Performance Tuning Guidelines for Low Latency Response on AMD EPYC™-Based Servers Application Note

Publication #	**56263** Revision: **3.01**
Issue Date:	**June 2018**

Advanced Micro Devices

Contents

List of Figures

Revision History

Date	Revision	Description
June 2018	3.01	**Overview** section: • Modified the opening paragraph. **Hardware Configuration** section: • Modified text and figures throughout. **BIOS Configuration** section: • Updated the **Profiles** subsection. **Operating System Configuration** section: • Modified text and figures throughout and added the following new subsections: ○ **The /proc filesystem** ○ **The sysfs** ○ **Kswapd Thread Affinity** ○ **IRQ Affinity** ○ **Kernel Timer Interrupts** ○ **RCU Threads**
January 2018	3.00	Initial public release.

Overview

Low latency market segments, such as financial trading or real time processing, require for a server to provide consistent system response under 10 µs. This document provides guidance for tuning servers utilizing the AMD EPYC™ processor to reach this requirement. The guidelines cover hardware configuration, BIOS settings, operating system kernel configurations, scripts to control the environment of the target applications, and a description of the proper technique for collecting code execution timing data.

Hardware Configuration

To achieve low latency in the µs range, it is important to understand the hardware configuration of the system under test. Important factors affecting response times include the number of cores, the execution threads per core, the number of sockets, the number of NUMA nodes, CPU and memory arrangement in the NUMA topology, and the cache topology in a NUMA node. For Linux based systems, there are many tools which display the configuration at varying levels of detail and in various formats. For a very high-level overview of the system, use the **lscpu** command on the Linux command line as shown in Figure 1 on page 7.

```
[root@localhost ~]# lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                64
On-line CPU(s) list:   0-63
Thread(s) per core:    1
Core(s) per socket:    32
Socket(s):             2
NUMA node(s):          8
Vendor ID:             AuthenticAMD
CPU family:            23
Model:                 1
Model name:            AMD Eng Sample: 2S2201BDVIHAF_32/22_N
Stepping:              2
CPU MHz:               2200.000
CPU max MHz:           2200.0000
CPU min MHz:           1200.0000
BogoMIPS:              4391.47
Virtualization:        AMD-V
L1d cache:             32K
L1i cache:             64K
L2 cache:              512K
L3 cache:              8192K
NUMA node0 CPU(s):     0-7
NUMA node1 CPU(s):     8-15
NUMA node2 CPU(s):     16-23
NUMA node3 CPU(s):     24-31
NUMA node4 CPU(s):     32-39
NUMA node5 CPU(s):     40-47
NUMA node6 CPU(s):     48-55
NUMA node7 CPU(s):     56-63
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext
fxsr_opt pdpe1gb rdtscp lm constant_tsc art rep_good nopl nonstop_tsc
extd_apicid amd_dcm aperfmperf eagerfpu pni pclmulqdq monitor ssse3 fma
cx16 sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand lahf_lm
cmp_legacy svm extapic cr8_legacy abm sse4a misalignsse 3dnowprefetch osvw
skinit wdt tce topoext perfctr_core perfctr_nb bpext perfctr_l2 cpb
hw_pstate retpoline_amd vmmcall fsgsbase bmi1 avx2 smep bmi2 rdseed adx
smap clflushopt sha_ni xsaveopt xsavec xgetbv1 clzero irperf xsaveerptr
ibpb arat npt lbrv svm_lock nrip_save tsc_scale vmcb_clean flushbyasid
decodeassists pausefilter pfthreshold avic v_vmsave_vmload vgif
overflow_recov succor smca
```

Figure 1. Output from lscpu for a Two-Socket System Under Test

For a view of the topology of the system under test, use the **lstopo** topology tool. For RHEL 7, use the following repository to install lstopo, which is a part of the hwloc package. Install the hwloc-gui package to get the option to format the lstopo output in multiple graphic formats:

```
subscription-manager repos -- enable=rhel-7-server-optional-rpms
```

```
yum install hwloc hwloc-gui -y
```

Multiple options for the output format are available. An example of the graphical output generated by the following command line is provided in Figure 2.

```
>lstopo –physical –output-format png><output filename>
```

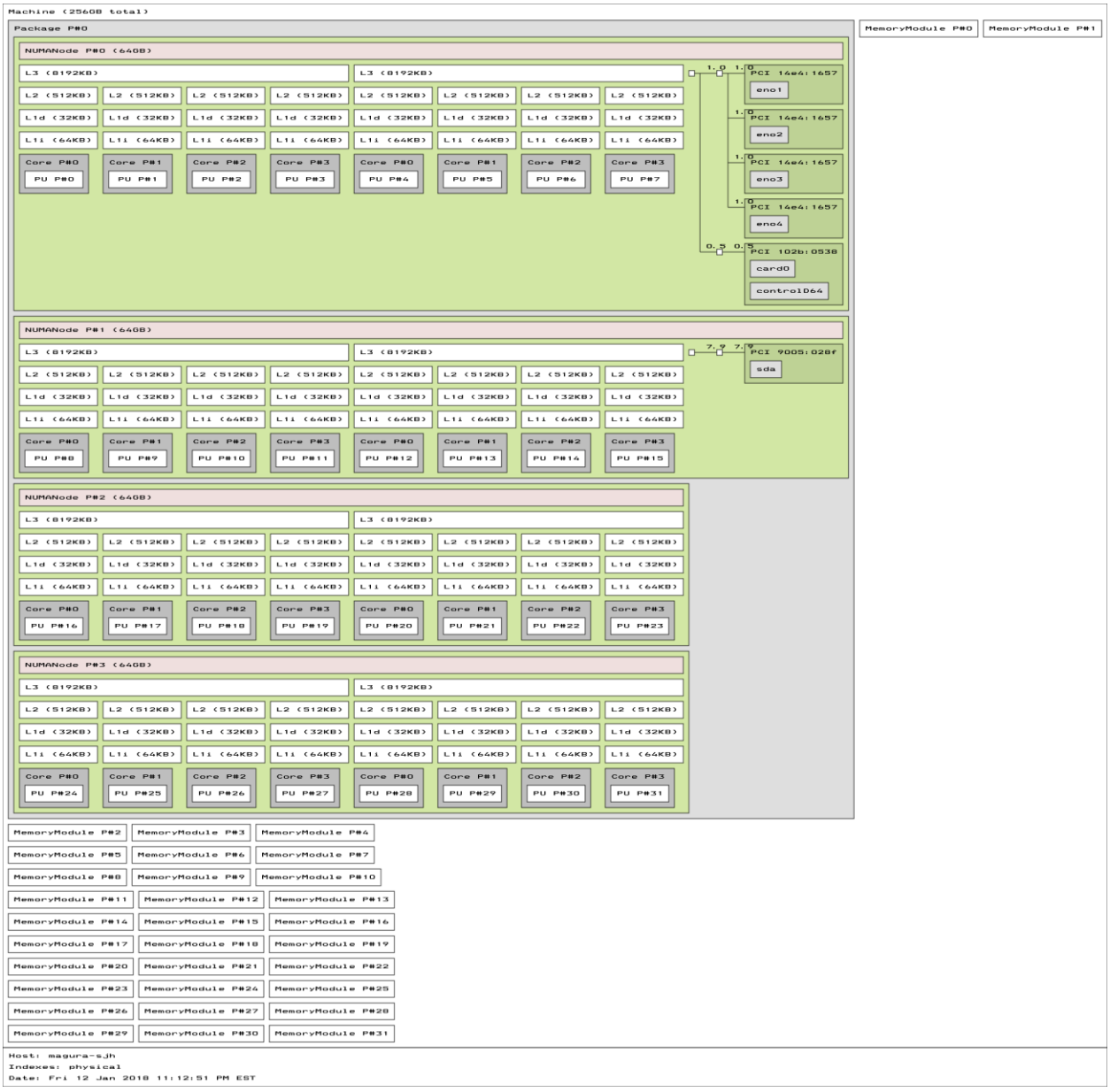

Figure 2. Output from lstopo for a One-Socket System Under Test

To achieve the best response times, optimize the system topology where possible to match your operational needs. Be aware of the memory placement, install memory evenly across the NUMA nodes, and try to maximize the use of local memory. Isolate the cores executing your time-critical application from the operating system scheduler so that other applications and kernel threads do not steal execution time from your application. Isolate your application's cores from interrupts as much as possible. Utilize the Linux utilities and techniques described in later paragraphs to optimally manage hardware components and attributes.

System Management Mode

System Management Mode (SMM) runs in firmware context outside the visibility of the operating system to perform hardware related tasks and OEM code. Transfer to SMM context is achieved through a System Management Interrupt (SMI) that puts all cores into SMM mode. Because all cores are interrupted by the SMI, the latency impact increases proportionally to the number of cores.

On AMD EPYC™-based platforms, you can monitor the SMI count with the perf subsystem. Run the following command:

```
perf stat -a -I 100 -e r02b
```

If this command results in nonzero values over a sampling period, consult the system documentation on the probable causes and consider the BIOS setting options discussed below that can be used to decrease or remove SMIs.

BIOS Configuration

Many sources of system latency can be disabled through the settings in the BIOS setup utility.

Profiles

If the BIOS setup utility provides a profile designed for **Low Latency,** select this option to lock in a predetermined set of settings known by the system vendor to optimize latency. For example, on the system under test for this investigation, choosing the Low Latency profile locked in the following BIOS settings:

- Prefetcher -Enabled
- L2 Stream HW Prefetcher - Enabled
- SR-IOV – Disabled
- AMD IOMMU – Disabled
- Min Processor Idle Power Core C-State – No C-State - Disabled
- AMD TurboCore – Disabled
- L1 Stream HW – Enabled
- NUMA Group Size Optimization – Clustered
- Memory patrol scrubbing – Disabled
- Memory Refresh Rate – 1X

If there are other options that differentiate between power and performance, latency, or deterministic performance, choose those over power. For example, if available, choose **Performance Determinism** to minimize performance jitter.

SMM SMIs

Consider the benefit and risk of disabling the following settings to decrease the source of potential SMIs:

- **Processor Power and Utilization Monitoring**—Processor State Mode Switching and Insight Power Management Processor Utilization Monitoring
- **Memory Patrol Scrubbing**—Corrects soft memory errors that might reduce the risk of uncorrectable errors later
- **Memory Pre-Failure Notification**—Disables notification when correctable errors occur above a threshold

Core Count

If your computational needs do not require the power of a large core system, the BIOS setup utility can be used to down core the system to decrease operating system overhead and latency.

SMT

Using multiple execution threads per core requires resource sharing and is a possible source of inconsistent system response. In the BIOS setup, disable the **AMD SMT option**. After boot, verify SMT is disabled by running **lscpu** and confirm that Threads per core = 1.

Memory Options

Disable **Node Interleaving** to preserve the use of local node memory allocation. Local memory should provide the lowest access latency.

The new **AMD Secure Memory Encryption** feature could cause a small latency increase. Disable if low latency is of higher priority than the new increased security.

Virtualization Options

If your application scenario does not require virtualization, then disable **AMD Virtualization Technology**. With virtualization disabled, also, disable **AMD IOMMU**. It can cause differences in latency for memory access. Finally, also disable **SR-IOV**.

Core Performance Boost

Core Performance Boost can cause jitter due to frequency transitions of the processor cores. Disable AMD Core Performance Boost. Verify CPB is disabled by checking bit 25 = 1 of MSRC001_0015 [Hardware Configuration] (HWCR) with rdmsr 0xc0010015.

ACPI

The **ACPI SLIT** table provides the relative latencies between nodes. Enable this feature so the information can be used by the Linux scheduler.

Disable the **ACPI Watchdog Timer** to remove a source of interrupts if you do not need the automatic reset feature in case of an overflow.

Memory Proximity Reporting for IO

Enable **Memory Proximity Reporting for IO**. This feature reports the proximity relationship between IO devices and system memory to the operating system.

C-States

C-States can be a source of jitter during the process of state transition. Disable all c-states.

Operating System Configuration

Optional Use of the Red Hat Enterprise 7 Real Time Kernel

Consider using the RHEL 7 real time kernel when the latency requirements are below 10 microseconds. This kernel has been especially tuned for low latency deterministic response.

The real time kernel can be installed with the following commands:

```
>subscription-manager repos –enable rhel-7-server-rt-rpms
>yum groupinstall RT
```

Red Hat Enterprise Linux 7.4/7.5

However, assuming the user does not want to use anything but the standard RHEL 7.4 kernel, the remaining tuning tips are illustrated with a two socket AMD EPYC™-based system with the standard kernel installed.

***Note**: The 10 µs target was achieved with the standard kernel using the following tuning tips.*

The /proc filesystem

The /proc filesystem interface can expose information on a per process level, or the internal kernel data and the kernel subsystems, and system devices. A few examples of reading kernel information are shown below.

Enter the following command to verify the running kernel:

```
> cat/proc/sys/kernel/osrelease
> 3.10.0-862.el7.x86_64
```

Enter the following command to verify the kernel command line parameters used when the kernel was booted:

```
> cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-3.10.0-862.el7.x86_64 root=/dev/mapper/rhel-root ro
crashkernel=auto rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet idle=poll
transparent_hugepage=never audit=0 selinux=0 nmi_watchdog=0 nohz=on
clocksource=tsc nosoftlockup mce=ignore_ce cpuidle.off=1 skew_tick=1
processor.max_cstate=0 isolcpus=8-15 rcunocbs rcu_nocb_poll nohz_full
acpi_irq_nobalance LANG=en_US.UTF-8
```

***Note**: The kernel boot parameters utilized for low latency tuning are described in a later paragraph.*

However, the /proc filesystem is not just read-only display of this data. Many of the files are writable and can be used to fine tune system parameters. Here are the parameters that were found to be helpful with the low latency benchmark used in these experiments:

```
>echo 0 > /proc/sys/kernel/watchdog
```

Watchdog monitoring and interrupts can cause jitter. Disable them during the benchmarking.

```
>echo 10 > /proc/sys/vm/swappiness
```

Lowering the swappiness to 10 means that swap will be used when RAM is 90% full. If your system has a lot of memory, this should be a safe and effective choice to improve performance.

```
>echo 10 > /proc/sys/vm/dirty_ratio
```

Dirty_ratio is the limit to which, if the total number of dirty pages exceed, then all writes are blocked until some of the dirty pages are written to disk.

```
>echo 3 > /proc/sys/vm/dirty_background_ratio
```

Dirty_background_ratio is the limit to which, when dirty page exceeds, then they start getting written to the disk.

```
>echo 24000000 >  /proc/sys/kernel/sched_latency_ns
```

Sched_lantency_ns represents the preemption latency of a CPU bound task. Increasing this value increases the task's timeslice.

```
>echo 10000000 > /proc/sys/kernel/sched_min_granularity_ns
```

Sched_min_granularity_ns is the minimum preemption granularity for CPU bound tasks.

```
>echo 5000000 > /proc/sys/kernel/sched_migration_cost_ns
```

Sched_migration_cost_ns determines how long a task remains cache-hot after the last execution and, hence, avoid migration off the CPU. Increasing this variable reduces task migrations.

```
>echo -1 > /proc/sys/kernel/sched_rt_runtime_us
```

Sched_rt_runtime_us is the quantum allowed rt tasks. The Default is 950000 out of 1000000. A value of -1 disables the enforcement.

```
>echo 1000 > /proc/sys/vm/stat_interval
```

Stat_interval is the interval in seconds in which the vm statistics are updated. The default is 1, which was causing a spike every second while collecting the statistics. Changing it to 1000 avoids these interrupts for at least 16 mins.

As the optimal values from these parameters are determined experimentally, they can be made persistent by writing the values into a custom sysctl configuration file and added into the /etc/sysctl.d directory. For example, the runtime tuning parameter above was written into a file called 100-lowlatency.conf. The number at the front of the filename was chosen to be the largest of any named file in this directory so that it would be read last. The file content of this conf file is listed below:

```
kernel.watchdog = 0
vm.swappiness = 10
vm.dirty_ratio = 10
vm.dirty_background_ratio = 3
kernel.sched_latency_ns = 24000000
kernel.sched_min_granularity_ns = 10000000
kernel.sched_migration_cost_ns = 5000000
kernel.sched_rt_runtime_us = -1
vm.stat.interval = 1000
```

This settings is read from the conf file at boot time. To have the conf file read and the settings put in effect immediately, the following command can be used:

```
sysctl -p <xx-customfile.conf>
```

The sysfs

Much like the /proc filesystem, the /sys filesystem has parameters that can be written to in order to improve deterministic behavior.

```
>echo performance > /sys/devices/system/cpu/cpu57/cpufreq/scaling_governor
```

This sets the system to execute your benchmark at the highest fixed frequency state.

```
>echo 0 > /sys/bus/workqueue/devices/writeback/numa
```

This disables the NUMA affinity for the writeback threads. These threads should be moved to only the housekeeping cpus.

```
>echo ff > /sys/bus/workqueue/devices/writeback/cpumask
```

This sets a cpumask to allow these threads to execute on the lower 8 cpus. The cpumask for the benchmark excludes these cpus.

Kernel Boot Parameter

Experiment and consider utilizing the following kernel boot parameters. To make them permanent across system boots, modify the /etc/default/grub configuration file and generate a new grub.cfg file with the following commands:

```
>vi /etc/default/grub
..... make your changes and save the file
>grub2-mkconfig -o /boot/<path for your system>/grub.cfg
```

The parameters chosen to be made persistent by the grub file are added to the command line by default when booting the system.

- **idle=poll**—forces a polling idle loop that can slightly improve the performance of waking up an idle CPU.
- **transparent_hugepage=never**—This disables transparent_hugepages because managing transparent_hugepages can cause latency spikes when the kernel cannot find a contiguous 2M page, or the daemon is defragmenting and collapsing memory into one huge page in the background. If your application could benefit from huge pages, consider using static huge pages by configuring hugetlbfs. The memory allocations in your application need to be from heap memory to see the benefit of static huge pages.
- **audit=0**—The Linux Audit system provides a way to track and log events that are happening on your system. This causes the kernel audit subsystem to be disabled and cannot be enabled until the next reboot.

- **selinux=0**—This disables the Linux Security module that provides a mechanism for supporting access control security policies.
- **nmi_watchdog=0**—This disables the nmi watchdog because it uses the Perf infrastructure. Perf is not intended to run as a continuous profiling utility, especially in low latency environments where it can cause spikes.
- **nohz=on**—Disables the kernel timer tick on idle cores.
- **clocksource=tsc**—Select the preferred kernel clock source.
- **nosoftlockup**—Disables logging of backtraces when a process executes on a CPU or longer than the softlockup threshold (default 120 seconds).
- **mce=ignore_ce**—Disable features for corrected errors.
- **cpuidle.off=1**—Disable the cpuidle sub-system.
- **skew_tick=1**—Causes the kernel to program each CPU's tick timer to fire at different times to avoid any possible lock contention.
- **processor.max_cstate=0**—Disables going from C0 into any other C-state.
- **isolcpus=<core list>**—Isolates the cores from the scheduler.
- **rcunocbs=<core list>**—Restricts these cores from receiving any rcu call backs.
- **rcu_nocb_poll**—Relieves each CPU from the responsibility awakening their RCU offload threads. Using the combination of rcunocbs and rcu_nocb_poll reduces the interference on your benchmark cpus.
- **nohz_full=<core list>**—Restricts cores from receiving any timer ticks if only one process is running.
- **acpi_irq_nobalance**—ACPI will not move active IRQs.

Tuned-adm Profiles

Tuned-adm profiles can be used instead of writing directly to the /proc filesystem or using sysctl. However, if you stopped or disabled the Tuned daemon, you must restart it to set the profile. The real time profile is a child of the network-latency profile, and thus inherits all the settings of latency-performance profile and the network-latency profiles. Make sure that the Tuned is configured for static settings and not dynamic. Be sure and set any suggested settings mentioned in prior paragraphs if they are missing from the real time profile, or create your own profile that inherits from the real time profile. The final experiments in support of this paper did not utilize Tuned, because it was preferable to disable the Tuned daemon. However, it is likely an option that could also be made to work.

```
<tuned-adm profile realtime
```

Systemd Services

Disable or stop all unnecessary services with **systemctl**. An example script to stop a set of services and ban a set of cores from receiving interrupts is provided in Figure 3 on page 18. This takes iterative experimentation. The services started on a system by default are dependent on the platform, OS configuration, devices, and workload scenario. An example list is provided in Figure 4 on page 19.

***Note**: Some services may be required for the system to boot.*

```
avahi-daemon.service \
chronyd.service kdump.service ksm.service tuned.service libstoragemgmt.service
libvirtd.service mcelog.service \
ModemManager.service rpc-gssd.service rpcbind.socket rpcbind.service
smartd.services systemd-journald.socket \
systemd-journald.service virtlockd.socket virtlogd.socket nfs-client.target
rhnsd.service messagebus.service anacron autofs bluetooth certmonger \
haldaemon hidd ip6tables iprdump iprinit iprupdate mdmonitor nfs-lock restorecond
rhsmcertd}
do
systemctl stop $SERVICE
done
service irqbalance stop
IRQBALANCE_ONESHOT=1 IRQBALANCE_BANNED_CPUS=${COREMASK}
irqbalance
```

Figure 3. Script to Use systemctl to Disable and Stop Services

```
>systemctl list-unit-files | grep enabled
auditd.service                                enabled
autovt@.service                               enabled
dbus-org.freedesktop.NetworkManager.service   enabled
dbus-org.freedesktop.nm-dispatcher.service    enabled
getty@.service                                enabled
lttng-sessiond.service                        enabled
NetworkManager-dispatcher.service             enabled
NetworkManager.service                        enabled
rhsmcertd.service                             enabled
rsyslog.service                               enabled
rt-setup.service                              enabled
rtctl.service                                 enabled
sshd.service                                  enabled
systemd-readahead-collect.service             enabled
systemd-readahead-drop.service                enabled
systemd-readahead-replay.service              enabled
tuned.service                                 enabled
dm-event.socket                               enabled
lvm2-lvmetad.socket                           enabled
lvm2-lvmpolld.socket                          enabled
default.target                                enabled
multi-user.target                             enabled
remote-fs.target                              enabled
runlevel2.target                              enabled
runlevel3.target                              enabled
runlevel4.target                              enabled
```

Figure 4. Example of Enabled Services

Kernel Modules

Unload unnecessary modules. This is highly dependent on the platform configuration, system devices and OS configuration. This will take iterative experimentation, but good candidates to start with are the iptables, netfilter, and the edac driver.

You can temporarily disable a module until the next reboot if the module is not being used or another module is not dependent on it. First check for use with:

```
>lsmod | grep <module-name>
...if no modules are using the module then run the next command
>modprobe -r <module-name>
```

To prevent the module from being loaded at boot, make sure that the module is not configured in any of the module configuration files, such as /etc/modprobe.conf , /etc/modprobe.d/*, /etc/rc.modules, or /etc/sysconfig/modules/*. Then add the following line: blacklist <module-name> to a configuration file such as /etc/modprob.d/local-blacklist.conf.

Here is the script used for the experiments to support this paper:

```
#! /bin/bash
iptables -F ; iptables -t nat -F; iptables -t mangle -F ; ip6tables -F
iptables -X ; iptables -t nat -X; iptables -t mangle -X ; ip6tables -X
iptables -t raw -F ; iptables -t raw -X
modprobe -r ebtable_nat ebtables
modprobe -r ipt_SYNPROXY nf_synproxy_core xt_CT nf_conntrack_ftp \
nf_conntrack_tftp nf_conntrack_irc nf_nat_tftp ipt_MASQUERADE \
iptable_nat nf_nat_ipv4 nf_nat nf_conntrack_ipv4 nf_nat \
nf_conntrack_ipv6 xt_state xt_conntrack iptable_raw \
nf_conntrack iptable_filter iptable_raw iptable_mangle \
ipt_REJECT xt_CHECKSUM ip_tables nf_defrag_ipv4 ip6table_filter \
ip6_tables nf_defrag_ipv6 ip6t_REJECT xt_LOG xt_multiport \
nf_conntrack
```

Kswapd Thread Affinity

The kswapd threads manage the free pages and take action if they fall below a watermark. To make sure these threads do not cause jitter on the benchmark cpu, set the affinity of all the kswapd threads to other cpus. The script used in these experiments follows:

```
#!/bin/bash
for i in `pgrep kswapd[^c]`
do taskset -pc 0-7 $i
done
```

IRQ Affinity

Interrupts are a key cause of jitter. To prevent jitter caused by system interrupts being handled by the benchmarking cpus, the /proc subsystem can be utilized to affinitize these interrupts to other cpus. The following script was used in the experiments to support this paper:

```
#! /bin/bash
for f in /proc/irq/*/smp_affinity
do
echo 00000000,000000ff > $f
cat $f
done
```

Kernel Timer Interrupts

Check that your running kernel was built with CONFIG_HOTPLUG_CPU=y. The /proc filesystem can help again by utilizing the online file. Bring the benchmark cpus offline, then immediately bring them back online. This forces timers to migrate to other cpus. The following script was used in the experiments supporting this paper:

```
#!/bin/bash
for i in {8..15}
do `echo 0 > /sys/devices/system/cpu/cpu$i/online`
cat /sys/devices/system/cpu/cpu$i/online
done
sleep 10
for i in {8..15}
do `echo 1 > /sys/devices/system/cpu/cpu$i/online`
cat /sys/devices/system/cpu/cpu$i/online
done
```

RCU Threads

The RCU kernel synchronization operations occur on multiple threads. The callbacks have already been moved off the benchmarking cpus via kernel boot parameters (rcunocbs and rcu_nocb_poll). Now the core operational threads need to be affinitized to non-benchmarking cpus. The following script was used in the experiments for this paper:

```
#!/bin/bash
for i in `pgrep rcu[^c]`
do taskset -pc 0 $i
done
```

Benchmarking System Latency

The latency benchmark tests for this paper were performed on a pre-production HPE AMD EPYC™-based 1 socket 32 core server and a 2 socket 64 core server. The benchmark application was HP-TimeTest7.4. The tests were run with the standard RHEL 7.4 kernel and the RHEL 7.4 real time kernel. All tests were run with the full available cores enabled. After proper tuning settings were made and scripts were run, the tests completed successfully with both kernels.

Important considerations when using this benchmark application:

1) Always run this benchmark application using the cycles method (-m cycles).

2) Always set the threshold based on the frequency of your system under test. The critical threshold is 10 µs. Therefore, calculate the cycles for your system that represent 10 µs. The systems used in these experiments had a frequency of 2.2 GHZ, so 2.2x10^9 * 10x10^-6 would be 2.2x10^3 cycles for a 10 µs threshold (-t 22000).

3) Set the loopcount parameter to represent the number loops through the measurement cycle that converts to the desired time of the test. For the systems used in these experiments, 22000000000 loops represents approximately a 10 minute run. During tuning experiments, the -l 22000000000 parameter was used. After the system was completely tuned, experiments up to 2 hours were successfully run.

4) Make sure that the isolcpus kernel boot parameter includes the core that the benchmark application will be run on. In these experiments, cores 8-15 were isolated so, the whole numa node of the benchmark would be isolated.

5) Taskset was used to pin the application to a core from the isolated list.

The same /etc/default/grub settings were used to generate the kernel command line options for both kernels as shown in Figure 5.

An example command line execution of the benchmark is shown in Figure 6.

```
BOOT_IMAGE=/vmlinuz-3.10.0-862.el7.x86_64 root=/dev/mapper/rhel-root ro crashkernel=auto
rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet idle=poll transparent_hugepage=never audit=0 selinux=0
nmi_watchdog=0 nohz=on clocksource=tsc nosoftlockup mce=ignore_ce cpuidle.off=1 skew_tick=1
processor.max_cstate=0 isolcpus=8-15 rcunocbs rcu_nocb_poll nohz_full acpi_irq_nobalance
LANG=en_US.UTF-8
```

Figure 5. Kernel Command Line Options

```
>taskset -c 8 ./HP-TimeTest7.4 -m cycles -t 22000 -l 22000000000
```

Figure 6. Example Usage to Start the Benchmark on a 2.2 GHz System with a Threshold of 10 µs and a Test Run of 10 Minutes

References

1. “Configuring and tuning HPE ProLiant Servers for low-latency applications,” (Link: *HPE Low Latency White Edition 10*).
2. “HPE Gen10 Servers Intelligent System Tuning” (Link: *HPE Gen10 Intelligent System Tuning*).
3. The HPE Community Blog, "Reducing latency with HPE ProLiant Gen10 Servers," Scott_Faasse (Link: *HPE Community Blog*).
4. “Low Latency Performance Tuning for Red Hat Enterprise Linux 7,” *https://access.redhat.com/sites/default/files/attachments/201501-perf-brief-low-latency-tuning-rhel7-v1.1.pdf*
5. “Red Hat Enterprise Linux 7 Performance Tuning Guide,” *https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/pdf/performance_tuning_guide/Red_Hat_Enterprise_Linux-7-Performance_Tuning_Guide-en-US.pdf*