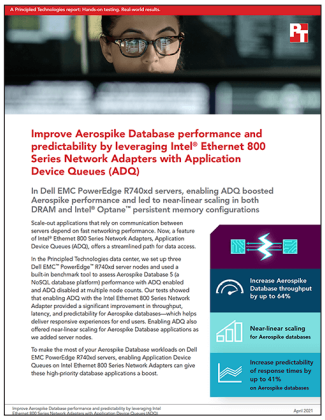

The science behind the report:

Improve Aerospike Database performance and predictability by leveraging Intel® Ethernet 800 Series Network Adapters with Application Device Queues (ADQ)

This document describes what we tested, how we tested, and what we found. To learn how these facts translate into real-world benefits, read the report Improve Aerospike Database performance and predictability by leveraging Intel® Ethernet 800 Series Network Adapters with Application Device Queues (ADQ).

We concluded our hands-on testing on February 26, 2021. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on February 10, 2021 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

To learn more about how we have calculated the wins in this report, go to http://facts.pt/calculating-and-highlighting-wins. Unless we state otherwise, we have followed the rules and principles we outline in that document.

Table 1: Aerospike database read throughput (in transactions per second) and scaling comparison with ADQ enabled and ADQ off (NUMA pinning) for multiple server node and memory configurations. We present the median result of three runs. Higher numbers are better. Source: Principled Technologies.

Server nodes	Total instances	Clients	Type	ADQ status	TPS	Scaling
1	2	8	Memory-only	ADQ-on	15,051,776	1.00
2	4	8	Memory-only	ADQ-on	29,169,294	1.94
3	6	8	Memory-only	ADQ-on	43,217,907	2.87
1	2	8	Memory-only	ADQ-off	10,254,738	1.00
2	4	8	Memory-only	ADQ-off	17,971,014	1.75
3	6	8	Memory-only	ADQ-off	26,219,536	2.56
1	2	8	PMem	ADQ-on	13,850,155	1.00
2	4	8	PMem	ADQ-on	26,933,812	1.94
3	6	8	PMem	ADQ-on	39,671,664	2.86
1	2	8	PMem	ADQ-off	9,829,639	1.00
2	4	8	PMem	ADQ-off	17,633,166	1.79
3	6	8	PMem	ADQ-off	25,952,236	2.64

Table 2: Weighted average latency calculations for one server using DRAM only. Source: Principled Technologies.

Weighted average latency calculations (1 server, DRAM only)											
	Latency thresholds in microseconds										
	>0us	>40us	>80us	>120us	>160us	>200us	>240us	>280us	>320us	>360us	>400us
Cumulative bucket percentages	100.0	98.2	81.8	60.0	46.7	30.0	7.3	0.4	0.0	0.0	0.0
	100.0	100.0	99.4	86.2	70.6	55.8	38.0	23.2	13.7	7.9	3.8
Interval midpoint	20	60	100	140	180	220	260	300	340	380	420
Percentage of total samples per bucket	1.8	16.3	21.9	13.3	16.7	22.6	6.9	0.3	0.0	0.0	0.0
	0.0	0.6	13.2	15.6	14.8	17.8	14.8	9.5	5.8	4.1	2.1

	Latency thresholds in microseconds										Weighted average latency (P50)
	>440us	>480us	>520us	>560us	>600us	>640us	>680us	>720us	>760us	>800us	
Cumulative bucket percentages	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	149.9
	1.7	0.8	0.3	0.1	0.1	0.0	0.0	0.0	0.0	0.0	220.6
Interval midpoint	460	500	540	580	620	660	700	740	780	820	Percent lower weighted average latency with ADQ
Percentage of total samples per bucket	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
	1.0	0.	0.2	0.1	0.1	0.0	0.0	0.0	0.0	0.0	32.0

Table 3: Weighted average latency calculations for one server using Intel Optane PMem. Source: Principled Technologies.

Weighted average latency calculations (1 server, Intel Optane PMem)											
	Latency thresholds in microseconds										
	>0us	>40us	>80us	>120us	>160us	>200us	>240us	>280us	>320us	>360us	>400us
Cumulative bucket percentages	100.0	98.6	87.5	69.0	51.8	35.3	13.9	1.6	0.9	0.0	0.0
	100.0	99.2	88.9	75.0	64.6	55.6	43.9	32.3	21.9	15.8	11.2
Interval midpoint	20	60	100	140	180	220	260	300	340	380	420
Percentage of total samples per bucket	1.4	11.1	18.5	17.2	16.6	21.4	12.3	1.5	0.0	0.0	0.0
	0.8	10.3	13.9	10.4	9.0	11.7	11.6	10.4	6.1	4.7	3.9

	Latency thresholds in microseconds										Weighted average latency (P50)
	>440us	>480us	>520us	>560us	>600us	>640us	>680us	>720us	>760us	>800us	
Cumulative bucket percentages	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	163.2
	7.2	5.0	2.7	1.5	0.8	0.0	0.0	0.0	0.0	0.0	230.3
Interval midpoint	460	500	540	580	620	660	700	740	780	820	Percent lower weighted average latency with ADQ
Percentage of total samples per bucket	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
	2.2	2.3	1.2	0.7	0.8	0.0	0.0	0.0	0.0	0.0	29.1

ADQ-on ADQ-off

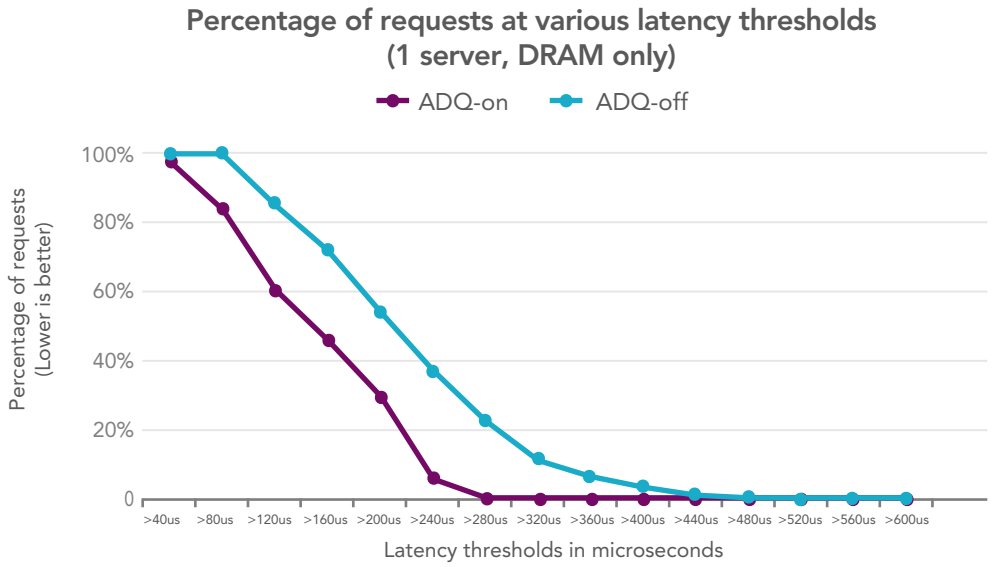

Figure 1: Latency histogram buckets for a single Dell EMC PowerEdge R740xd server node (DRAM only) with Intel Ethernet 800 Series Network Adapters with ADQ on and ADQ off. Source: Principled Technologies.

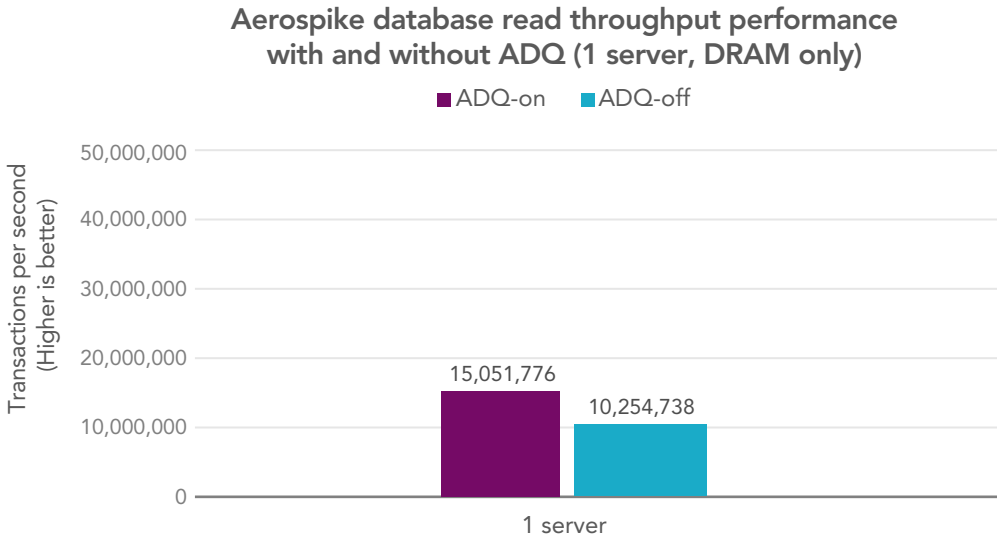

Figure 2: Aerospike database read throughput performance, in transactions per second, for a single Dell EMC PowerEdge R740xd server node (DRAM only) with Intel Ethernet 800 Series Network Adapters with ADQ on and ADQ off. Source: Principled Technologies.

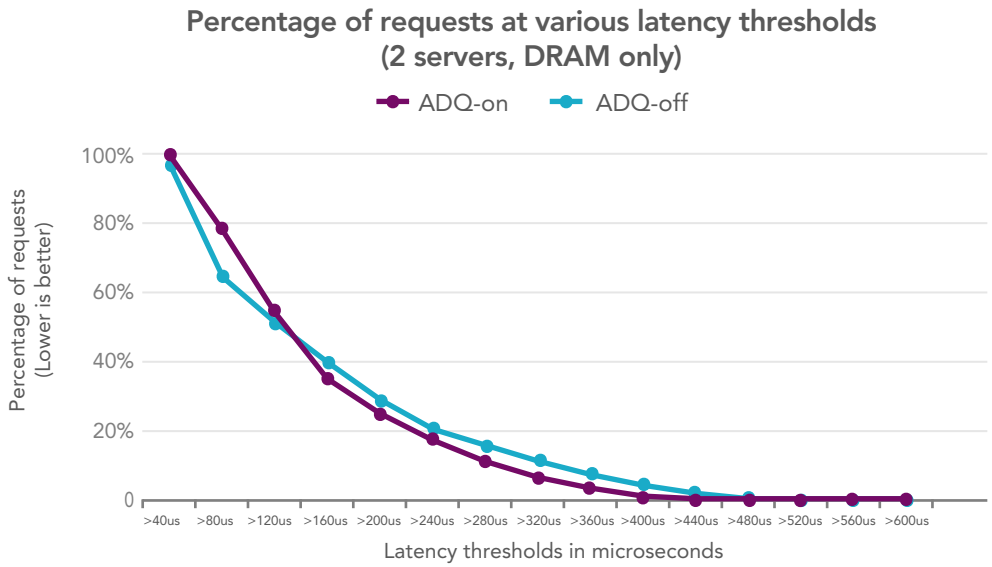

Figure 3: Latency histogram buckets for two Dell EMC PowerEdge R740xd server nodes (DRAM only) with Intel Ethernet 800 Series Network Adapters with ADQ on and ADQ off. Source: Principled Technologies.

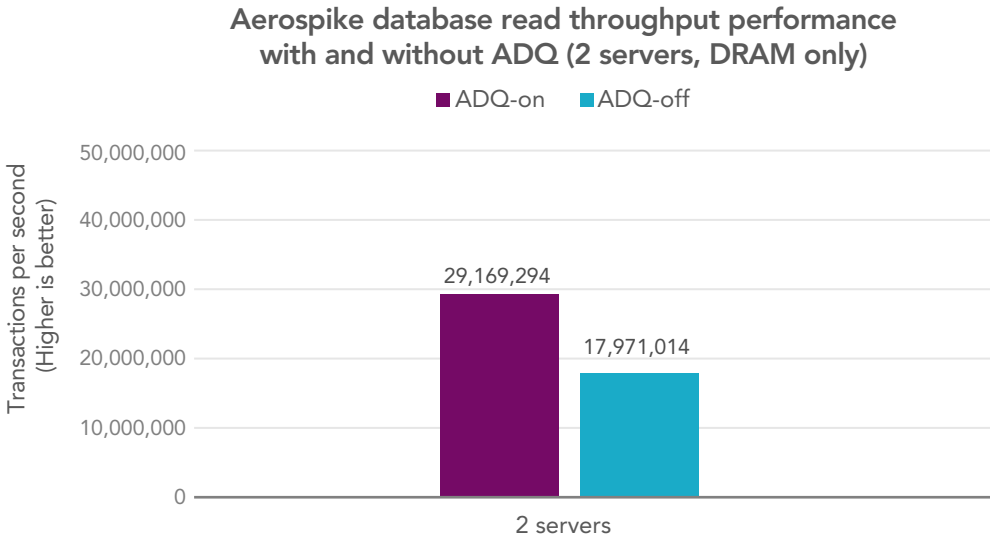

Figure 4: Aerospike database read throughput performance, in transactions per second, for two Dell EMC PowerEdge R740xd server nodes (DRAM only) with Intel Ethernet 800 Series Network Adapters with ADQ on and ADQ off. Source: Principled Technologies.

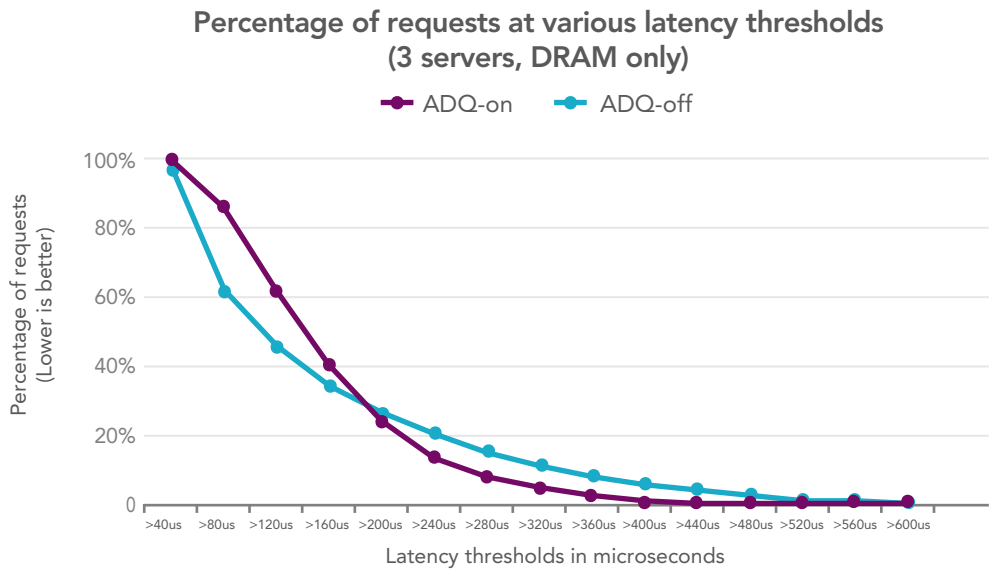

Figure 5: Latency histogram buckets for three Dell EMC PowerEdge R740xd server nodes (DRAM only) with Intel Ethernet 800 Series Network Adapters with ADQ on and ADQ off. Source: Principled Technologies.

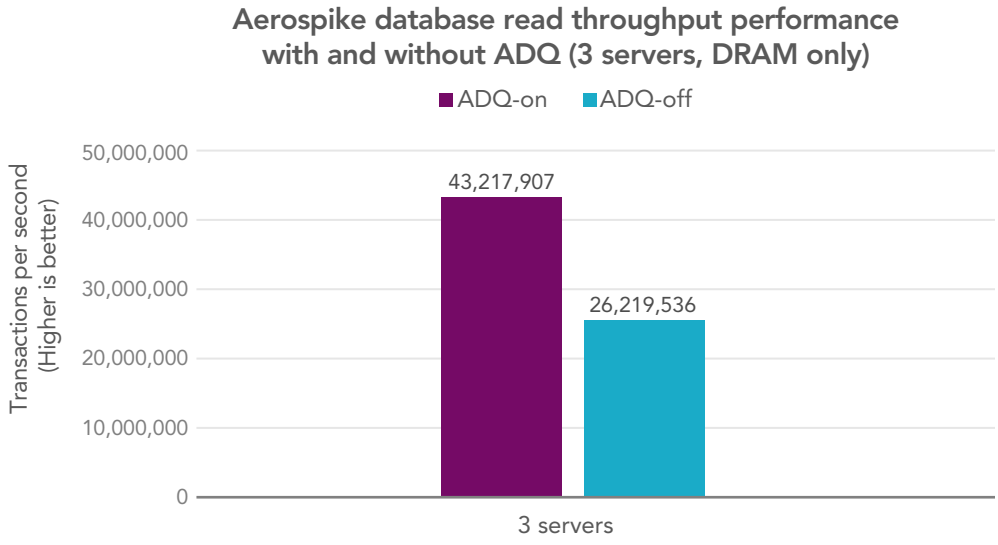

Figure 6: Aerospike database read throughput performance, in transactions per second, for three Dell EMC PowerEdge R740xd server nodes (DRAM only) with Intel Ethernet 800 Series Network Adapters with ADQ on and ADQ off. Source: Principled Technologies.

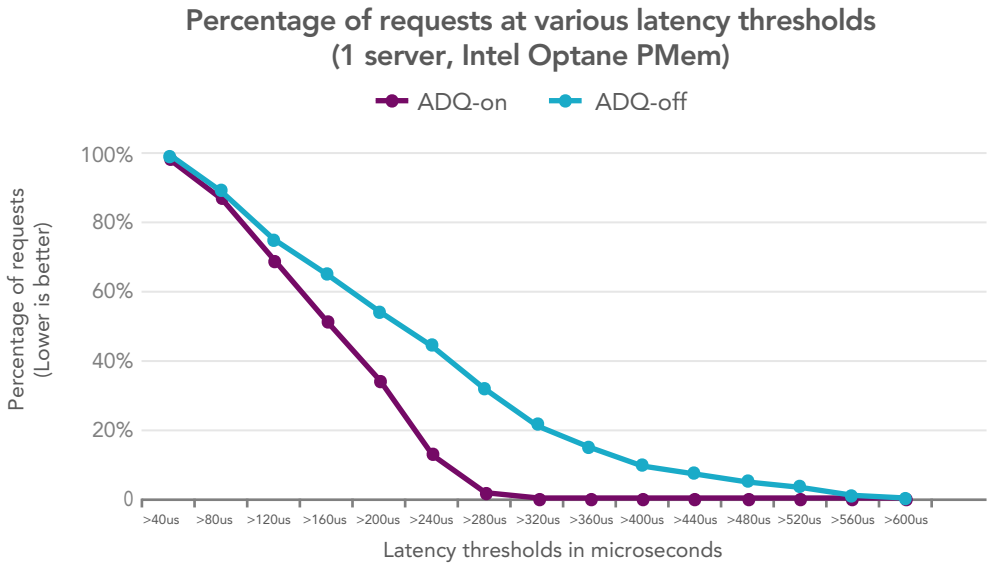

Figure 7: Latency histogram buckets for a single Dell EMC PowerEdge R740xd server node (with Intel Optane PMem) with Intel Ethernet 800 Series Network Adapters with ADQ on and ADQ off. Source: Principled Technologies.

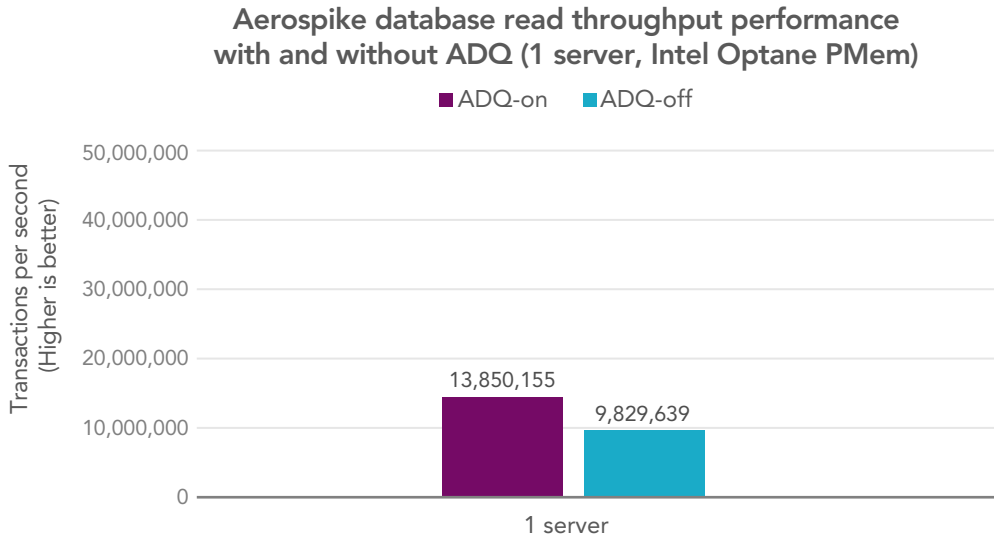

Figure 8: Aerospike database read throughput performance, in transactions per second, for a single Dell EMC PowerEdge R740xd server node (with Intel Optane PMem) with Intel Ethernet 800 Series Network Adapters with ADQ on and ADQ off. Source: Principled Technologies.

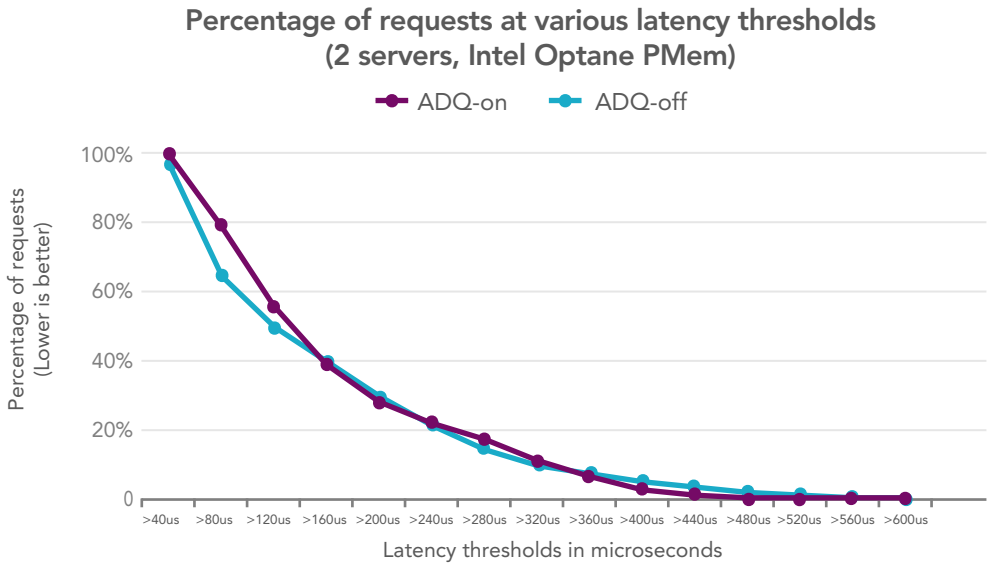

Figure 9: Latency histogram buckets for two Dell EMC PowerEdge R740xd server nodes (with Intel Optane PMem) with Intel Ethernet 800 Series Network Adapters with ADQ on and ADQ off. Source: Principled Technologies.

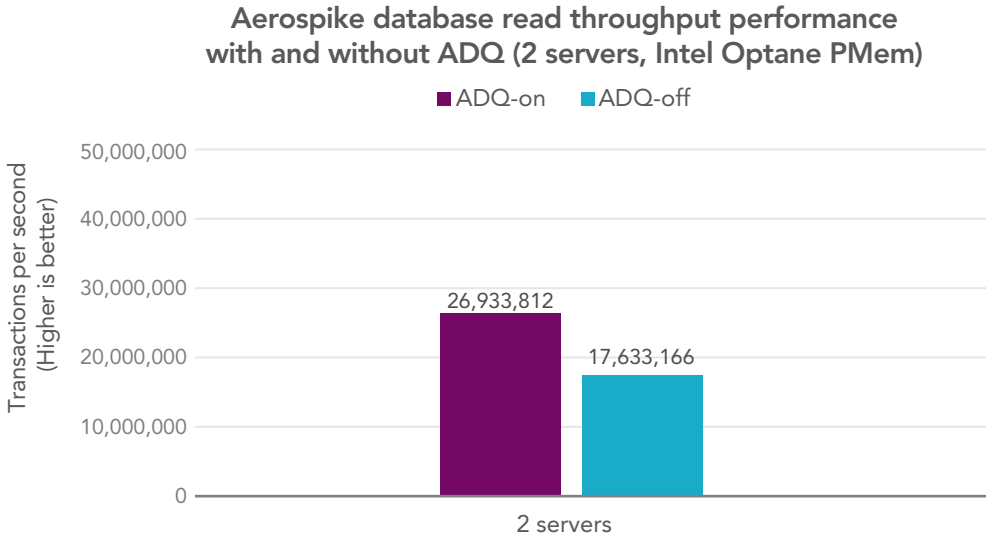

Figure 10: Aerospike database read throughput performance, in transactions per second, for two Dell EMC PowerEdge R740xd server nodes (with Intel Optane PMem) with Intel Ethernet 800 Series Network Adapters with ADQ on and ADQ off. Source: Principled Technologies.

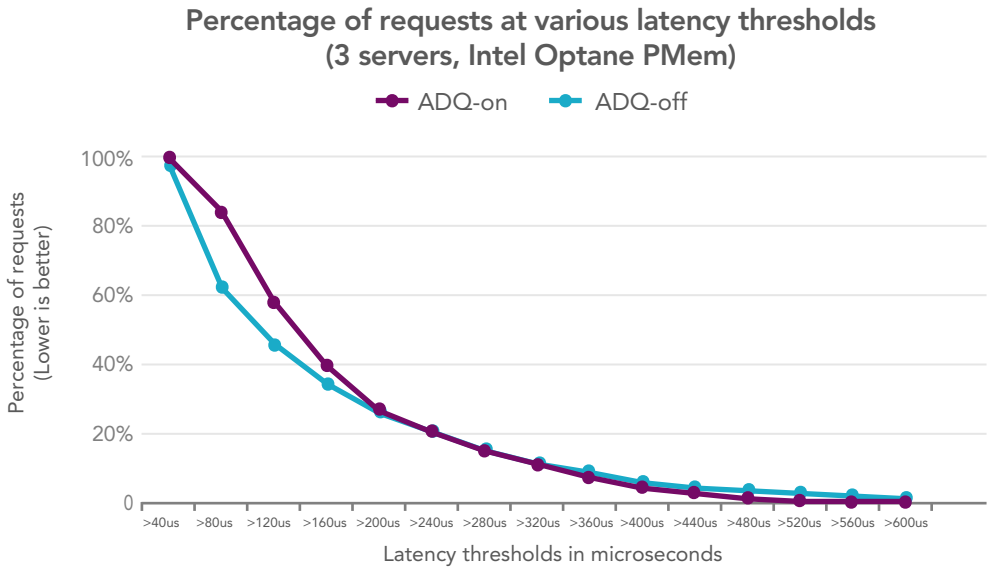

Figure 11: Latency histogram buckets for three Dell EMC PowerEdge R740xd server nodes (with Intel Optane PMem) with Intel Ethernet 800 Series Network Adapters with ADQ on and ADQ off. Source: Principled Technologies.

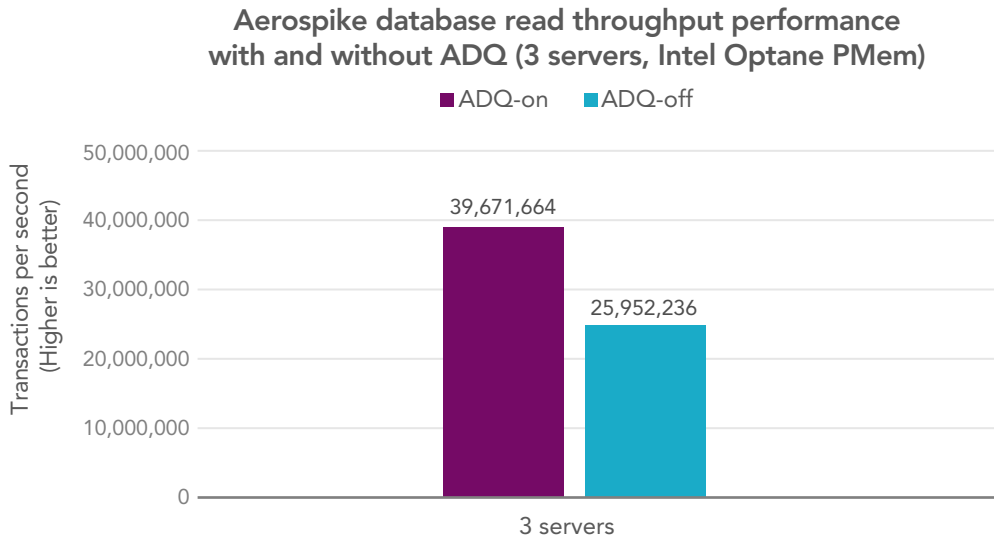

Figure 12: Aerospike database read throughput performance, in transactions per second, for three Dell EMC PowerEdge R740xd server nodes (with Intel Optane PMem) with Intel Ethernet 800 Series Network Adapters with ADQ on and ADQ off. Source: Principled Technologies.

Note that for all ADQ-enabled results, the system(s) provide lower latencies than with ADQ disabled while simultaneously achieving significantly higher throughput—handling up to 64 percent more transactions per second. The latency/predictability numbers are better for ADQ enabled in all configurations, and show the biggest response time predictability improvement for a single node. Subsequent predictability charts (at two and three nodes) show ADQ enabled and ADQ disabled numbers to be closer to each other—but the ADQ-enabled servers handle a particularly high load at this scale and are still able to provide better latency predictability than with ADQ disabled.

System configuration information

For our client systems we used eight dual-socket white box servers, each with 2nd Generation Intel® Xeon® Scalable processors, 384 GB of RAM, and a two-port 25 GbE NIC. Table 2 details our servers under test.

Table 4: Detailed information for our test systems.

Server configuration information	3 x Dell EMC™ PowerEdge™ R740xd
BIOS name and version	Dell 2.10.0
Non-default BIOS settings	N/A
Operating system name and version/build number	CentOS 8.3 Kernel 4.18.0-240.1.1.el8_3.x86_64
Date of last OS updates/patches applied	1/15/2021
System profile settings	Performance
Processor	
Number of processors	2
Vendor and model	Intel Xeon Gold 6258R
Core count (per processor)	28
Core frequency (GHz)	2.7
Stepping	Model 85 Stepping 7
Memory module(s)	
Total memory in system (GB)	384
Number of memory modules	12
Vendor and model	Samsung® M393A4K40BB1-CRC
Size (GB)	32
Type	DDR-4
Speed (MHz)	2,400
Speed running in the server (MHz)	2,400
Persistent memory module(s)	
Total memory in system (GB)	1,536
Number of memory modules	12
Vendor and model	Intel NMA1XXD128GPS
Size (GB)	128
Type	Intel® Optane™ Persistent Memory (PMem)
Speed (MHz)	2,666
Speed running in the server (MHz)	2,400

Server configuration information	3 x Dell EMC™ PowerEdge™ R740xd
Local storage	
Number of drives	1
Drive vendor and model	Intel SSDPE2ME020T4D
Drive size (GB)	2,000
Drive information (interface, type)	NVMe PCIe SSD
Network adapter	
Vendor and model	2x Intel Ethernet Network Adapter E810-CQDA2
Number and type of ports	4 total ports (2x dual-port 100GbE)
Cooling fans	
Vendor and model	Nidec® UltraFlo4VXP3-X30
Number of cooling fans	6
Power supplies	
Vendor and model	Dell 0PJMDN
Number of power supplies	2
Wattage of each (W)	750
Aerospike C Benchmark configuration parameters	
MTU	1500
Object specification	String of length: 32
Workload	100% Random read

How we tested

Changing BIOS System Profile to Performance

1. Power on each server.
2. Enter BIOS, and change the System Profile to Performance.
3. Repeat for other servers.

Installing CentOS Linux 8.3

On each server, boot to the CentOS Linux 8.3 installation media using the CentOS-8.3.2011-x86_64-boot.iso image.

1. Select Install CentOS Linux 8.
2. Choose English, and click Continue.
3. Under Installation Destination, select the desired disk to install the OS.
4. Under Storage Configuration, select Custom, and click Done.
5. Select Click here to create them automatically.
6. If it exists, remove the /home partition.
7. Expand the swap partition to 4GB.
8. Assign all remaining free space to the / partition.
9. Click Done.
10. Click Accept Changes.
11. Select Network & Hostname.
12. Enter the desired hostname for the system.
13. Turn on the desired network ports, and click Configure.
14. On the General tab, select Connect automatically with priority.
15. On the IPv4 Settings tab, choose the drop-down named Method, and select Manual.
16. Under Addresses, click Add, and enter the desired static IP information for the server.
17. Enter the desired DNS information.
18. Click Save, and click Done.
19. Select Date & Time, and ensure the correct date, time, and time zone are set.
20. Click the cog next to the Network Time On/Off switch to add your NTP server.
21. Add the IP address of your NTP server, and click +.
22. Uncheck all other NTP servers.
23. Click OK.
24. Click Done.
25. Click Software Selection
26. Choose the Base Environment of Minimal Install.
27. Click Done.
28. Click Begin Installation.
29. Select Root Password.
30. Enter the desired root password, and click Done.
31. When the installation completes, select Reboot to restart the server.
32. Repeat for other servers.

Configuring firewall and passwordless SSH

On each server, log onto the server as root.

1. Disable the firewall and SELinux by typing the following commands:

```
setenforce 0
sed -i 's/SELINUX=.*/SELINUX=disabled/' /etc/selinux/config
systemctl disable --now firewalld
```

2. On the first server, configure passwordless SSH:

```
rm -rf ~/.ssh
mkdir -p ~/.ssh
chmod 700 ~/.ssh
cd ~/.ssh
# Hit Enter for any prompts
ssh-keygen -t rsa -q

cp id_rsa.pub authorized_keys
echo "StrictHostKeyChecking=no" > config
echo > known_hosts
cp -rp ~/.ssh ~/clean_ssh
cd ~/clean_ssh
```

3. Copy to other hosts:

```
scp -rp ~/clean_ssh remotehost:~/.ssh
```

4. Install the EPEL Repository by typing the following into a terminal window:

```
dnf install -y epel-release
```

5. Update the host:

```
dnf update -y
```

6. Reboot the host.
7. After reboot, install tools:

```
dnf install -y wget vim tar zip unzip numactl sysstat nmon ksh pciutils ipmitool smartmontools
```

Installing software, NIC driver, and NIC firmware

1. Install the development tools software by Issuing the following commands:

```
dnf groupinstall -y "Development Tools"
dnf install -y kernel-modules-extra elfutils-libelf-devel iproute-tc libcgroup libcgroup-tools
```

2. Download ice-1.2.1.tar.gz from the Intel website.
3. Extract the contents of the file to both the servers under test and the client machines.

```
tar -xf ice-1.2.1.tar.gz
cd ice-1.2.1/src/
```

4. On the server under test, build and install the driver with ADQ support enabled by issuing the following command:

```
make -j$(nproc) CFLAGS_EXTRA="-DADQ_PERF -DADQ_PERF_COUNTERS" install
```

5. On the client systems, build and install the driver with default settings by issuing the following command:

```
make -j$(nproc) install
```

6. Reload the driver by issuing the following command:

```
modinfo ice
modprobe -r ice
modprobe ice
```

7. Update the NIC firmware on the servers containing the E810 NICs by issuing the following commands:

```
cd NVMUpdatePackage/E810/
tar -xf E810_NVMUpdatePackage_v2_22_Linux.tar.gz
cd E810/Linux_x64/
./nvmupdate64e -u -l -o update.xml -b -c nvmupdate.cfg -rd
reboot
```

Configuring networking on each server under test

1. Add two lines to the bottom of /etc/iproute2/rt_tables:

```
cat <<EOF >>/etc/iproute2/rt_tables
100  t1
101  t2
EOF
```

2. Use the following script as a guide to assign IP addresses to NICs and configure routes:

```
# Assign IP addresses on interfaces.
# For our servers, these interfaces are:
 ens1f0 and
ens8f0

IPNUM_PREFIX=1  # 1 = server1, 2 = server2, etc...
IFNAME=ens1f0
IPNUM=${IPNUM_PREFIX}1
echo "net.ipv4.conf.${IFNAME}.arp_ignore=1" >> /etc/sysctl.conf
echo "net.ipv4.conf.${IFNAME}.arp_announce=2" >> /etc/sysctl.conf
nmcli connection delete ${IFNAME}
nmcli connection add type ethernet con-name ${IFNAME} ifname ${IFNAME} ipv4.method manual ipv4.
addresses 192.168.40.${IPNUM}/24 ipv4.routes "192.168.40.0/24 src=192.168.40.${IPNUM} table=100"
ipv4.routing-rules "priority 100 from 192.168.40.${IPNUM} table 100"

IFNAME=ens8f0
IPNUM=${IPNUM_PREFIX}2
echo "net.ipv4.conf.${IFNAME}.arp_ignore=1" >> /etc/sysctl.conf
echo "net.ipv4.conf.${IFNAME}.arp_announce=2" >> /etc/sysctl.conf
nmcli connection delete ${IFNAME}
nmcli connection add type ethernet con-name ${IFNAME} ifname ${IFNAME} ipv4.method manual ipv4.
addresses 192.168.40.${IPNUM}/24 ipv4.routes "192.168.40.0/24 src=192.168.40.${IPNUM} table=101"
ipv4.routing-rules "priority 101 from 192.168.40.${IPNUM} table 101"

# Finish setting up routing with multiple NICs on same subnet
sysctl -p
sysctl -a | grep arp_
ip -s -s neigh flush all

echo "** Routes **"
ip route show
echo "** Rules **"
ip rule show
echo "** Tables **"
ip route show table t1
ip route show table t2
```

Configuring networking on clients

1. Use the following script as a guide to assign IP addresses to NICs and configure routes:

```
IPNUM_PREFIX=1  # 1 = client1, 2 = client2, etc...
IFNAME=ens786f0
nmcli connection delete ${IFNAME}
nmcli connection add type ethernet con-name ${IFNAME} ifname ${IFNAME} ipv4.method manual ipv4.
addresses 192.168.40.$((IPNUM_PREFIX+100))/24
reboot
```

Installing Aerospike, creating working directories, and assigning licenses

1. Install Aerospike on each server by issuing the following commands:

```
dnf install -y python3
mkdir aerospike
cd aerospike
wget -O aerospike.tgz 'https://www.aerospike.com/enterprise/download/server/latest/artifact/el8'
tar -xvf aerospike.tgz
cd aerospike-server-enterprise-*-el8
sudo ./asinstall
```

2. Create working directories:

```
rm -rf /opt/aerospike-inst1
rm -rf /opt/aerospike-inst2
mkdir -p /opt/aerospike-inst1
mkdir -p /opt/aerospike-inst2
cp -rfp /opt/aerospike/* /opt/aerospike-inst1/
cp -rfp /opt/aerospike/* /opt/aerospike-inst2/
cp /etc/aerospike/aerospike.conf aerospike1.conf
cp /etc/aerospike/aerospike.conf aerospike2.conf
```

3. Install the Aerospike license as needed.

Configuring Intel Optane PMem (PMem configurations only)

We configured PMem on only a subset of our tests. For those sections of testing, we configured PMem by creating a namespace in fsdax mode, built an XFS filesystem, then mounted the PMem volumes. We did this on each of the three servers under test.

1. List the regions to confirm parameters:

```
ndctl list --regions
```

Example output:

```
[
  {
    "dev":"region1",
    "size":811748818944,
    "available_size":811748818944,
    "max_available_extent":811748818944,
    "type":"pmem",
    "iset_id":-8672667031271953204,
    "persistence_domain":"memory_controller"
  },
  {
    "dev":"region0",
    "size":811748818944,
    "available_size":811748818944,
    "max_available_extent":811748818944,
    "type":"pmem",
    "iset_id":-500885507041579828,
    "persistence_domain":"memory_controller"
  }
]
```

2. Create the namespaces:

```
ndctl create-namespace --mode=fsdax --region=region0
ndctl create-namespace --mode=fsdax --region=region1
```

Example output:

```
{
  "dev":"namespace0.0",
  "mode":"fsdax",
  "map":"dev",
  "size":"744.19 GiB (799.06 GB)",
  "uuid":"799a5566-4a74-4ba7-9c17-b4b541359c02",
  "sector_size":512,
  "align":2097152,
  "blockdev":"pmem0"
}
```

3. Build the filesystem on each pmem device:

```
mkfs.xfs -m reflink=0 -d su=2m,sw=1 /dev/pmem0
mkfs.xfs -m reflink=0 -d su=2m,sw=1 /dev/pmem1
```

4. Mount the filesystems:

```
mkdir -p /mnt/pmem0
mkdir -p /mnt/pmem1
echo "$(blkid /dev/pmem0 | cut -d' ' -f2) /mnt/pmem0 xfs dax 0 0" >> /etc/fstab
echo "$(blkid /dev/pmem1 | cut -d' ' -f2) /mnt/pmem1 xfs dax 0 0" >> /etc/fstab
mount -v /mnt/pmem0
mount -v /mnt/pmem1
```

Configuring the benchmark

To configure the benchmark tool, we followed the procedure below. We applied a patch to the available code to expose microsecond latencies, called asd-client-bench_v5.1.0_usecfloatlatency.patch below.

1. Install and configure the benchmark tool and patch by issuing the following commands:

```
dnf install -y openssl-devel libev-devel
git clone --branch 5.1.0 https://github.com/aerospike/aerospike-client-c
cd aerospike-client-c/
git apply asd-client-bench_v5.1.0_usecfloatlatency.patch
git submodule update --init
make EVENT_LIB=libev
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
cd benchmarks
make EVENT_LIB=libev
cd target
cp -p benchmarks ~/asd-bench
```

2. Copy the compiled asd-bench tool to the /root folder of each client.

Initializing the database and example test run

1. Install and configure the benchmark tool and patch by issuing the following commands:

```
# Example database init
./benchmarks -h 192.168.40.11 -k 100000000 -o S:32 -w I -a -c 168 -W 28 --latency 10,2,us
# When using replication, tell it not to wait for commit all
./asd-bench -h 192.168.40.11:3000 -k 1000000000 -o S:32 -w I -a -c 168 -W 28 --latency 10,2,u -M
master
```

2. Use the following as a guide to run the tests:

```
./asd-bench -h 192.168.40.11 -k 100000000 -o S:32 -w RU,100 -a -c 142 -W 12 --latency 16,4
```

Scripts and configuration files

aerospike1.conf

```
# aerospike1.conf : Aerospike configuration file for instance 1
#
service {
    run-as-daemon false
    work-directory /opt/aerospike-inst1
    paxos-single-replica-limit 1 # Number of nodes where the replica count is automatically reduced
to 1.
    proto-fd-max 15000
    auto-pin adq # adq or numa
    #node-id-interface ens1f0
    microsecond-histograms true
    ticker-interval 1
}

logging {
    file /var/log/aerospike1.log {
            context any info
    }
    console {
            context any info
    }
}

mod-lua {
    user-path /opt/aerospike-inst1/usr/udf/lua
}

network {
    service {
            address ens1f0
            port 3000
            access-address ens1f0
    }

    heartbeat {
            address ens1f0
            mode mesh
            port 3010
            mesh-seed-address-port 192.168.40.11 3010
            mesh-seed-address-port 192.168.40.12 4010
            mesh-seed-address-port 192.168.40.21 3010
            mesh-seed-address-port 192.168.40.22 4010
            mesh-seed-address-port 192.168.40.31 3010
            mesh-seed-address-port 192.168.40.32 4010
            interval 150
            timeout 10
    }

    fabric {
            address ens1f0
            port 3011
    }

    info {
            address ens1f0
            port 3012
    }
}

namespace test {
```

```
    replication-factor 2
    memory-size 128G
    default-ttl 0 # 30 days, use 0 to never expire/evict.
    nsup-period 86400
    #enable-benchmarks-read true

    storage-engine memory
#   storage-engine pmem {
#           file /mnt/pmem0/data000
#           file /mnt/pmem0/data001
#           file /mnt/pmem0/data002
#           file /mnt/pmem0/data003
#           filesize 128G
#   }
}
```

aerospike2.conf

```
# aerospike2.conf : Aerospike configuration file for instance 2
#
service {
    run-as-daemon false
    work-directory /opt/aerospike-inst2
    paxos-single-replica-limit 1 # Number of nodes where the replica count is automatically reduced
to 1.
    proto-fd-max 15000
    auto-pin adq # adq or numa
    #node-id-interface ens8f0
    microsecond-histograms true
    ticker-interval 1
}

logging {
    file /var/log/aerospike2.log {
            context any info
    }
    console {
            context any info
    }
}

mod-lua {
    user-path /opt/aerospike-inst2/usr/udf/lua
}

network {
    service {
            address ens8f0
            port 4000
            access-address ens8f0
    }

    heartbeat {
            address ens8f0
            mode mesh
            port 4010
            mesh-seed-address-port 192.168.40.11 3010
            mesh-seed-address-port 192.168.40.12 4010
            mesh-seed-address-port 192.168.40.21 3010
            mesh-seed-address-port 192.168.40.22 4010
            mesh-seed-address-port 192.168.40.31 3010
            mesh-seed-address-port 192.168.40.32 4010
            interval 150
            timeout 10
    }
```

```
    fabric {
            address ens8f0
            port 4011
    }

    info {
            address ens8f0
            port 4012
    }
}

namespace test {
    replication-factor 2
    memory-size 128G
    default-ttl 0 # 30 days, use 0 to never expire/evict.
    nsup-period 86400
    #enable-benchmarks-read true

    storage-engine memory
#   storage-engine pmem {
#           file /mnt/pmem1/data100
#           file /mnt/pmem1/data101
#           file /mnt/pmem1/data102
#           file /mnt/pmem1/data103
#           filesize 128G
#   }
}
```

systune.sh

```
#!/bin/bash
# systune.sh : master ADQ configuration

CHK () {
"$@"
if [ $? -ne 0 ]; then
echo "Error with ${1}, stopping now!" >&2
exit 1
fi
}

service firewalld stop
systemctl mask firewalld
tuned-adm profile throughput-performance
cat /etc/tuned/active_profile
cat /etc/tuned/profile_mode
x86_energy_perf_policy performance

CHK sysctl -w net.core.somaxconn=4096
CHK sysctl -w net.core.netdev_max_backlog=8192
CHK sysctl -w net.core.rmem_max=16777216
CHK sysctl -w net.core.wmem_max=16777216
CHK sysctl -w net.ipv4.tcp_max_syn_backlog=16384
CHK sysctl -w net.ipv4.tcp_mem="764688 1019584 16777216"
CHK sysctl -w net.ipv4.tcp_rmem="8192 87380 16777216"
CHK sysctl -w net.ipv4.tcp_wmem="8192 65536 16777216"

ulimit -n 4096
```

setup-adq.sh

```
#!/bin/bash
# setup-adq.sh : master ADQ configuration
#
# References for further reading:
# * Ethernet Flow Director
# * Link Layer Discovery Protocol
# * Busy Polling
# * tc(8) Linux man page
# * MQPRIO(8) Linux man page
# * tc-flower(8) Linux man page
# * irqbalance(1) Linux man page
# * Adaptive Interrupt Moderation

# Input parameters (and defaults if not specified)
iface=${1:-eth2}   # NIC to be configured
queues=${2:-56}            # Should be the number of vCPUs
port=${3:-3000}            # Aerospike server service port
rx=${4:-0}                 # rx-usecs value or "on" (for Adaptive Interrupt Moderation)
tx=${5:-500}               # tx-usecs value or "on" (for Adaptive Interrupt Moderation)
bp=${6:-50000}             # busy polling interval in microseconds
loadit=${7:-0}             # load ice driver

pathtoiproute2="/usr" # replacement for ifconfig & route commands

if [ $loadit -eq 1 ]; then
    echo "Loading driver"
    modprobe -vr ice # Remove Intel 'ice' kernel module (incompatible with ADQ)
    sleep 2
    modprobe -v ice # Verify removed
    ifup ${iface} # Bring up NIC
fi

# Extract IP address of NIC
addr=$(ip -f inet -o addr show dev $iface|cut -d\  -f7|cut -d/ -f1)
echo $addr

# Some commands fail silently, so it is imperative to explicitly check
# exit status by running them with this script rather than directly.
CHK () {
    "$@"
    if [ $? -ne 0 ]; then
            echo "Error with ${1}, stopping now!" >&2
            exit 1
    fi
}

# -set-priv-flags is a mechanism for ethtool to pass vendor-specific
# configuration to a NIC. As a corollary, such private flags will only
# be effective with specific NICs (not only vendor-specific, but also
# within a model family).

# Enable the Intel Ethernet Flow Director. This is a mechanism for
# directing ethernet packets to the same CPU core as the consuming
# application. It is a prerequisite for ADQ.
CHK ethtool --set-priv-flags $iface channel-inline-flow-director on

# Disable processing of LLDP packets by the NIC firmware. Link Layer
# Discovery Protocol implements the IEEE 802.1AB standard: it is used by
# NICs to advertise link layer identity and capabilities. It can
# interfere with ADQ, which is why it is disabled.
CHK ethtool --set-priv-flags $iface fw-lldp-agent off

# Enable hardware-based network traffic classification (TC). Linux has
```

```
# a software-based traffic classifier; with ADQ, the 800-series NICs
# take over this function.
CHK ethtool --offload $iface hw-tc-offload on

# Enable busy polling. This reduces interrupts by having socket code
# poll receive queue of the NIC. Under high-traffic situations this
# is much more performant than depending on interrupts to process
# incoming packets.
CHK sysctl -w net.core.busy_poll=$bp

# Add a queueing discipline (mqprio) to the root of the filter chain
# of the specified NIC ($iface). The mqprio discipline supports
# multi-queue hardware QoS, and maps packets to device queues of the
# NIC according to the parameters at the end of the command. In this
# instance, are 2 traffic classes defined (num_tc), with priorities 0
# and 1 (map). The first two queues (2@0) are in traffic class 0, and
# by default queues 2-57 ($queues@2) are in traffic class 1. This
# function is supported in hardware by the NIC (hw), channel mode
# (channel) is enabled. The traffic classes are used in the filter
# command below.
CHK ${pathtoiproute2}/sbin/tc qdisc add dev $iface root mqprio num_tc 2 map 0 1 queues 2@0 $queues@2
hw 1 mode channel

# Add the ingress queuing discipline to handle incoming packets to the
# specified NIC ($iface).
CHK ${pathtoiproute2}/sbin/tc qdisc add dev $iface ingress

# Due to timing issues, applying TC filters immediately after the tc
# qdisc add command might result in the filters not being offloaded in
# hardware. An error in dmesg is logged if the filter fails to add
# properly. It is recommended to wait five seconds after tc qdisc add
# before adding TC filters.
sleep 5

# Add a flow-based filter (flower) to classify incoming traffic to the
# specified NIC ($iface). The filter is attached to the ingress queue
# discipline (indicated by the reserved handle ffff:0). The filter
# handles TCP/IP (protocol; ip_proto) packets in traffic class (prio)
# 1, whose destination IP (dst_ip) is $addr on port (dst_port, the
# service port of the Aerospike server). The packets are passed on to
# the hardware traffic class 1 (hw_tc), and are processed by the NIC
# (skip_sw).
CHK ${pathtoiproute2}/sbin/tc filter add dev $iface protocol ip parent ffff: prio 1 flower dst_ip
$addr/32 ip_proto tcp dst_port $port skip_sw hw_tc 1

# Disable adaptive interrupt moderation for both incoming and outgoing
# packets. This is enabled by default, and it interferes with ADQ
# busy polling. rx-usecs is the minimum time to wait before raising
# an interrupt after a packet has been received. If 0, rx-frames is
# used. tx-usecs is the time to wait before raising a TX interrupt
# after a packet has been sent.
[ "$rx" != "on" ] && ethtool --coalesce $iface adaptive-rx off rx-usecs $rx
[ "$tx" != "on" ] && ethtool --coalesce $iface adaptive-tx off tx-usecs $tx

# Stop the irqbalance daemon. This daemon attempts to distribute
# interrupt requests evenly across CPU cores. It interferes with the
# ADQ mechanism.
systemctl stop irqbalance

# Affinitize NIC interrupts to cores + Set XPS (Transmit Packet Steering) maps (-x)
CHK ./set_irq_affinity -x local $iface

# Affinitize XPS queues
CHK ./set_xps_rxqs $iface
```

```
# Display post-configuration data for review
ethtool --show-channels $iface
ethtool --show-coalesce $iface
ethtool --show-ring $iface
ethtool --show-priv-flags $iface
```

asd-startup.sh

```
#!/bin/bash
# asd-startup.sh : Affinitize interrupts to cores
ulimit -n 100000

# modify as needed
server1="asd --config-file ./aerospike1.conf --foreground --instance 0"
server2="asd --config-file ./aerospike2.conf --foreground --instance 1"

# create & set cgroup

cgdelete -g net_prio:asd
cgcreate -g net_prio:asd
cgset -r net_prio.ifpriomap="ens1f0 1" asd
cgset -r net_prio.ifpriomap="ens8f0 1" asd

# start server in the foreground
cgexec -g net_prio:asd --sticky numactl --cpunodebind=netdev:ens1f0 --membind=netdev:ens1f0 $server1
&> $(hostname -s)_aerospike1.log &
cgexec -g net_prio:asd --sticky numactl --cpunodebind=netdev:ens8f0 --membind=netdev:ens8f0 $server2
&> $(hostname -s)_aerospike2.log &
```

asd-startup-numa.sh

```
#!/bin/bash
#
# asd-startup.sh : Affinitize interrupts to cores
ulimit -n 100000

# modify as needed
server1="asd --config-file ./aerospike1.conf --foreground --instance 0"
server2="asd --config-file ./aerospike2.conf --foreground --instance 1"


cgdelete -g net_prio:asd
numactl --cpunodebind=netdev:ens1f0 --membind=netdev:ens1f0 $server1 &> $(hostname -s)_aerospike1.log
&
numactl --cpunodebind=netdev:ens8f0 --membind=netdev:ens8f0 $server2 &> $(hostname -s)_aerospike2.log
&
```

run_test.sh

```
#!/bin/bash
TIMESTAMP=$(date '+%Y%m%d_%H%M%S')
TARGET_HOSTS="192.168.40.11:3000"
SERVERS="$(echo server{1..3})"

#CLIENT_INSTANCES=${2:-1}
CLIENT_INSTANCES='expr $(echo $SERVERS | wc -w) \* 2'

AUTOPIN=adq
#AUTOPIN=numa

WARMUP=10
RUNTIME=20
STEP=1      # seconds
TOTALTIME=$((WARMUP+RUNTIME))
SAMPLES_TOTAL=$((TOTALTIME/STEP+5))

STARTKEY=0
KEYS=1000000 # 1M
#KEYS=10000000 # 10M
#KEYS=100000000 # 100M
#KEYS=1000000000 # 1B

CLIENTS="$(echo client{7..14})"

BENCHMARK=asd-bench

CLIENT_CMD="./asd-bench -h ${TARGET_HOSTS} -k ${KEYS} -o S:32 -w RU,100 -a -c 142 -W 12 --latency
21,4"
DB_INIT_CMD="./asd-bench -h ${TARGET_HOSTS} -k ${KEYS} -o S:32 -w I -a -c 100 -W 10 -M master
--maxRetries 10 -T 10000"

RESULTS_DIR=results/${BENCHMARK}_${1}_S$(echo $SERVERS | wc -w)-I2_C$(echo $CLIENTS | wc -w)-
I${CLIENT_INSTANCES}_$((KEYS/1000000))M_${TIMESTAMP}
echo "RESULTS DIRECTORY: ${RESULTS_DIR}"
mkdir -p ${RESULTS_DIR}

echo "Preparing..."
sed -i "s/auto-pin.*/auto-pin ${AUTOPIN}/" server_files/aerospike*.conf
for CLIENT in ${CLIENTS};
do
  scp -p client_files/* ${CLIENT}:
  ssh $CLIENT "pkill ${BENCHMARK} ; pkill nmon ; ./systune.sh ; sleep $STEP ; systemctl stop
irqbalance ; ./set_irq_affinity -x local ens786f0 ; sleep $STEP ; rm -f result_${CLIENT}* ; rm -f
${CLIENT}.nmon ; sync" &
done
for SERVER in ${SERVERS};
do
  scp -p server_files/* ${SERVER}:
done
wait

for SERVER in ${SERVERS};
do
  # Memory-only / ADQ-on:
  #ssh $SERVER "pkill nmon ; pkill asd ; sleep $STEP ; rm -f server*_aerospike*.log ; rm -f
${SERVER}.nmon ; ./systune.sh ; ./setup-adq.sh ens1f0 54 3000 0 500 50000 1 ; ./setup-adq.sh ens8f0
54 4000 0 500 50000 0 ; sleep 3 ; ./asd-startup.sh; sleep $STEP ; " &

  # Memory-only / ADQ-off:
  ssh $SERVER "pkill nmon ; pkill asd ; sleep $STEP ; rm -f server*_aerospike*.log ; rm -f ${SERVER}.
nmon ; ./systune.sh ; sysctl -w net.core.busy_poll=0 ; modprobe -r ice ; modprobe ice ; systemctl
```

```
stop irqbalance ; ./set_irq_affinity -x local ens1f0 ; ./set_irq_affinity -x local ens8f0 ; ifup ens1f0
; ifup ens8f0 ; ./asd-startup-numa.sh; sleep $STEP ; " &

  # PMEM / ADQ-on:
  #ssh $SERVER "pkill nmon ; pkill asd ; sleep $STEP ; rm -f /mnt/pmem*/* ; rm -f server*_aerospike*.
log ; rm -f ${SERVER}.nmon ; ./systune.sh ; ./setup-adq.sh ens1f0 54 3000 0 500 50000 1 ; ./setup-
adq.sh ens8f0 54 4000 0 500 50000 0 ; sleep 3 ; ./asd-startup.sh; sleep $STEP ; " &

  # PMEM / ADQ-off:
  #ssh $SERVER "pkill nmon ; pkill asd ; sleep $STEP ; rm -f /mnt/pmem*/* ; rm -f server*_aerospike*.
log ; rm -f ${SERVER}.nmon ; ./systune.sh ; sysctl -w net.core.busy_poll=0 ; modprobe -r ice ;
modprobe ice ; systemctl stop irqbalance ; ./set_irq_affinity -x local ens1f0 ; ./set_irq_affinity -x
local ens8f0 ; ifup ens1f0 ; ifup ens8f0 ; ./asd-startup-numa.sh; sleep $STEP ; " &
done
wait
sleep 5

for CLIENT in ${CLIENTS};
do
  ssh $CLIENT "$DB_INIT_CMD"
  break
done
sleep 3

echo "Running for $TOTALTIME seconds..."
for SERVER in ${SERVERS};
do
  ssh $SERVER "nmon -F ${SERVER}.nmon -s${STEP} -c$((SAMPLES_TOTAL)) -J -t" &
done
for CLIENT in ${CLIENTS};
do
  ssh ${CLIENT} "nmon -F ${CLIENT}.nmon -s${STEP} -c$((SAMPLES_TOTAL)) -J -t" &

  for CLIENT_INSTANCE in 'seq 1 ${CLIENT_INSTANCES}';
  do
    ssh $CLIENT "${CLIENT_CMD} 1> result_${CLIENT}-${CLIENT_INSTANCE}.out 2> result_${CLIENT}-
${CLIENT_INSTANCE}.err" &
    if [ "$CLIENT" == "client5" ]; then
      break
    fi
  done
done

sleep ${TOTALTIME}

for CLIENT in ${CLIENTS};
do
  ssh $CLIENT "pkill ${BENCHMARK} ; pkill nmon" &
done
for SERVER in ${SERVERS};
do
  ssh $SERVER "pkill asd ; pkill nmon" &
done
wait
sleep $STEP

for CLIENT in ${CLIENTS};
do
  scp $CLIENT:result_${CLIENT}* ${RESULTS_DIR}/
  scp $CLIENT:${CLIENT}.nmon ${RESULTS_DIR}/
done
for SERVER in ${SERVERS};
do
```

```
  scp $SERVER:server*_aerospike*.log ${RESULTS_DIR}/
  scp $SERVER:${SERVER}.nmon ${RESULTS_DIR}/
done
wait
find ${RESULTS_DIR}/ -size 0 -delete

echo "Client Errors:"
tail ${RESULTS_DIR}/result_*.err 2> /dev/null
grep ERROR ${RESULTS_DIR}/result_*.out 2> /dev/null
echo

cp -fvp ${0} ${RESULTS_DIR}/
cp -rvp client_files ${RESULTS_DIR}/
cp -rvp server_files ${RESULTS_DIR}/
echo -en "${RESULTS_DIR}\t" >> results.txt
./parse_results.sh ${RESULTS_DIR} ${RUNTIME} | tail -n 1 | tee -a results.txt
cp -fvp parse_results.sh ${RESULTS_DIR}/
```

parse_results.sh

```
#!/bin/bash
RESULTS_DIR=${1}
TIME=${2:-20}

pushd $RESULTS_DIR
RESULTS_FILES=result_client*.out
RESULTS_SORTED=$(ls ${RESULTS_FILES} | sort -V)
RESULTS_COUNT=$(ls ${RESULTS} | wc -w)

echo "BEGIN PARSE RESULTS:"


awk -F"[ =]" '
fname != FILENAME { fname = FILENAME ; files[count++]=fname ; row=1 }
/total\(tps=[0-9]+/{ val_array[row++][fname]=$13 ; }
END { for (n in files) printf("%s,",files[n]) ; print ""; for (row in val_array) { for (n in files)
printf("%d,",val_array[row][files[n]]); print ""}}
' $RESULTS_SORTED > results_total_tps.csv

awk -F"[ %]+" '
fname != FILENAME { fname = FILENAME ; files[count++]=fname ; file_count++ ; row=0 }
/read  /{row++; col=0; for(i = 2; i < NF; i++) { lat_sum[row][++col]+=$i; printf $i"\t" } ; print
row,fname ; if (nfmax<NF) nfmax=NF; if(colmax<col) colmax=col; }
END { print "Files:",file_count,", Rows:",row,", Columns:",colmax; for (row in lat_sum) { for (col in
lat_sum[row]) { printf("%.4f\t",lat_sum[row][col]/file_count) } print ""} }
' $RESULTS_SORTED > results_read_latency.txt


echo
echo -e "         \tRead Latency Histogram (Average):"
echo -e "Total_TPS\t<=40us\t>40us\t>80us\t>120us\t>160us\t>200us\t>240us\t>280us\t>320us\t>360us\
t>400us\t>440us\t>480us\t>520us\t>560us\t>600us"
awk -F"[ =]" '/total\(tps=[0-9]+/{ sum[FNR]+=$13 } END { for(row in sum) print sum[row] }' ${RESULTS_
SORTED} | tail -n ${TIME} | awk '/[0-9]+/{sum+=$1;count++} END{printf("%.f\t",sum/count)}'
tail -n ${TIME} results_read_latency.txt | awk '/[0-9]/{ for(i = 1; i <= NF; i++) lat_sum[i]+=$i }
END { for (i in lat_sum) printf("%.4f\t",lat_sum[i]/NR) ; print "" }
```

Read the report at http://facts.pt/VEgYlbs ▶

This project was commissioned by Dell EMC.

Principled Technologies®

Facts matter.®