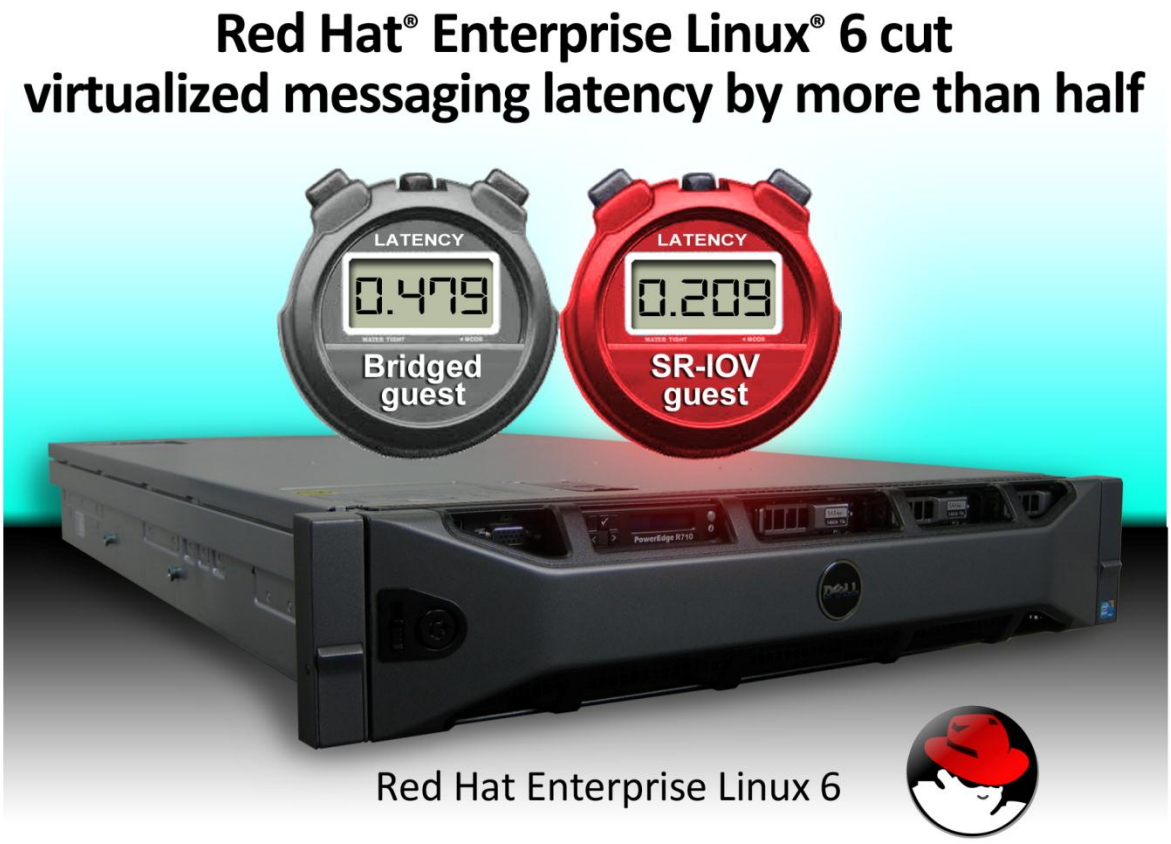
HARDWARE-ASSISTED VIRTUALIZED-I/O MESSAGING ANALYSIS ON RED HAT ENTERPRISE LINUX 6



OUR FINDINGS

Companies who rely upon time-sensitive, high-communication software applications require the lowest possible messaging latency. In Principled Technologies' tests in our labs, hardware-assisted Single Root I/O Virtualization (SR-IOV) guest virtual machines delivered messages on Red Hat Enterprise Linux 6.0 with Kernel-based Virtual Machine (KVM) with less than half the latency of standard KVM bridged networking for message sizes ranging from 8 to 1,024 bytes.

OUR PROCESS

To analyze the Advanced Message Queuing Protocol (AMQP) performance of the KVM hypervisor platform with different network I/O connections between the KVM guest and the host's network interface card (NIC), we used the profiling tools that Red Hat includes with MRG Messaging to compare both throughput and latency with and without SR-IOV hardware-assisted network I/O.

NOVEMBER 2010
A PRINCIPLED TECHNOLOGIES TEST REPORT
Commissioned by Red Hat, Inc.

PROJECT OVERVIEW

We ran Red Hat Enterprise Linux 6.0 with KVM on a server using the Intel® Ethernet Server Adapter X520-SR2, which can present up to 64 direct, virtualized network interfaces per port to the operating system and its guests via SR-IOV technology. PCI passthrough also provides direct connections between guests and the host's NIC, but can present only one interface per port. We tested the messaging performance of KVM guests in a configuration where standard PCI passthrough is not possible; that is, we use twice as many guests as there are network ports. For such a configuration, another network virtualization choice is to use a software bridge between the guest and the host's NIC. We compared the messaging performance for SR-IOV networking to this bridged networking with paravirtualized virtio network drivers on the guest.

To use SR-IOV to directly connect a guest to the host's NIC, the BIOS, server hardware, network adaptor, and operating system must all support the technology. We installed Red Hat Enterprise Linux 6 (kernel-2.6.32-71.el6) as the host operating system on a Dell™ R710 PowerEdge™ server with an SR-IOV enabled Dell BIOS and with an Intel X520-SR2 10Gb NIC.

We used the Advanced Message Queuing Protocol (AMQP) programs, qpid-perftest and qpid-latency-test, to compare both throughput and latency on the KVM hypervisor platform with different network I/O connections between the KVM guest and the host's NIC. We ran each test with bridged networking between the guest and NIC and with SR-IOV passthrough between the guest and NIC. For a baseline, we also ran each test without guest virtualization where each AMQP process could access the NIC without passing through any KVM layers (we refer to this baseline as *bare metal*). Qpid-latency-test scores measure response times, making lower numbers better, while qpid-perftest scores

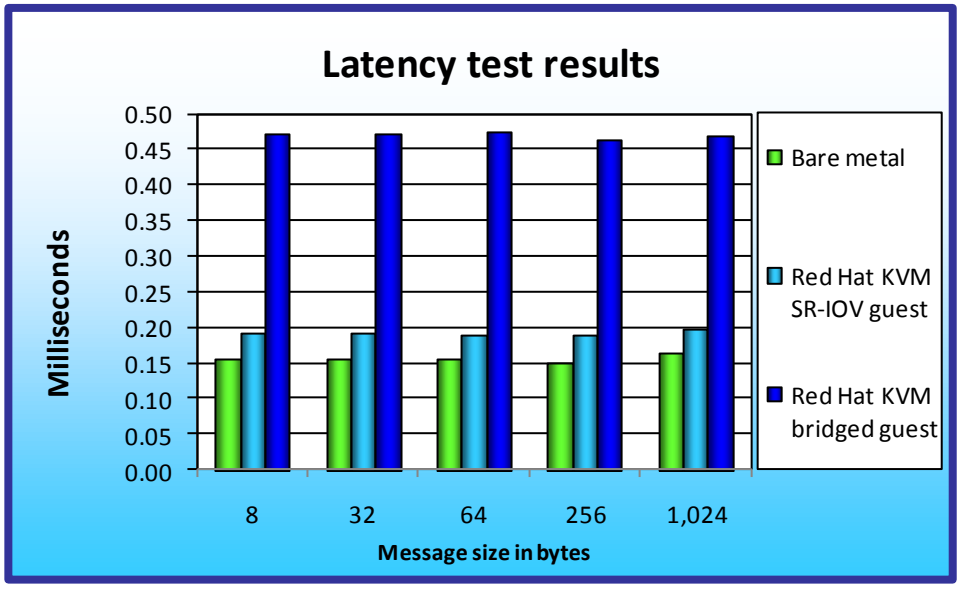


Figure 1: Latency test results, in milliseconds, for the three configurations at differing message sizes. Lower numbers are better.

measure number of messages transmitted successfully per second, making higher numbers better. The What we tested section explains the qpid-perftest and qpid-latency-test workloads in greater detail.

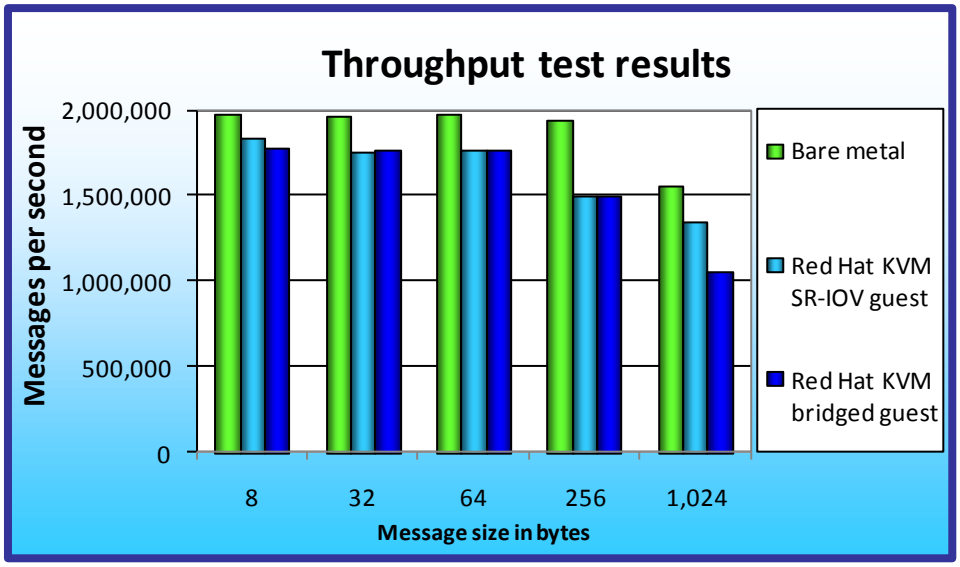


Figure 2: Throughput test results, in message per second, for the three configurations at differing message sizes. Higher numbers are better.

Figures 1 and 2 show the latency and throughput test results, respectively, of bare metal and KVM with the two networking approaches while running four guests on the host. The messaging latencies in the SR-IOV RHEL 6 KVM configuration were consistently two and one half times lower than those in the bridged networking configuration, and were within 20 percent of bare metal. The throughout performance for the SR-IOV and bridged configurations were comparable until a message size of 1,024 bytes. In the What we found section, we discuss the latency and throughput tests further.

For the latency tests, all configurations featured two virtual CPUs (vCPUs) and 5 GB of RAM on each guest. For the throughput tests, the bare metal and bridged configurations featured six vCPUs on each guest, and the SR-IOV configuration featured five vCPUs per guest. We choose to use fewer vCPUs for SR-IOV in order to dedicate two CPUs for interrupt handling in the SR-IOV network driver. The boot partition was on the virtual IDE controller. We used no external storage for this test.

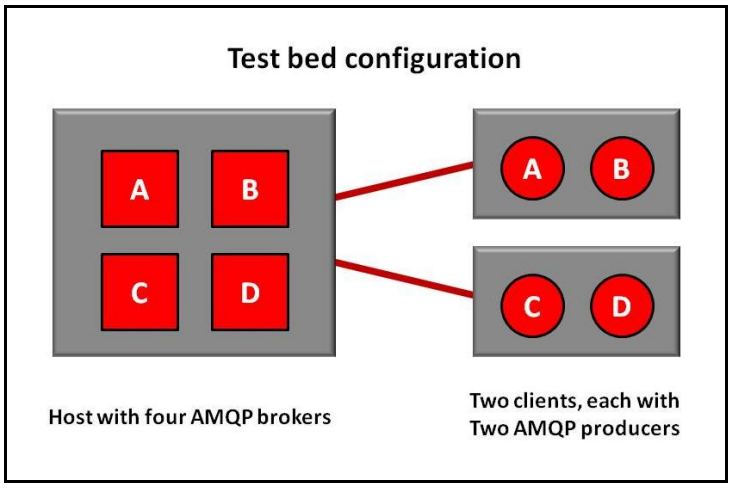


Figure 3: Schematic of the test bed.

Figure 3 illustrates the test bed configuration. We configured the AMQP broker host as follows: two Intel Xeon® X5670 2.93 GHz processors

with 24 GB RAM, and one 10Gb Intel X520-SR2 NIC. We installed and ran the qpid-perftest and qpid-latency-test workloads from two physical clients configured as follows: one client with two Intel Xeon Processors E5520 2.27 GHz and the second with two Intel Xeon Processors E5540 2.53 GHz. Both clients had 48 GB of RAM, one 10Gb Intel X520-SR NIC, and two SAS 15K 73GB hard drives. We installed Red Hat Enterprise Linux 6 (kernel-2.6.32-71.el6) as the operating system for the client systems and cabled the client NICs directly to the server under test in a peer-to- peer configuration.

WHAT WE TESTED

We conducted our testing using the software Red Hat includes in Enterprise MRG Messaging technology. The qpid-perftest and qpid-latency-test applications test throughput and latency, respectively, using MRG Messaging as the underlying software.

MRG Messaging enables the creation of distributed applications in which programs exchange data via sending and receiving messages. An organization can distribute a single application over a network, irrespective of differing operating systems, network protocols, or languages. In the MRG messaging model, the producer is any program that sends messages, the consumer is any program that receives them, and a broker is a messaging hub that routes messages from the publisher to their appropriate consumer (see Figure 4).

The producer sends messages to exchanges, while consumers subscribe to queues to receive the messages. Exchanges use routing and binding information to decide which queue to deliver a message to (see Figure 5).

Thus, the procedure is as follows: the producer sends a message to the exchange, where the exchange then assesses the bindings and places the message in the correct queue. The consumers then retrieve the messages from the

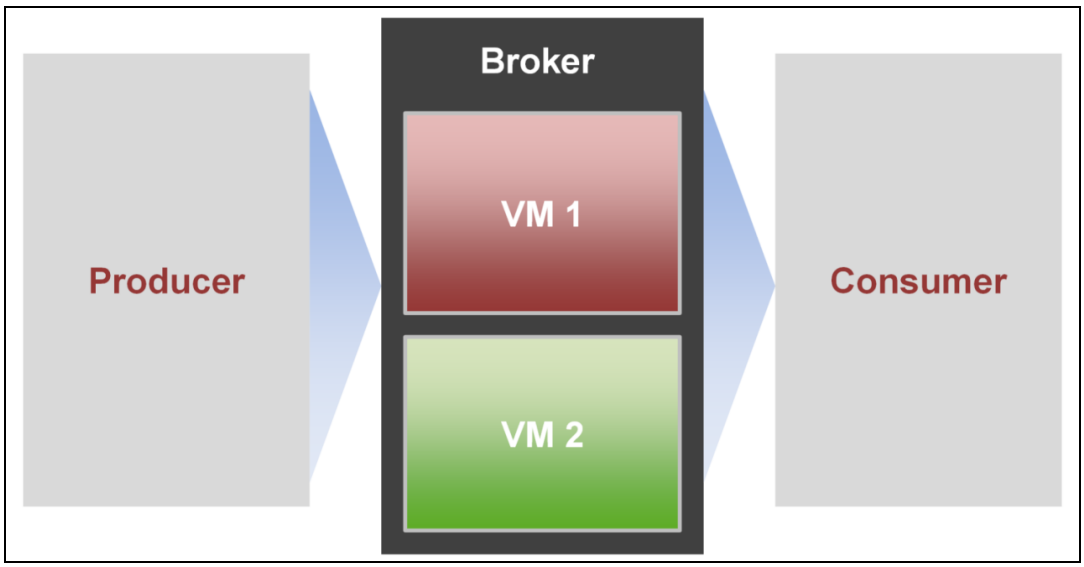


Figure 4: High-level view of the MRG messaging model. The producer is any program that sends messages, the consumer is any program that receives them, and a broker is a messaging hub that routes messages from the publisher to their appropriate consumer.

correct queues.

Red Hat MRG Messaging uses the Advanced Message Queuing Protocol (AMQP) protocol, an open-source messaging protocol, and is based on Apache Qpid (Qpid), a multi-platform messaging implementation of the AMQP protocol. Red Hat Enterprise MRG Messaging adds to Qpid by implementing certain message persistence and durability options, Linux kernel optimizations, clustering and failover options, support for transactions, and additional OS services.

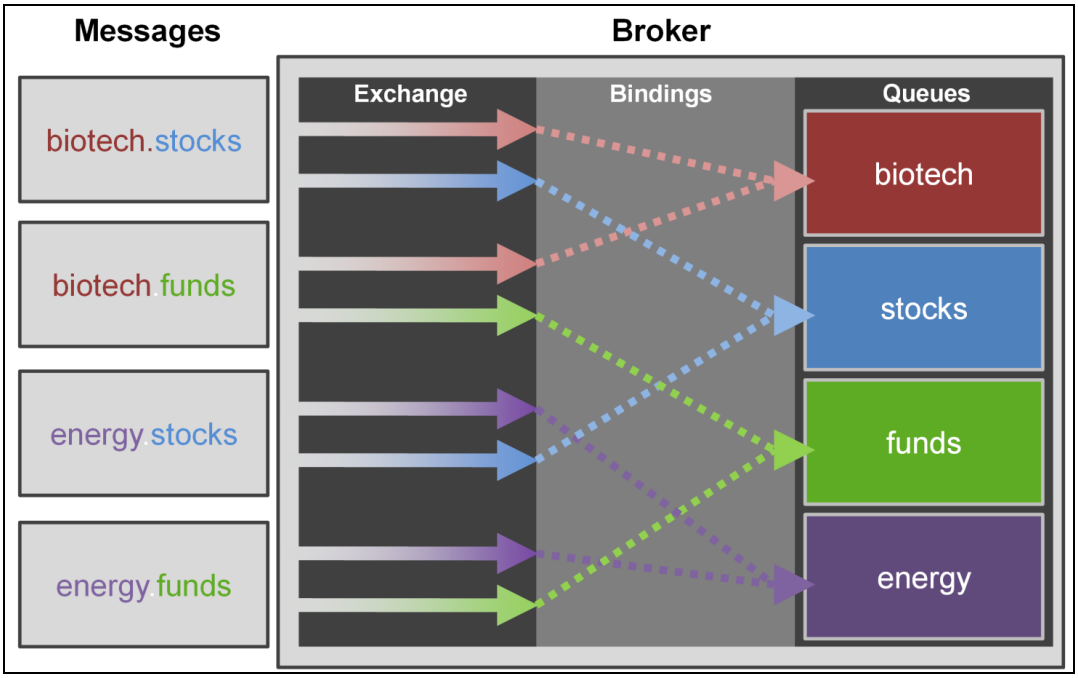


Figure 5: How the MRG messaging model broker handles messages. The producer sends messages to exchanges, while consumers subscribe to queues to receive the messages. Exchanges use routing and binding information to decide which queue to deliver a message to.

Our MRG Messaging configuration involves three systems: two client systems (producer/subscriber) and the server under test (broker). Acting as producer, the client systems generate messages that each sends to the broker in virtual machines on the server under test. The brokers inside the virtual machines contain exchanges that then route these messages to the subscriber, which in our case, was, again, the client systems.

The qpid-perftest program measures throughput as the total number of messages reliably transferred, divided by the time it took to transfer those messages. To run each test, qpid-perftest creates a control queue and manages the send-receive message process. When the message transfers are complete, qpid-perftest then records a timestamp and calculates the transfer rate for the messages. For each throughput configuration, we sent one million messages and ran each configuration 15 times.

The qpid-latency-test program measures messaging latency as the total time it takes the producer (client system) to send messages to the broker (server under test) and for the exchange inside the broker (server under test) to route the message back to the subscriber (client system). The client system reports the minimum, maximum, and average reporting interval time when using a rate, and reports all the sent messages

when using a count. Qpid-latency-test specifies a message count and timestamps all messages, and measures latency in milliseconds (ms). We ran all latency configurations for 60-second durations and ran each configuration nine times. We ran each workload for messages of 8, 32, 64, 512, and 1,024 bytes.

For messaging throughput tests, the goal was to see how many messages could complete in a certain time, and for messaging latency, the goal was to determine the quickest response possible. We ran each workload for messages of 8, 32, 64, 512, and 1,024 bytes.

For each workload, we ran two virtual networking configurations and one unvirtualized (bare metal). The first virtualized configuration was a typical bridged scenario, while the second configuration used SR-IOV passthrough, where a guest directly drives the PCI device.

WHAT WE FOUND

Here we review the test results for both messaging latency and throughput tests. For each latency-test configuration, we ran 9 iterations of the workload, and 15 iterations for throughout tests. We chose the run that corresponded to the median standard deviation, and report the results from that run for each configuration.

Messaging latency tests

Figure 6 shows the median latencies, in milliseconds, for the bare metal configuration and for Red Hat KVM running four AMQP brokers at differing message sizes with bridged networking and SR-IOV passthrough. The latencies using SR-IOV hardware-assisted networking were approximately 2.5 times better than those using software-assisted networking and were within 20 percent of bare metal (unvirtualized AMQP brokers) over the entire range of message sizes.

Message size in bytes	Bare metal	Red Hat KVM SR-IOV guests	Red Hat KVM bridged guests
8	0.156132	0.190674	0.473136
32	0.156916	0.190900	0.473425
64	0.156558	0.190457	0.473871
256	0.150549	0.190061	0.462908
1,024	0.164895	0.197115	0.469656

Figure 6: Latency test results, in milliseconds, for the three configurations at differing message sizes. Lower numbers are better.

Messaging throughput tests

Figure 7 shows the throughput results, in messages per second for the bare metal configuration and for Red Hat KVM running four AMQP brokers at differing message sizes with bridged networking and SR-IOV passthrough. The throughputs for SR-IOV and bridged networking were equal to within 1 to 3 percent for small message sizes. The SR-IOV throughputs were 22 percent higher than bridged networking for largest message size. The SR-IOV throughputs were generally within 10 to 15 percent of bare metal values with the exception of the 256-byte message (30 percent lower than bare metal values).

Message size in bytes	Bare metal	Red Hat KVM SR-IOV guests	Red Hat KVM bridged guests
8	1,980,471	1,833,481	1,782,404
32	1,971,225	1,754,824	1,768,600
64	1,980,776	1,764,252	1,770,448
256	1,939,169	1,498,026	1,500,313
1,024	1,556,762	1,344,517	1,052,262

Figure 7: Throughput test results, in messages per second, for the three configurations at differing message sizes. Higher numbers are better.

HOW WE TESTED

Adjusting BIOS settings

We enabled the virtualization feature in the host BIOS for our virtualization testing and enabled the SR-IOV feature in the BIOS for our SR-IOV testing. We disabled C-states and set Power Management to Maximum Performance for this testing.

Setting up the host server with Red Hat Enterprise Linux 6 with KVM

For the Red Hat KVM configuration, we installed kernel-2.6.32-71.el6 of Red Hat Enterprise Linux 6 on the host server, and then installed the packages necessary for KVM and AMQP.

Installing Red Hat Enterprise Linux 6

1. Insert and boot from the Red Hat Enterprise Linux 6.0 Install DVD.
2. Press Enter to install using graphical mode.
3. At the media test screen, select Skip.
4. At the Red Hat Enterprise Linux 6 title screen, click Next.
5. At the Choose an Installation Language screen, select English, and click Next.
6. At the Keyboard Type screen, select U.S. English, and click Next.
7. At the Storage Devices screen, select Basic Storage Devices, and click Next.
8. If a warning for device initialization appears, select Re-initialize for every storage device needed by the installation.

9. At the Name the Computer screen, type `##.domain.com`, where ## is the host name, and click Next.
10. At the Time zone selection screen, select the appropriate time zone, and click Next.
11. Enter the root password in the Root Password and Confirm fields, and click Next.
12. At the Partition selection screen, select Replace Existing Linux System(s), and click Next.
13. If a warning appears, click Write changes to disk.
14. At the default installation screen, click Next to begin the installation.
15. At the Congratulations screen, click Reboot.
16. After the system reboots and at the logon screen, type `root` for the user, enter the root password, and press Enter.

Installing the KVM packages in Red Hat Enterprise Linux 6

1. Copy the Red Hat Enterprise Linux 6.0 ISO image, and mount it at /mnt/rhel60/:

```
# mount –o loop /PATH/TO/ISO /mnt/rhel60
```

2. Create a text file at /etc/yum.repos.d/rhel-60-local.repo with the following contents:

```
[rhel-60-beta-dvd]
name=Red Hat Enterprise Linux $releasever - $basearch - DVD
baseurl=file:///mnt/rhel60/Server/
enabled=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-beta
```

3. Install the two GPG keys located on the Red Hat ISO by typing the following commands in a command prompt:

```
# rpm --import /mnt/rhel60/RPM-GPG-KEY-redhat-beta
# rpm --import /mnt/rhel60/RPM-GPG-KEY-redhat-release
```

4. Type the following command to install KVM packages:

```
# yum groupinstall Virtualization “Virtualization Client” \
                   “Virtualization Platform”
```

Setting up the network adapters on Red Hat Enterprise Linux KVM for bare metal and SR-IOV tests.

1. Log onto the host.
2. Type the following command to edit the network configuration settings, where X is the relevant host network interface to modify:

```
# vi /etc/sysconfig/network-scripts/ifcfg-ethX
```

3. Modify the following lines to set the static IP address and netmask, where XXX represents the remaining parts of the relevant IP address:

```
BOOTPROTO=static
IPADDR=192.168.XXX.XXX
NETMASK=255.255.255.0
```

```
IPV6INIT=no
```

4. Save the file, and exit vi.
5. Repeat steps 2 through 4 for the remaining interfaces.
6. Type the following command to modify the remaining network settings:

```
# vi /etc/sysconfig/network
```

7. Modify the following lines to set the default IP gateway where XXX represents the remaining parts of the relevant gateway IP address.

```
NETWORKING=yes
HOSTNAME=hostname.local.domain
GATEWAY=192.168.XXX.XXX
```

8. Save the file, and exit vi.
9. Type the following command to restart the network services:

```
# service network restart
```

Setting up the client with the Red Hat Enterprise Linux 6

For the Red Hat client to drive the workloads, we installed Red Hat Enterprise Linux 6 (kernel-2.6.32-44-1.el6) on the server following the instructions in section Installing Red Hat Enterprise Linux 6, and then installed the packages necessary for AMQP, following the instructions in the section Installing AMQP on the host, client and host VMs.

Completing configurations in the Red Hat KVM host

Disabling unneeded services

1. Disable the following services by running this bash shell loop:

```
# for service in \
   abrtd atd auditd autofs certmonger cgred crond dnsmasq ebtables
   haldaemon ip6tables iptables iscsi iscsid kdump ksm ksmtune libvirtd
   mdmonitor messagebus netconsole netfs nfs nfslock nscd nslcd oddjobd
   postfix psacct qpidd rdisc restorecond rhnsd rpcbind rpcgssd
   rpcidmapd rpcsvcgssd saslauthd smartd sssd ypbind;
do
  chkconfig $service off
  service $service stop
done

# chkconfig –level 0123456 cpuspeed off
# service cpuspeed stop
```

Disabling SELinux on the KVM host.

1. Log onto the host.

2. Type the following command to edit the SELinux configuration settings:

```
# vi /etc/selinux/config
```

3. Change the line beginning with SELINUX= to

```
SELINUX=disabled
```

4. Save the file, and exit vi.

Enabling huge pages memory functionality in Red Hat KVM

1. Log onto the host.
2. Type the following command to edit the system configuration settings:

```
# vi /etc/sysctl.conf
```

3. Add the following lines to the bottom of the file to reserve approximately 17.5GB for VM huge pages, with each page of size 2,048 KB:

```
vm.nr_hugepages = 9000
```

4. Save the file, and exit vi
5. Type the following command to edit the fstab file:

```
# vi /etc/fstab
```

6. Add the following line to the bottom of the file:

```
hugetlbfs /mnt/libhugetlbfs hugetlbfs defaults 0 0
```

7. Save the file, and exit vi.
8. Type the following to create the directory for the hugepages files:

```
# mkdir –p /mnt/libhugetlbfs
```

9. Type the following commands to immediately activate hugepages, if desired:

```
# sysctl vm.nr_hugepages=9000
# mount /mnt/libhugetlbfs
```

Setting up the bridged network for the Red Hat Enterprise Linux KVM

1. Log onto the host.
2. Type the following command to create a configuration file for the bridged network device:

```
# vi /etc/sysconfig/network-scripts/ifcfg-br0
```

3. Add the following lines to the file. Note that, in the below example, XXX signifies a placeholder:

```
# Interface details – copy from ifcfg-ethX file
DEVICE=br0
```

```
TYPE=Bridge
DELAY=0
NM_CONTROLLED=no
BOOTPROTO=none
ONBOOT=yes
IPV6INIT=no
IPADDR=192.168.XXX.XXX
NETMASK=255.255.255.0
```

4. Save the file, and exit vi.
5. Type the following command to edit the ifcfg-ethX file, where X is the number of the network interafce you are using for the bridge:

```
# vi /etc/sysconfig/network-scripts/ifcfg-ethX
```

6. Modify the file to point to the bridge device:

```
# Interface details
DEVICE=eth0
HWADDR=00:1B:21:29:CE:74
NM_CONTROLLED=no
BRIDGE=br0
```

7. Save the file, and exit vi.
8. Repeat for each 10Gb interface.
9. Restart the network by typing the following:

```
# service network restart
```

Setting the elevator=deadline option in the grub.conf file

1. Log onto the host.
2. Type the following command to edit the grub configuration settings:

```
# vi /etc/grub.conf
```

3. Add the following text to the end of kernel line:

```
elevator=deadline
```

4. Save the file, and exit vi.

Setting system start parameters in the /etc/sysctl.conf file

1. Log onto the host, right-click the desktop, and choose Open Terminal.
2. Type the following command to edit the system configuration settings:

```
# vi /etc/sysctl.conf
```

3. Add or modify the following lines:

```
kernel.msgmni = 16384
kernel.sem = 250 32000 32 2048
kernel.msgmax = 65535
kernel.msgmnb = 1310724

fs.file-max = 65536

net.ipv4.conf.all.arp_filter = 0
net.ipv4.conf.lo.arp_filter = 0
net.ipv4.conf.default.arp_filter = 1
```

4. Save the file, and exit vi.

Creating and configuring the VMs

Creating the first guest using the Virtual Machine Manager

1. Log onto the Red Hat Enterprise Linux 6.0 host.
2. Go to Applications→System Tools→Virtual Machine Manager.
3. Click New.
4. Click Forward.
5. Choose a name for the guest. Click Forward.
6. Keep the default of Fully virtualized, choose the CPU architecture, and choose kvm as the hypervisor.
7. Assuming you have copied the Red Hat Enterprise Linux 6 media to the host machine, choose Local install media, Linux as the OS Type, and Red Hat Enterprise 6 as the OS Variant. Click Forward.
8. Click Browse to browse to the ISO Linux location.
9. Locate the ISO file, select it, and click Open. Click Forward.
10. Choose File (disk image), and specify the location where you wish to store the IMG file. In our case, we chose the default location.
11. Specify 8192MB for the size of the IMG file, and uncheck and recheck the Allocate entire virtual disk now checkbox. Click Forward.
12. Choose Shared physical device, and select the management NIC.
13. Set Max memory size and Startup memory size to 8192MB, and set the number of Virtual CPUs to 12. Click Forward.
14. Review the summary information, and click Finish.

Installing the guest operating system

1. Double-click the new VM to connect to the console.
2. On the Hardware tab in Virtual Machine Manager, specify the ISO image on the host machine as the CD drive of the VM. Right-click the VM, and choose Run.
3. Power on the VM.
4. Press Enter to install using graphical mode.
5. At the media test screen, select Skip.
6. At the Red Hat Enterprise Linux 6 title screen, click Next.
7. At the Choose an Installation Language screen, select English, and click Next.
8. At the Keyboard Type screen, select U.S. English, and click Next.
9. At the Storage Devices screen, select Basic Storage Devices, and click Next.

10. If a warning for device initialization appears, select Re-initialize for every storage device needed the installation.
11. At the Name the Computer screen, type `##.domain.com`, where ## is the host name, and click Next.
12. At the Time zone selection screen, select the appropriate time zone, and click Next.
13. Enter the root password in the Root Password and Confirm fields, and click Next.
14. At Partition selection screen, select Replace Existing Linux System(s), and click Next.
15. If a warning appears, click Write changes to disk.
16. At the default installation screen, click Next to begin the installation.
17. At the Congratulations screen, click Reboot.

Configuring networking in the VM

1. Power on the VM, and open the console.
2. Log onto the VM, right-click the desktop, and choose Open Terminal.
3. Type the following command to edit the network configuration settings:

```
# vi /etc/sysconfig/network-scripts/ifcfg-eth0
```

4. Modify the following lines to set the static IP address and netmask, where XXX is the remaining portion of your IP Address:

```
BOOTPROTO=static
IPADDR=192.168.XXX.XXX
NETMASK=255.255.255.0
```

5. Save the file, and exit vi.
6. Repeat steps 3 through 5 for eth1, the 10Gb NIC reserved for testing.
7. Type the following command to modify the remaining network settings:

```
# vi /etc/sysconfig/network
```

8. Modify the following lines to disable IPv6 and set the hostname:

```
NETWORKING=yes
HOSTNAME=VM1.local.domain
```

9. Save the file, and exit vi.

Configuring additional tuning options in the VM

1. Log onto the VM using Virtual Machine Manager, or using an ssh client, such as Putty.
2. Modify the following lines to /etc/sysctl.conf:

```
net.ipv4.conf.all.arp_filter = 0
net.ipv4.conf.lo.arp_filter = 0
net.ipv4.conf.default.arp_filter = 1
```

3. Save the file, and exit vi.
4. Disable the unnecessary services as above for the KVM host.

5. Disable SELinux as above for the KVM host.

Creating VM Startup Shell scripts to use huge pages

1. Start the libvirt daemon, if necessary, by executing service libvirtd start.
2. Stop all running VMs by executing virsh shutdown VM-name.
3. Type the following to edit each VM's configuration:

```
# virsh edit VM-name
```

Add the following lines after the <memory> .. </memory> section to each VM's XML description file:

```
<memoryBacking> <hugepages/> </memoryBacking>
```

4. Save and exit the editor.

Setting up AMQP

Installing AMQP on the host, client and host VMs

1. Log onto the host.
2. Extract the MRG 1.3 software into a directory. If necessary, remount the Red Hat Enterprise Linux 6 Beta media.
3. Using rpm or yum, install the following AMQP packages and dependencies:
   - boost-program-options-1.41.0-11.el6.x86_64
   - boost-system-1.41.0-11.el6.x86_64
   - boost-filesystem-1.41.0-11.el6.x86_64
   - boost-date-time-1.41.0-11.el6.x86_64
   - boost-thread-1.41.0-11.el6.x86_64
   - boost-wave-1.41.0-11.el6.x86_64
   - boost-serialization-1.41.0-11.el6.x86_64
   - libicu-4.2.1-9.el6.x86_64
   - boost-regex-1.41.0-11.el6.x86_64
   - boost-graph-1.41.0-11.el6.x86_64
   - boost-iostreams-1.41.0-11.el6.x86_64
   - boost-python-1.41.0-11.el6.x86_64
   - boost-signals-1.41.0-11.el6.x86_64
   - boost-test-1.41.0-11.el6.x86_64
   - boost-1.41.0-11.el6.x86_64
   - boost-devel-1.41.0-11.el6.x86_64
   - libuuid-devel-2.17.2-4.el6.x86_64
   - qpid-cpp-client-devel-0.7.946106-7.el6.x86_64
   - qpid-cpp-client-0.7.946106-7.el6.x86_64

4. Type the following to prevent the Qpid Broker from starting automatically:

```
# chkconfig qpidd off
```

5. Repeat for the client, and each guest.

Creating additional guests on the host server

After installing and configuring the initial guest, we created the additional guest by using virt-clone on the Red Hat Enterprise Linux platform.

Cloning the guests on Red Hat Enterprise Linux

Run the following command to clone GUEST1 to the area reserved for GUEST2:

```
# virt-clone --original GUEST1 --name GUEST2 --file
/var/lib/libvirt/images/MRG-GUEST2.img –nonsparse
```

Configuring the additional guests after cloning

Modify the IP addresses in each guest as we discuss in the above section, Configuring networking in the VM.

Performing the tests

Bare metal procedure

1. On the host, configure the network adapter for standard networking as in the section Setting up the bridged network adapters for the Red Hat Enterprise Linux KVM for bare metal and SR-IOV tests.
2. Start the AMQP brokers on the host with this script. Note that the brokers 2-4 must listen on a different TCP/IP port (5673), and have their own HOME directory. Replace NumThreads by 1 for latency tests, and 7 for throughput tests.

```
pkill -9 qpidd
sleep 2
mkdir /tmp/qp-{1,2,3,4}
opts="--worker-threads NumThreads --auth=no --mgmt-enable=no --tcp-
nodelay -d"
cmd= nice -n -20 qpidd $opts”

env HOME=/tmp/qp-1 numactl -C0,2,4,6,8,10      -m0 $cmd
env HOME=/tmp/qp-2 numactl -C12,14,16,18,20,22 -m0 $cmd -p 5673
env HOME=/tmp/qp-3 numactl -C1,3,5,7,9,11      -m1 $cmd -p 5674
env HOME=/tmp/qp-4 numactl -C13,15,17,19,21,23 -m1 $cmd -p 5675
unset opts
```

3. Log onto the second client.
4. Distribute the client-to-host network interface’s IRQs over the CPUs by running the following script in which ethX is the interface name.

```
service irqbalance stop
shift=0
for irq in `awk -F: '/ethX-/ {n++; if(n==1){a=$1}; b=$1} \
     END{for (i=a; i <= b; i+=2) {print i}}' /proc/interrupts`; do
```

```
        let jrq="$irq+1"
        echo "$shift, $irq, $jrq"
        printf "%X" $((1<<$shift)) > /proc/irq/$irq/smp_affinity
        printf "%X" $((1<<$shift)) > /proc/irq/$jrq/smp_affinity
        let shift+=2
    done
```

5. Log onto the first client and repeat Step 4.
6. Perform the throughput test from the first client by running the following script, where 192.168.XXX.AAA/192.168.XXX.BBB and 192.168.XXX.CCC/192.168.XXX.CCC are the IP addresses for the host's network adapters on client 1 and 2, respectively:

```
for msg in 8 32 64 256 1024;
    do echo $msg;
     for run in `seq 1 15`;
     do echo -e "\nRun $run; size $msg";

        opts="--size $msg --count 1000000 --nsubs 1 --npubs 1 --qt 6 -
        -bounds-multiplier 102400 --summary --tcp-nodelay"
        cmd="nice -n -20 /usr/bin/qpid-perftest $opts"

        pkill -9 qpid- ; ssh client-2 pkill -9 qpid-
        sleep 2

        numactl -N0 -m0 $cmd -b 192.168.XXX.AAA &
        numactl -N1 -m1 $cmd -b 192.168.XXX.BBB –p 5673 &
        ssh client-2 \
        numactl -N0 -m0 $cmd -b 192.168.XXX.CCC –p 5674 &
        ssh client-2 \
        numactl -N1 -m1 $cmd -b 192.168.XXX.DDD –p 5675 &


        wait
    done 2>&1 | tee -a PerfTest.txt
done
```

7. Perform the latency test from the first client by running the following script:

```
for msg in 8 32 64 256 1024;
   do echo $msg;
       for run in `seq 1 9`;
       do echo -e "\nRun $run; size $msg";
           rate1=1000
           let rate2="$rate1+1"
           let rate3="$rate1+2"
           let rate4="$rate1+3"
           opts="--size $msg --csv --tcp-nodelay"
           cmd="nice -n -20 /usr/bin/qpid-latency-test $opts"
           pkill -9 qpid- ; ssh client-2 pkill -9 qpid-
           sleep 2
```

```
            numactl -N0 -m0 $cmd -b 192.168.XXX.AAA –rate $rate1 &
            numactl -N1 -m1 $cmd -b 192.168.XXX.BBB –rate $rate2 –p 5673 &
            ssh client-2 \
            numactl -N0 -m0 $cmd -b 192.168.XXX.CCC –rate $rate3 –p 5674 &
            ssh client-2 \

            numactl -N1 -m1 $cmd -b 192.168.XXX.DDD –rate $rate4 –p 5675 &

            sleep 60; pkill -9 qpid- ; ssh client-2 pkill -9 qpid-
        done 2>&1 | tee -a LatencyTest.txt
done
```

Bridged networking procedure

1. On the host, configure the network adapter for bridged networking as in the Setting up the bridged network for the Red Hat Enterprise Linux KVM section, and reboot.
2. Start the libvirt daemon by typing the following command:

```
# service libvirtd start
```

3. Start the guests by typing the following commands (replace the guest domain names with their appropriate values):

```
# virsh start name-of-guest-one
# virsh start name-of-guest-two
# virsh start name-of-guest-three
# virsh start name-of-guest-four
```

4. Start the AMQP broker on each guest with the following script, where 192.168.XXX.AAA, 192.168.XXX.BBB, etc. are the IP addresses of the guests, and replace NumThreads by 1 for latency tests, and 6 for throughput tests.

```
ips=(192.168.XXX.AAA 192.168.XXX.BBB 192.168.XXX.CCC 192.168.XXX.DDD)
opts="--worker-threads NumThreads –auth=no --mgmt-enable=no --tcp-
nodelay –d"
for ip in  ${ips[*]}; do
     ssh $ip pkill -9 qpidd \; sleep 2\; qpidd $opts
done
unset ips
unset opts
```

5. Pin each guest's vCPUs to one physical CPU with the following script:

```
for vpu in `seq 0 5`; do
    # socket 0
    let cpu1="4*$vcpu"
    let cpu2="$cpu1+2"
    # socket 1
    let cpu3="$cpu1+1"
    let cpu4="$cpu1+3
```

```
    echo "$vcpu – Guests 1-4: $cpu1, $cpu2, $cpu3, $cpu4"

    virsh vcpupin name-of-guest-one   $vcpu $cpu1
    virsh vcpupin name-of-guest-two   $vcpu $cpu2
    virsh vcpupin name-of-guest-three $vcpu $cpu3
    virsh vcpupin name-of-guest-four  $vcpu $cpu4
done
```

6. Log onto the second client.
7. Distribute the client-to-host network interface's IRQs over the CPUs by running the following script in which ethX is the interface name:

```
service irqbalance stop
shift=0
for irq in `awk -F: '/ethX-/ {n++; if(n==1){a=$1}; b=$1} \
     END{for (i=a; i <= b; i+=2) {print i}}' /proc/interrupts`; do
     let jrq="$irq+1"
     echo "$shift, $irq, $jrq"
     printf "%X" $((1<<$shift)) > /proc/irq/$irq/smp_affinity
     printf "%X" $((1<<$shift)) > /proc/irq/$jrq/smp_affinity
     let shift+=2
done
```

8. Log onto the first client and repeat the Step 7.
9. Perform the throughput test from the first client by running the following script, where 192.168.XXX.AAA/192.168.XXX.BBB and 192.168.XXX.CCC/192.168.XXX.CCC are the IP addresses for the host's network adapters on client 1 and 2, respectively:

```
for msg in 8 32 64 256 1024;
do echo $msg;
    for run in `seq 1 15`;
    do echo -e "\nRun $run; size $msg";
        opts="--size $msg --count 1000000 --nsubs 1 --npubs 1 --qt 8 -
        -bounds-multiplier 102400 --summary --tcp-nodelay"
        cmd="nice -n -20 /usr/bin/qpid-perftest $opts"

        pkill -9 qpid- ; ssh client-2 pkill -9 qpid-
        sleep 2

        numactl -N0 -m0 $cmd -b 192.168.XXX.AAA &
        numactl -N1 -m1 $cmd -b 192.168.XXX.BBB &
        ssh client-2 \
        numactl -N0 -m0 $cmd -b 192.168.XXX.CCC &
        ssh client-2 \
        numactl -N1 -m1 $cmd -b 192.168.XXX.DDD &

        wait
    done 2>&1 | tee -a PerfTest.txt
done
```

10. Perform the latency test from the first client by running the following script:

```
for msg in 8 32 64 256 1024;
do echo $msg;
    for run in `seq 1 9`;
    do echo -e "\nRun $run; size $msg";
        rate1=1000
        let rate2="$rate1+1"
        let rate3="$rate1+2"
        let rate4="$rate1+3"
        opts="--size $msg --csv --tcp-nodelay"
        cmd=”nice -n -20 /usr/bin/qpid-latency-test $opts”
        pkill -9 qpid- ; ssh client-2 pkill -9 qpid-
        sleep 2

        numactl -N0 -m0 $cmd -b 192.168.XXX.AAA –rate $rate1 &
        numactl -N1 -m1 $cmd -b 192.168.XXX.BBB –rate $rate2 &
        ssh client-2 \
        numactl -N0 -m0 $cmd -b 192.168.XXX.CCC –rate $rate3 &
        ssh client-2 \
        numactl -N1 -m1 $cmd -b 192.168.XXX.DDD –rate $rate4 &

        sleep 60; pkill -9 qpid- ; ssh client-2 pkill -9 qpid-
    done 2>&1 | tee -a LatencyTest.txt
done
```

SR-IOV passthrough procedure

1. On the host, configure the network adapter for standard networking as in the Setting up the bridged network for the Red Hat Enterprise Linux KVM section.
2. Edit the /etc/grub.conf file to enable SR-IOV in the kernel. Type the following:

```
# vi /etc/grub.conf
```

3. Add the following text to the end of the kernel line:

```
intel_iommu=on
```

4. Save the file, and exit vi.
5. Configure one VF per network adaptor. Create a new driver-initialization file by typing the following:

```
# echo “options ixgbe max_vfs=2” > /etc/modprobe.d/sriov.conf
```

6. Reboot the host.
7. Log onto the host.
8. Start the irqbalance and libvirt daemons by typing the following command:

```
# service libvirtd start
```

9. Start the guests by typing the following commands (replace the guest domain names with their appropriate values):

```
# virsh start name-of-guest-one
# virsh start name-of-guest-two
# virsh start name-of-guest-three
# virsh start name-of-guest-four
```

10. Start the AMQP broker on each guest with the following script, where 192.168.XXX.AAA, 192.168.XXX.BBB, etc. are the IP addresses of the guests. Replace NumThreads by 1 for latency tests, and 7 for throughput tests.

```
ips=(192.168.XXX.AAA 192.168.XXX.BBB 192.168.XXX.CCC 192.168.XXX.DDD)
opts="--worker-threads NumThreads –auth=no --mgmt-enable=no --tcp-
nodelay –d"
for ip in  ${ips[*]}; do
     ssh $ip pkill -9 qpidd \; sleep 2\; qpidd $opts
done
unset ips
unset opts
```

11. Pin each guest's vCPUs to one physical CPU with the following script:

```
for vpu in `seq 0 4`; do
    # socket 0
    let cpu1="4*$vcpu+4"
    let cpu2="$cpu1+2"
    # socket 1
    let cpu3="$cpu1+1"
    let cpu4="$cpu1+3
    echo "$vcpu – Guests 1-4: $cpu1, $cpu2, $cpu3, $cpu4"

    virsh vcpupin name-of-guest-one   $vcpu $cpu1
    virsh vcpupin name-of-guest-two   $vcpu $cpu2
    virsh vcpupin name-of-guest-three $vcpu $cpu3
    virsh vcpupin name-of-guest-four  $vcpu $cpu4

done
```

12. Log onto the second client.
13. Distribute the client-to-host network interface's IRQs over the CPUs by running the following script in which ethX is the interface name:

```
service irqbalance stop
shift=0
for irq in `awk -F: '/ethX-/ {n++; if(n==1){a=$1}; b=$1} \
     END{for (i=a; i <= b; i+=2) {print i}}' /proc/interrupts`; do
     let jrq="$irq+1"
     echo "$shift, $irq, $jrq"
     printf "%X" $((1<<$shift)) > /proc/irq/$irq/smp_affinity
```

```
        printf "%X" $((1<<$shift)) > /proc/irq/$jrq/smp_affinity
        let shift+=2
done
```

14. Log onto the first client and repeat Step 13.
15. Perform the throughput test from the first client by running the following script, where 192.168.XXX.AAA/192.168.XXX.BBB and 192.168.XXX.CCC/192.168.XXX.CCC are the IP addresses for the host's network adapters on client 1 and 2, respectively:

```
for msg in 8 32 64 256 1024;
do echo $msg;
    for run in `seq 1 15`;
    do echo -e "\nRun $run; size $msg";
        opts="--size $msg --count 1000000 --nsubs 1 --npubs 1 --qt 6 -
        -bounds-multiplier 102400 --summary --tcp-nodelay"
        cmd=”nice -n -20 /usr/bin/qpid-perftest $opts”

        pkill -9 qpid- ; ssh client-2 pkill -9 qpid-
        sleep 2

        numactl -N0 -m0 $cmd -b 192.168.XXX.AAA &
        numactl -N1 -m1 $cmd -b 192.168.XXX.BBB &
        ssh client-2 \
        numactl -N0 -m0 $cmd -b 192.168.XXX.CCC &
        ssh client-2 \
        numactl -N1 -m1 $cmd -b 192.168.XXX.DDD &

        wait
    done 2>&1 | tee -a PerfTest.txt
done
```

16. Perform the latency test from the first client by running the following script:

```
for msg in 8 32 64 256 1024;
do echo $msg;
    for run in `seq 1 9`;
    do echo -e "\nRun $run; size $msg";
        rate1=1000
        let rate2="$rate1+1"
        let rate3="$rate1+2"
        let rate4="$rate1+3"
        opts="--size $msg --csv --tcp-nodelay"
        cmd=”nice -n -20 /usr/bin/qpid-latency-test $opts”
        pkill -9 qpid- ; ssh client-2 pkill -9 qpid-
        sleep 2

        numactl -N0 -m0 $cmd -b 192.168.XXX.AAA –rate $rate1 &
        numactl -N1 -m1 $cmd -b 192.168.XXX.BBB –rate $rate2 &
        ssh client-2 \
        numactl -N0 -m0 $cmd -b 192.168.XXX.CCC –rate $rate3 &
```

```
            ssh client-2 \
            numactl -N1 -m1 $cmd -b 192.168.XXX.DDD –rate $rate4 &

            sleep 60; pkill -9 qpid- ; ssh client-2 pkill -9 qpid-
        done 2>&1 | tee -a LatencyTest.txt
    done
```

17. At the completion of the SR-IOV tests, shutdown the guests, remove SR-IOV-specific configuration options, and reboot by typing the following commands:

```
# virsh shutdown name-of-guest-one
# virsh shutdown name-of-guest-two
# virsh shutdown name-of-guest-three
# virsh shutdown name-of-guest-four
# rm etc/modprobe.d/sriov.conf
# vi /etc/grub.conf # remove intel_iommu=on
# shutdown –r now
```

APPENDIX A – UNIX AND LINUX-BASED SERVER DISCLOSURE

Figure 8 provides detailed information about the servers we used in our testing.

System	System under test	Client 1	Client 2
Power supplies			
Total number	2	1	2
Vendor and model number	Dell A570P-00	Dell 870P-00	Dell OPT-164
Wattage of each (W)	570	870	870
Cooling fans			
Total number	5	5	5
Dimensions (h x w) of each	2-1/2" x 2-1/2"	2-1/2" x 2-1/2"	2-1/2" x 2-1/2"
Volts	12	12	12
Amps	1.6	1.6	1.5
General			
Number of processor packages	2	2	2
Number of cores per processor	6	4	4
Number of hardware threads per core	2	2	2
CPU			
Vendor	Intel	Intel	Intel
Name	Xeon	Xeon	Xeon
Model number	X5670	E5520	E5540
Stepping	01	04	05
Socket type	LGA1366	LGA1366	LGA1366
Core frequency (GHz)	2.93	2.27	2.53
Bus frequency (GT/s)	6.40	5.86	5.86
L1 cache (KB)	32 + 32 (per core)	32 + 32 (per core)	32 + 32 (per core)
L2 cache (KB)	256 (per core)	256 (per core)	256 (per core)
L3 cache (MB)	12	8	8
Platform			
Vendor and model number	Dell PowerEdge R710	Dell PowerEdge R710	Dell PowerEdge R710
Motherboard model number	PWB9YY69	PWB9YY69	PWBYN967
Motherboard chipset	Intel 5520	Intel 5520	Intel 5520

System	System under test	Client 1	Client 2
BIOS name and version	Dell Incorporated 2.2.2 (9/21/2010) Revision 2.2	Dell Incorporated 2.1.15 (9/13/2010)	Dell Incorporated 2.1.9
BIOS settings	All defaults with the exception that C1E and C-States were disabled. SR-IOV was enabled depending on the test being performed.	All defaults with the exception that C1E and C-States were disabled.	All defaults with the exception that C1E and C-States were disabled.
Memory module(s)			
Total RAM in system (GB)	24	48	48
Vendor and model number	Crucial CT51272BB1339.36SFD1	Samsung M393B1K70BH1-CH9	Crucial CT51272BB1339.36SFD1
Type	PC3-10600R	PC3-10600R	PC3-10600R
Speed (MHz)	1,333	1,066	1,333
Speed running in the system (MHz)	1,333	1,066	1,333
Timing/Latency (tCL-tRCD-tRP-tRASmin)	9-9-9-24	7-7-7-20	9-9-9-24
Size (GB)	4	8	4
Number of RAM module(s)	6	6	12
Chip organization	Double-sided	Double-sided	Double-sided
Rank	Dual	Dual	Dual
Hard disk			
Vendor and model number	Seagate ST9146802SS	Seagate ST9146802SS	Seagate ST973451SS
Number of disks in system	2	2	2
Size (GB)	146	146	73
Buffer size (MB)	16	16	16
RPM	10,000	10,000	15,000
Type	SAS	SAS	SAS
Disk controller			
Vendor and model	LSI Logic / Symbios Logic SAS1068E	LSI Logic / Symbios Logic SAS1068E	Dell PERC 6/i
Controller driver (module)	2.6.32-71.el6.x86_64	2.6.32-44.1.x86_64	2.6.32-71.el6.x86_64
Controller driver version	0B4D557979D0BC8F39D9984	0F46530DEF17FB76B772767	0B4557979D0BC8F39D9984
Controller firmware	0.25.47.00-IR	0.25.47.00-IR	1.22.02-0612

System	System under test	Client 1	Client 2
RAID configuration	RAID 1	RAID 1	RAID 1
Operating system			
Name	Red Hat Enterprise Linux 6.0	Red Hat Enterprise Linux 6.0	Red Hat Enterprise Linux 6.0
File system	ext4	ext4	ext4
Kernel	2.6.32-71.el6.x86_64	2.6.32-44.1.el6.x86_64	2.6.32-71.el6.x86_64
Language	English	English	English
Graphics			
Vendor and model number	Matrox® G200eW	Matrox G200eW	Matrox G200eW
Ethernet			
Vendor and model number	Intel Corporation Ethernet Server Adapter X520-SR2	Intel Corporation Ethernet Server Adapter X520-SR2	Intel Corporation Ethernet Server Adapter X520-SR1
Type	PCI Express	PCI Express	PCI Express
Driver (Module)	2.6.32-71.el6.x86_64	2.6.32-44.1.el6_64	2.6.32-44.1.el6_64
Driver Version	2.0.62-k2	2.0.62-k2	2.0.62-k2
Optical drive(s)			
Vendor and model number	TEAC DV-28S	TEAC DV-28S	TEAC DV-28S-VDB
Type	DVD ROM	DVD ROM	DVD ROM
USB ports			
Number	4	4	4
Type	2.0	2.0	2.0

Figure 8: Detailed system configuration information for the test-bed servers.

ABOUT PRINCIPLED TECHNOLOGIES

Principled Technologies, Inc.
1007 Slater Road, Suite 300
Durham, NC, 27703
www.principledtechnologies.com

We provide industry-leading technology assessment and fact-based marketing services. We bring to every assignment extensive experience with and expertise in all aspects of technology testing and analysis, from researching new technologies, to developing new methodologies, to testing with existing and new tools.

When the assessment is complete, we know how to present the results to a broad range of target audiences. We provide our clients with the materials they need, from market-focused data to use in their own collateral to custom sales aids, such as test reports, performance assessments, and white papers. Every document reflects the results of our trusted independent analysis.

We provide customized services that focus on our clients' individual requirements. Whether the technology involves hardware, software, Web sites, or services, we offer the experience, expertise, and tools to help our clients assess how it will fare against its competition, its performance, its market readiness, and its quality and reliability.

Our founders, Mark L. Van Name and Bill Catchings, have worked together in technology assessment for over 20 years. As journalists, they published over a thousand articles on a wide array of technology subjects. They created and led the Ziff-Davis Benchmark Operation, which developed such industry-standard benchmarks as Ziff Davis Media's Winstone and WebBench. They founded and led eTesting Labs, and after the acquisition of that company by Lionbridge Technologies were the head and CTO of VeriTest.