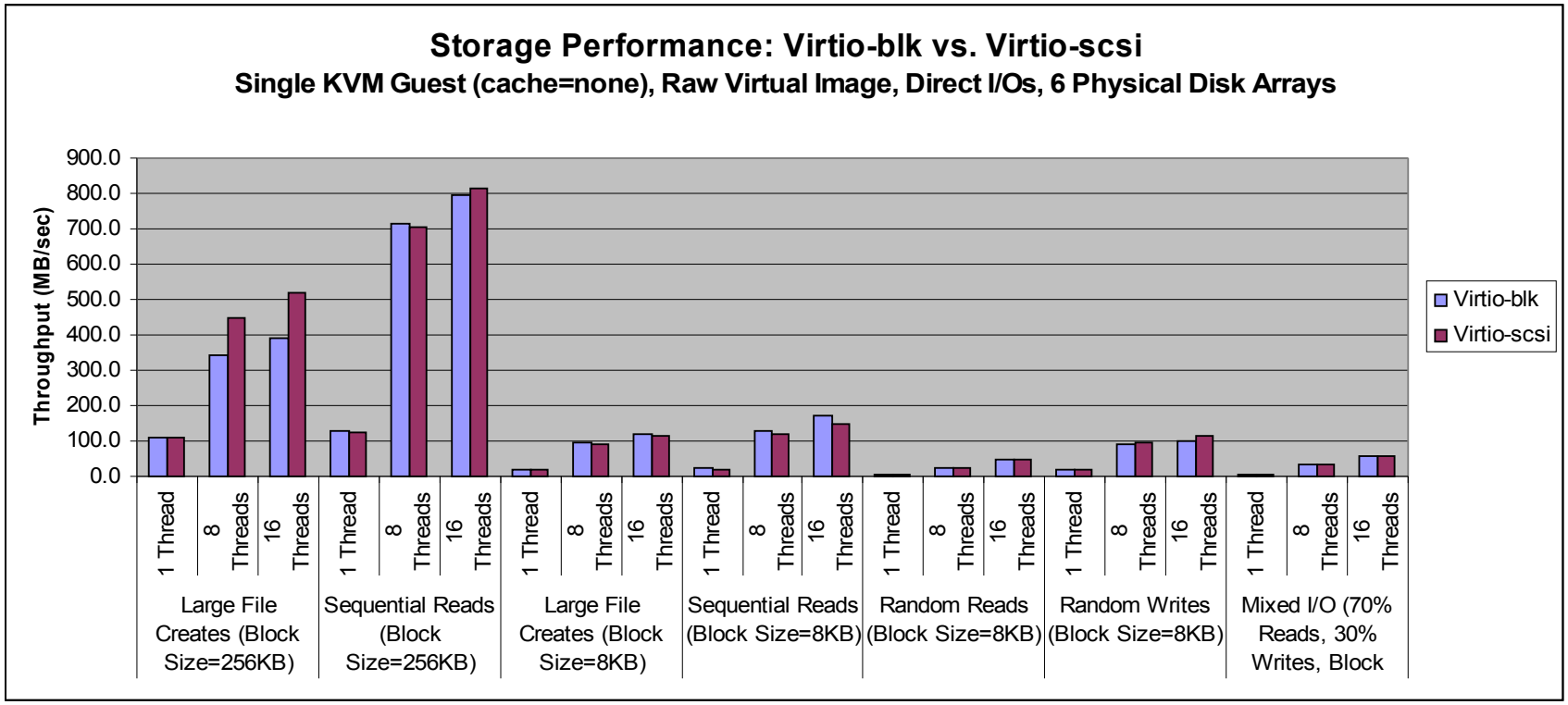
Bio-based Virtio-blk

- **Bio-based virtio-blk driver**
  - Skip I/O scheduler in KVM guest by adding bio-based path to virtio-blk (Asias He @ Red Hat)
    - Similar to bio-based driver for ramdisk
    - Shorter I/O path
    - Less global mutex contention
    - Performance – better (throughput, latency, vcpu utilization) for fast drives (e.g. Ramdisk, SSDs), but ***not*** for spinning disks
      - I/O scheduler's benefits (e.g. request merging) outweigh bio path's advantage
    - Availability
      - Upstream kernels (since 3.7)
    - How to enable (disabled by default)
      - Add 'virtio_blk.use_bio=1" to kernel cmdline
      - Modprobe virtio_blk use_bio=1

Virtio-scsi

- **SCSI support for KVM guests**
  - Rich SCSI feature set – true SCSI devices (seen as /dev/sd* in KVM guest)
  - Virtio-scsi device = SCSI Host Bus Adapter (HBA)
    - Large number of disks per virtio-scsi device
    - Easier for P2V migration – block devices appear as /dev/sd*
  - How to enable virtio-scsi https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Virtualization_Administration_Guide/sect-Managing_storage_controllers_in_a_guest.html
  - Performance is OK (*see next slide*)
  - Availability:
    - *Red Hat Enterprise Linux 6.4* and later
    - *SUSE Linux Enterprise Server 11 SP3* and later
  - More information
    - http://wiki.qemu.org/Features/VirtioSCSI
    - "*Better Utilization of Storage Features from KVM Guest via virtio-scsi*" presentation at 2013 LinuxCon/CloudOpen by Masaki Kimura (Hitachi)

Virtio-scsi vs. Virtio-blk Performance



Recap

- **Low Throughput / High Latencies**
  - Proper configuration & performance tuning are critical
    - Storage virtualization types (para-virtualization vs. full emulation), device- vs. file-backed virtual storage, image formats (raw vs. qcow2), guest caching mode (write-through vs. none), I/O scheduler (deadline) are all important
- **Support for high I/O rates**
  - KVM tops out at ~147,000 IOPS maximum per guest
  - Need *virtio-blk-data-plane* technology feature to scale better (1.6 million IOPS per guest!)
    - At least 50% higher than competing hypervisors
    - Available as a Technology Preview in RHEL 6.4 and SLES 11 SP3
- **Recent features**
  - *PV-EOI* ← reduce CPU utilization for workloads with high I/O & interrupt rates, need new guest kernels
  - *BIO-based virtio-blk* ← should help performance with fast storage (ramdisk, SSDs), but not slower spinning disks
  - *Virtio-scsi*
    - Support for SCSI devices in KVM guests, easier P2V migration
    - Performance is mostly comparable to virtio-blk
    - Available in RHEL 6.4 and SLES 11 SP3

धन्यवाद

Hindi

多謝

Traditional Chinese

ขอบคุณ

Thai

Спасибо

Russian

Gracias

Spanish

Thank You

English

شكراً

Arabic

Obrigado

Brazilian Portuguese

多谢

Simplified Chinese

Danke

German

שָׁלוֹם

Hebrew

Grazie

Italian

Merci

French

நன்றி

Tamil

ありがとうございました

Japanese

감사합니다

Korean