Performance Tuning Guidelines for Low Latency Response on AMD EPYC™ 7002 Series Processor Based Servers

Publication	57037_1.0
Revision	1.0
Issue Date	November, 2020

Trademarks

Chapter 1 Overview

Low latency market segments, such as financial trading or real time processing, require that servers provide under 10 μs variation in system response. In the financial world, the ability of companies that work on High-Frequency Trading (HFT) to make uncompromisingly fast executions in the stock market is pivotal to their success and profitability. Here even microseconds could have a major business impact and they demand ultra-low latency from compute to network.

This document provides guidance for tuning servers utilizing two AMD EPYC™ 7F32 processors, or other High Frequency SKUs such as the 7F52 and 7F72 processors, to reach stringent low latency requirements by reducing unwanted computational Jitter. The guidelines cover hardware configuration, BIOS settings, operating system kernel configurations, and scripts to control the environment of the target applications. Read Chapter 2 to understand the system specifications and configuration used for developing this performance tuning guidelines. This document focuses on the server's resources and provide details on tuning the BIOS and OS to optimize the server performance for an HFT environment.

1.1 Sysjitter

Sysjitter is a tool to measure Jitter events from user processes running on specified cores and reports statistics on the interrupts that occurred. This allows users in the HFT community to ensure their server is tuned properly to meet their expectations. See Chapter 5 on Sysjitter to learn more about how to run Sysjitter using a tailored script and what to look for when reading results.

Chapter 2 Hardware Configuration

This technical paper was written using a HPE ProLiant DL385 Gen10 Plus server with the latest HPE BIOS (A42 BIOS Rev 1.24) and iLO FW Rev 2.30 running Red Hat Enterprise Linux x86_64 8.1 with two AMD EPYC 7F32 Processors, and it is applicable to other AMD EPYC High Frequency processors such as the 7F52 and 7F72. To achieve low latency in the µs range, it is important to understand the hardware and firmware configuration of the System Under Test (SUT). Important factors affecting response times include:

- Number of cores
- Execution threads per core
- Number of processors
- Number of NUMA nodes
- CPU and memory arrangement in the NUMA topology
- Cache topology in a NUMA node

If the workload is not limited by memory bandwidth, you may optimize for better latency by running the memory at 2933 MT/s to synchronize with the Infinity Fabric, which runs at 1467 MHz. Linux based system tools such as ipmitool, dmidecode, etc. display the configuration at varying levels of detail and in various formats.

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.

⚠ This Low Latency Performance Tuning Guidelines document is verified on AMD EPYC 7F32 (an 8 Core Processor) and has been tested on a HPE ProLiant DL385 Gen10 Plus server configured with two AMD EPYC 7F32 processors. In addition, it is applicable to the AMD EPYC 7F52 and 7F72 processors but may not be appropriate for servers with other processor SKUs.

For the purposes of this Low Latency Performance Tuning document the System Under Test (SUT) was a HPE ProLiant DL385 Gen10 Plus server with the following resource/capacities. For more details about this server, see its Quick Spec at:
https://h20195.www2.hpe.com/v2/getdocument.aspx?docname=a00073549enw

2nd Generation AMD EPYC™ 7Fx2 Processors	
Processor technology	7nm
Processor Type/SKU	7F32
Number of cores	8
Number of Sockets	2
Memory speed used	3200 MT/s
Memory capacity used	512 GB
Storage	1TB OS Drive SSD/NVMe
NIC	This is a Single Node SUT based test, 1 GigE Network is sufficient.

Chapter 3 BIOS Configuration

Many sources of system latency can be disabled through BIOS settings in the RBSU setup utility. For the purposes of this document, we have tested with 2 x AMD EPYC 7F32 Processors, 512 GB DRAM at 3200 MT/s, and the Red Hat Enterprise Linux x86_64 8.1 Operating System. The AMD EPYC™ 7F32 Processor SKU has 8 cores (16 threads with SMT enabled) across 4 Core Complex Dies (CCDs).

AMD EPYC™ 7F32 Processor	
Base Frequency	3.7 GHz
Max Boost	3.9 GHz
Number of cores	8
L1 Cache Size	32 KB I + 32 KB D on chip per core
L2 Cache Size	512 KB I+D on chip per core
L3 Cache Size	128 MB I+D on chip per chip, 16 MB per Core
Max memory speed	3200 MT/s
Max memory capacity	4TB
Peripheral Component Interconnect	128 lanes PCIe Gen4

3.1 Considerations

Make sure to review the important notes below before you start configuring the system for Low Latency Performance Tuning

- Isolate your application's cores from interrupts as much as possible. Utilize the Linux utilities and techniques described in this document to optimally manage hardware components and attributes such as the Network Adapter's IRQs. See Linux® Network Tuning Guide for AMD EPYC™ 7002 Series Processor Based Servers for details.

  https://developer.amd.com/wp-content/resources/56739_Linux%20Network%20tuning%20v0.20.pdf
- Maximum performance from CPU cores can be extracted by enabling AMD Core Performance Boost. When doing so, monitor the operating frequencies of individual cores while running your workload to determine whether the maximum frequency should be capped (AMD Fmax Boost Limit) to maintain deterministic timing behavior. High fluctuation in CPU frequencies hurt consistent system response. For the same reason, Determinism Control should be set to Performance Deterministic, see Processor Options setting on Power/Performance Determinism document
- https://www.amd.com/system/files/2017-06/Power-Performance-Determinism.pdfFor additional information, see Workload Tuning Guide for AMD EPYC™ 7002 Series Processor Based Servers.
  https://developer.amd.com/wp-content/resources/56745_0.80.pdf

3.2 BIOS Profile Settings

The HPE ProLiant DL385 Gen10 Plus server system's BIOS setup utility provides a list of HPE BIOS profiles designed and tuned for various types of workloads. You can choose the right options to suit your workload needs. Two workloads that are relevant for low latency are:

- Low Latency
- Custom

3.2.1 Low Latency

It is recommended to first choose the Low Latency workload Profile. To choose this Low Latency workload profile,

1. From the BIOS/Platform Configuration (RBSU) menu, select the Low Latency workload Profile. This selects a combination of BIOS tuning options that HPE engineers have determined to be appropriate for Low Latency operation.
2. Allow the server to finish booting and run your Sysjitter workloads to observe the core jitter along with Linux OS tunings recommended in the following chapters.

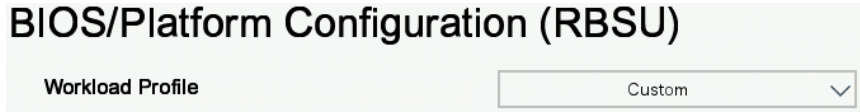
If you are not getting expected Low Latency performance with low Jitter on the CPU Cores, then use the optional Custom BIOS Profile and adjust individual BIOS settings knobs.

Note that AMD Core Performance Boost is explicitly disabled in the Low Latency workload profile. If you desire to enable AMD Core Performance Boost along with that workload profile, first select the Low Latency workload profile and then change to the Custom workload profile. Doing so will leave the Low Latency BIOS settings intact, but then allow you to enable AMD Core Performance Boost. Note that it is not necessary to boot the system between selecting the "Low Latency" workload profile and selecting the "Custom" workload profile.

3.2.2 Custom

The HPE ProLiant DL385 Gen10 Plus server system's BIOS has a Custom workload profile that does not modify any settings but allows changes to all settings. With this selection you can enable your customized Low Latency settings to achieve the latency options for your workloads.

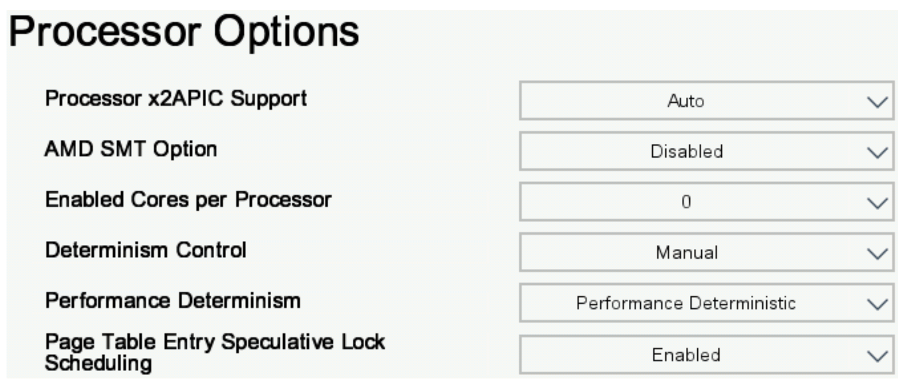
To configure such customized low latency, from the BIOS/Platform Configuration (RBSU) menu select the Custom BIOS profile and then adjust BIOS tuning options with your specific settings to customize it to its best Low Latency configuration for your environment to reduce jitter and improve performance.

<table>
<tr><td>BIOS Custom Profile</td><td>


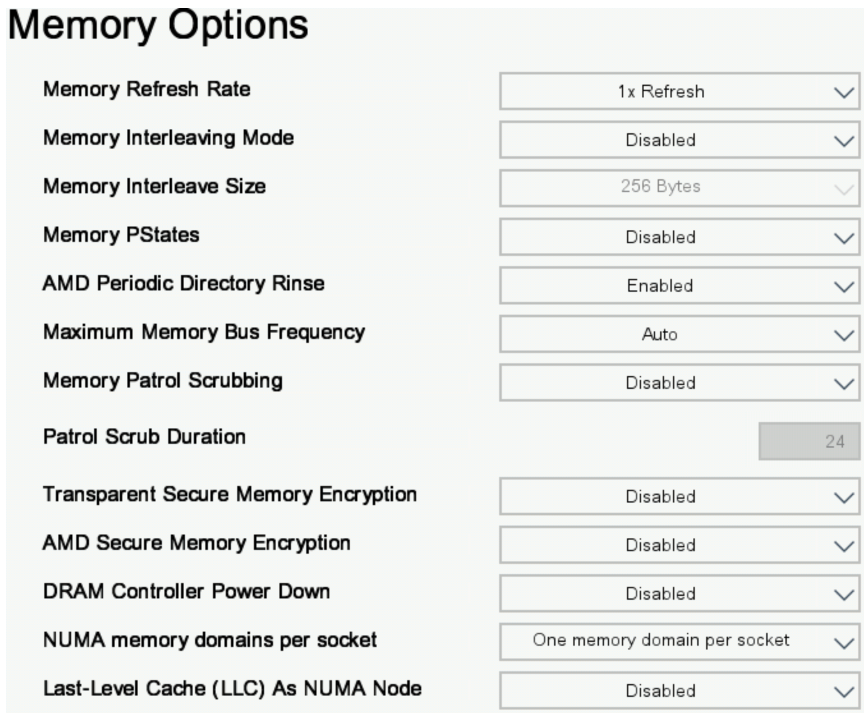
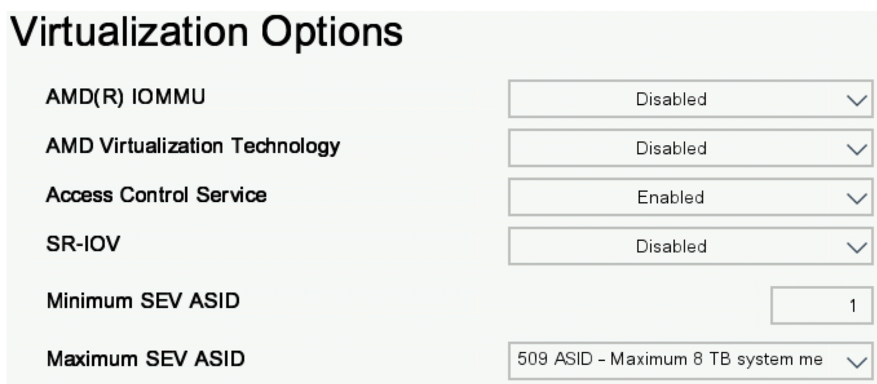
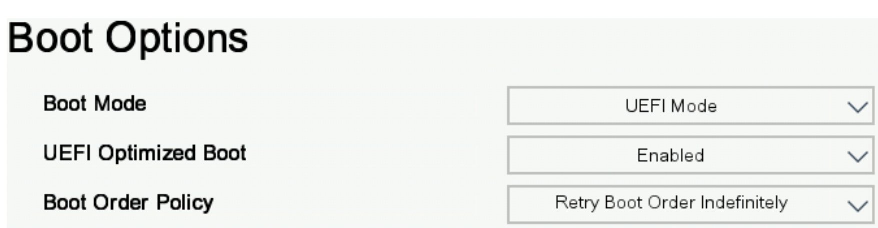
</td></tr>
<tr><td>Boot Time Optimizations</td><td>

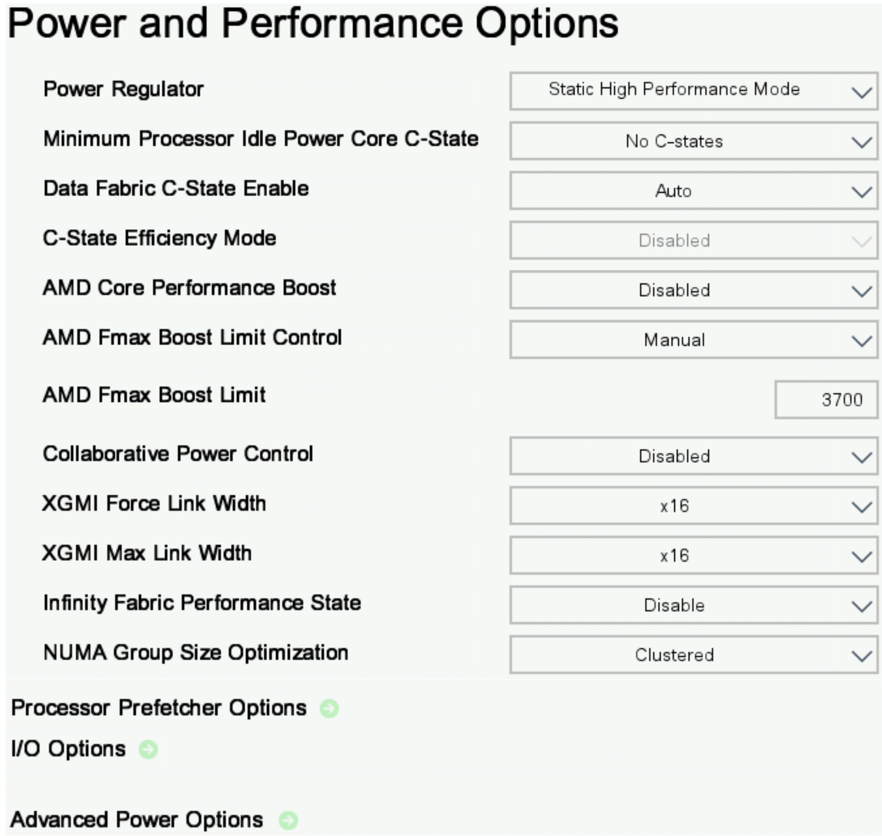

</td></tr>
<tr><td>Diagnostics Options</td><td>

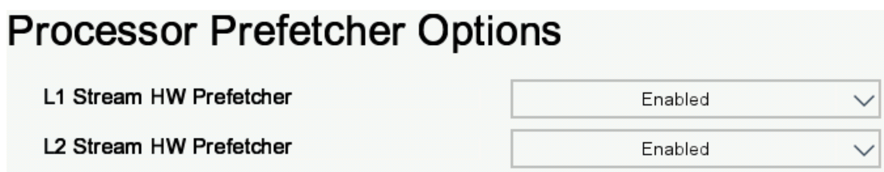

</td></tr>
<tr><td>Processor Options</td><td>
Disable SMT, Determinism Control set to Performance Deterministic

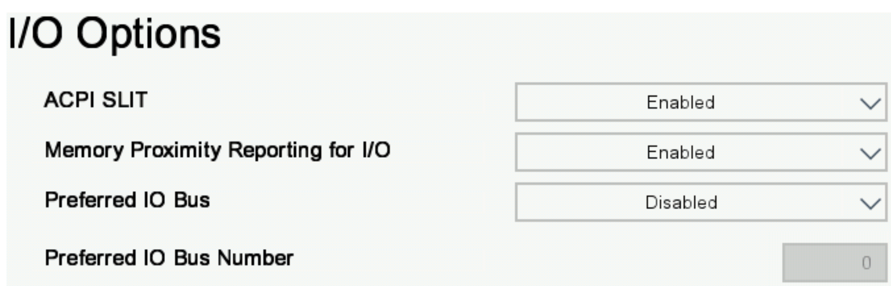


</td></tr>
</table>

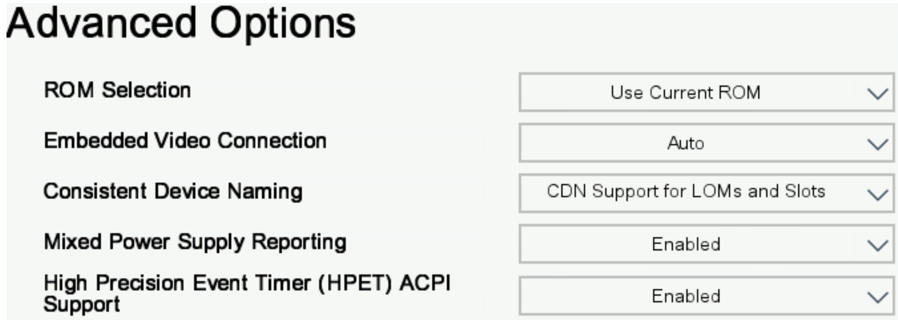
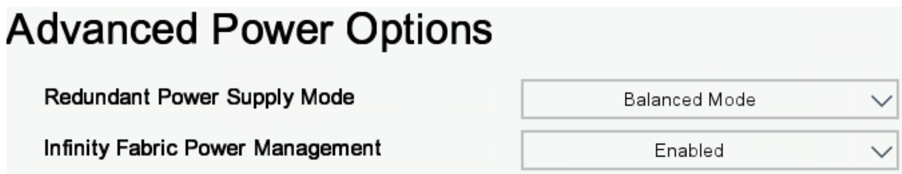
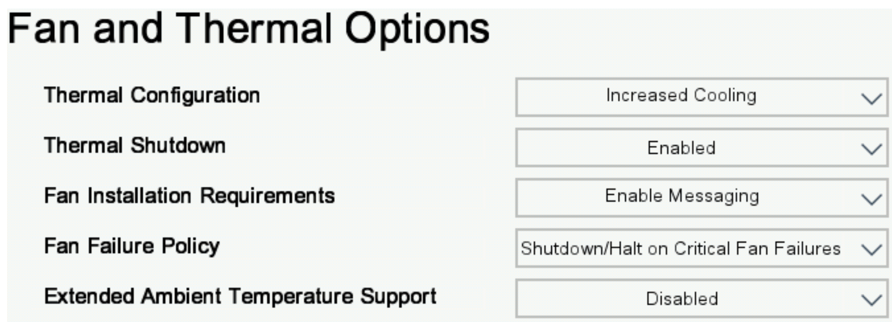
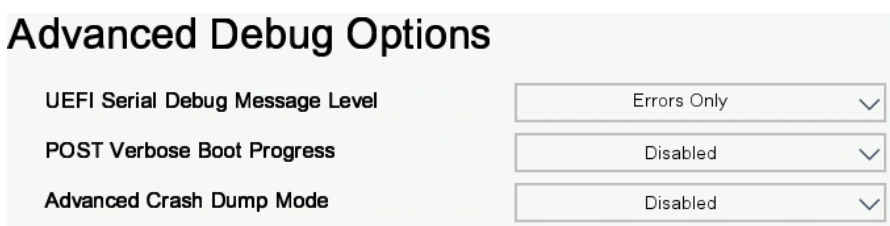
Memory Options	<u>Recommended</u>: This Low Latency test has been conducted with NPS=1 Setting only. 
Virtualization Options	
Boot Options	

<table>
<tr>
<td>Power and Performance Options</td>
<td>Option: You may increase higher CPU Core Performance by changing AMD Core Performance Boost to “Enabled”


</td>
</tr>
<tr>
<td>Processor Prefetcher Options</td>
<td>

</td>
</tr>
<tr>
<td>I/O Options</td>
<td>

</td>
</tr>
</table>

Advanced Options	
Advanced Power Options	
Fan and Thermal Options	<u>Option</u>: To increase more cooling change the Thermal Configuration to “Increased Cooling” 
Advanced Debug Options	

After the BIOS Settings are complete, save the settings and boot the server. While the server is booting up, check the initialization screen where it briefly displays the BIOS profile option which you've chosen in order to verify that your workload profile was properly selected.

Below are the screenshot examples for both the Low Latency and the Custom profile setting options.

Low Latency Profile

```
Workload Profile: Low Latency
Power Regulator Mode: Static High Performance
Advanced Memory Protection Mode: Advanced ECC Support
Boot Mode: UEFI
```

Custom Profile

```
Workload Profile: Custom
Power Regulator Mode: Static High Performance
Advanced Memory Protection Mode: Advanced ECC Support
Boot Mode: UEFI
```

After the system boots please run the following system checkup to make sure that your environment is ready for Linux OS level tunings described in the following sections.

3.3 BIOS, Memory, and CPU Version

The following output from the `xsos` utility shows BIOS, Memory and CPU Version.

```
# Instructions: https://access.redhat.com/discussions/469323
#
# yum install http://people.redhat.com/rsawhill/rpms/latest-rsawaroha-release.rpm
# yum install xsos rsar
# xsos --bios (produces the below output)
  BIOS:
    Vend: HPE
    Vers: A42
    Date: 04/29/2020
    BIOS Rev: 1.24
    FW Rev:   2.30 (iLO Version)
  System:
    Mfr:  HPE
    Prod: ProLiant DL385 Gen10 Plus
    Vers: Not Specified
    Ser:  M77932054N
    UUID: 474e4946-4c41-374d-3739-33323035344e
  CPU:
    2 of 2 CPU sockets populated, 8 cores/16 threads per CPU
    16 total cores, 32 total threads
```

```
    Mfr:  Advanced Micro Devices, Inc.
    Fam:  Zen
    Freq: 3700 MHz
    Vers: AMD EPYC 7F32 8-Core Processor
  Memory:
    Total: 524288 MiB (512 GiB)
    DIMMs: 16 of 160 populated
    MaxCapacity: 4194304 MiB (4096 GiB / 4.00 TiB)
```

3.3.1 NUMA Node

Output from **`numactl -H`** shows NUMA nodes.

```
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7
node 0 size: 257464 MB
node 0 free: 250483 MB
node 1 cpus: 8 9 10 11 12 13 14 15
node 1 size: 258018 MB
node 1 free: 251048 MB
node distances:
node   0   1
  0:  10  32
  1:  32  10
```

3.3.2 Processor

Output from `lscpu` shows two AMD EPYC 2-Socket 7F32 Processors.

```
# lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                16
On-line CPU(s) list: 0-15
Thread(s) per core:  1 (SMT is OFF in BIOS)
Core(s) per socket:  8
Socket(s):             2
NUMA node(s):          2
Vendor ID:             AuthenticAMD
CPU family:            23
Model:                 49
Model name:            AMD EPYC 7F32 8-Core Processor
Stepping:              0
CPU MHz:               2994.299
BogoMIPS:              7386.00
Virtualization:        AMD-V
L1d cache:             32K
L1i cache:             32K
L2 cache:              512K
L3 cache:              16384K
NUMA node0 CPU(s):   0-7
NUMA node1 CPU(s):   8-15
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 rep_good nopl xtopology nonstop_tsc cpuid extd_apicid aperfmperf 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 ibs skinit wdt tce topoext perfctr_core perfctr_nb bpext perfctr_llc mwaitx cpb
cat_l3 cdp_l3 hw_pstate ssbd mba ibrs ibpb stibp vmmcall fsgsbase bmi1 avx2 smep
bmi2 cqm rdt_a rdseed adx smap clflushopt clwb sha_ni xsaveopt xsavec xgetbv1 xsaves
cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local clzero irperf xsaveerptr wbnoinvd
arat npt lbrv svm_lock nrip_save tsc_scale vmcb_clean flushbyasid decodeassists
pausefilter pfthreshold avic v_vmsave_vmload vgif umip rdpid overflow_recov succor
smca
```

3.3.3 DRAM

Output from **dmidecode** (-t 17 → Memory) shows DRAM details.

```
Handle 0x0027, DMI type 17, 84 bytes
Memory Device
        Array Handle: 0x0010
        Error Information Handle: Not Provided
        Total Width: 72 bits
        Data Width: 64 bits
        Size: 32 GB
        Form Factor: DIMM
        Set: None
        Locator: PROC 1 DIMM 1
        Bank Locator: Not Specified
        Type: DDR4
        Type Detail: Synchronous Registered (Buffered)
        Speed: 3200 MT/s
        Manufacturer: Samsung
        Serial Number: 39CB333
        Asset Tag: Not Specified
        Part Number: M393A4G43AB3-CWE
        Rank: 2
        Configured Memory Speed: 3200 MT/s
        Minimum Voltage: 1.2 V
        Maximum Voltage: 1.2 V
        Configured Voltage: 1.2 V
        Memory Technology: DRAM
        Memory Operating Mode Capability: Volatile memory
        Firmware Version: Not Specified
        Module Manufacturer ID: Bank 1, Hex 0xCE
        Module Product ID: Unknown
        Memory Subsystem Controller Manufacturer ID: Unknown
        Memory Subsystem Controller Product ID: Unknown
        Non-Volatile Size: None
        Volatile Size: 32 GB
        Cache Size: None
        Logical Size: None
```

Chapter 4 Operating System

The operating system in the SUT is RedHat Enterprise Linux (RHEL) 8.1. Output from **`xsos --os`** shows Operating System Details.

```
OS
  Hostname: hpe-lowlat.amd.com
  Distro:  [redhat-release] Red Hat Enterprise Linux release 8.1 (Ootpa)
            [os-release] Red Hat Enterprise Linux 8.1 (Ootpa) 8.1 (Ootpa)
  RHN:      serverURL = https://enter.your.server.url.here/XMLRPC
            enableProxy = 0
  RHSM:     hostname = subscription.rhsm.redhat.com
            proxy_hostname =
  YUM:      3 enabled plugins: debuginfo-install, product-id, subscription-manager
  Runlevel: N 3  (default multi-user)
  SELinux:  disabled  (default disabled)
  Arch:     mach=x86_64  cpu=x86_64  platform=x86_64
  Kernel:
    Booted kernel:  4.18.0-147.5.1.el8_1.x86_64
    GRUB default:
    Build version:
      Linux version 4.18.0-147.5.1.el8_1.x86_64 (mockbuild@x86-vm-
07.build.eng.bos.redhat.com) (gcc version 8.3.1 20190507 (Red Hat 8.3.1-4) (GCC)) #1
SMP Tue Jan 14 15:50:19 UTC 2020
```

4.1 SUT Topology

After installing the OS, you can view the SUT physical topology using the **`lstopo`** topology tool in the command line interface.

For RHEL 8, use the following repositories 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. Subscriptions are required for repositories in RHEL 8.1. Below are the required subscriptions for **`lstopo`**.

```
Subscription for Red Hat 8
# subscription-manager repos --enable=rhel-8-for-x86_64-supplementary-rpms
# subscription-manager repos --enable=rhel-8-for-x86_64-baseos-source-rpms
# subscription-manager repos --enable=rhel-8-for-x86_64-appstream-source-rpms
# yum install hwloc hwloc-gui -y
```

4.1.1 System Topology through lstopo

The lstopo tool provides multiple output formats. Figure 1 is an example of graphical output generated by the following command line.

```
# lstopo --physical --output-format png AMD_EPYC_7F32_8-Core_Processor.png
```

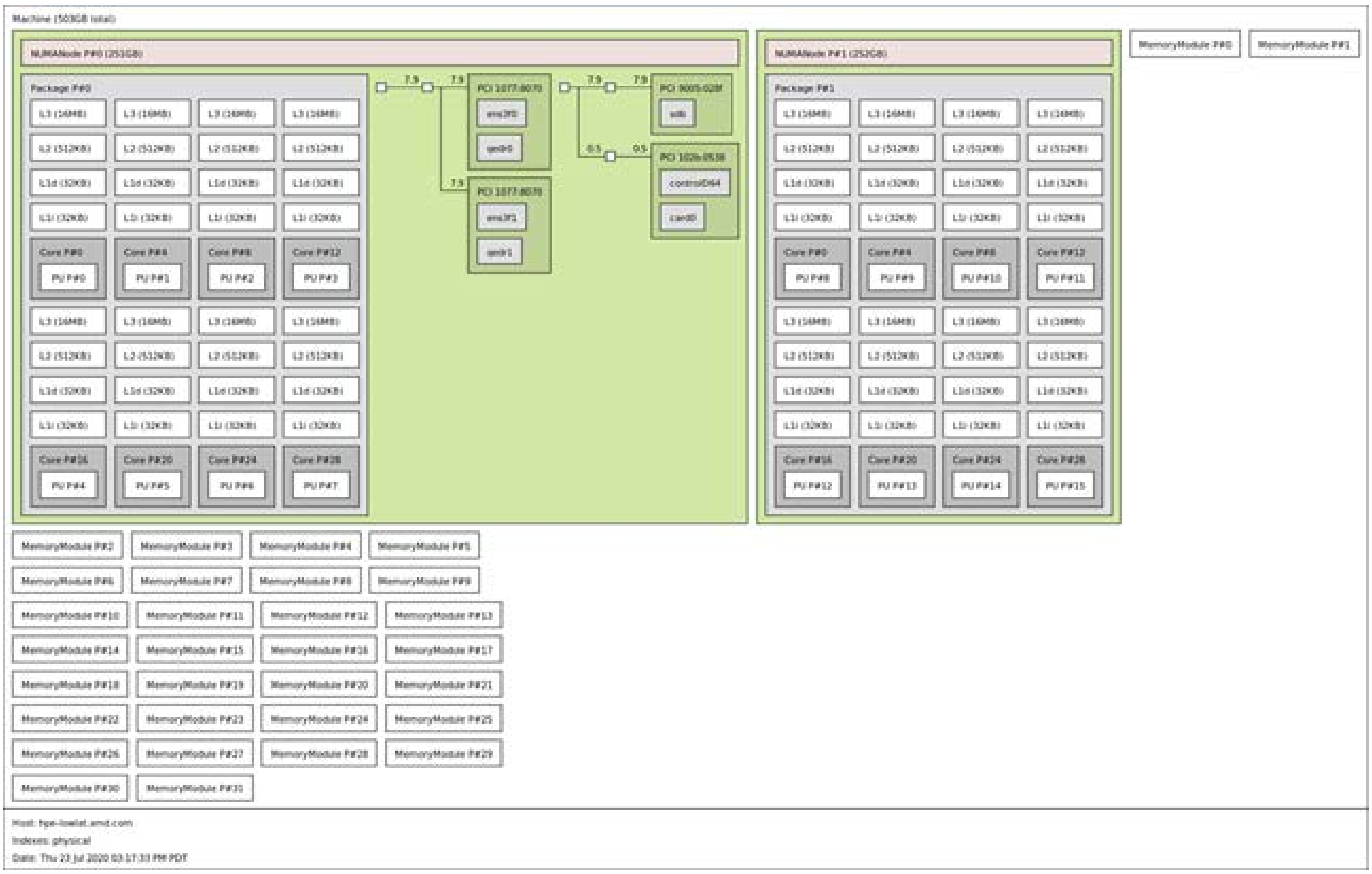


Figure 1 lstopo output (png) for AMD EPYC® Two-Processor High-Frequency 7F32 SUT

4.2 Configuring and Tuning RHEL 8.1 for Low Latency Performance

This section provides guidelines to configure RHEL 8.1 that would enable the two processor AMD EPYC 7F32 based system to achieve the optimal performance for attaining the best low latency performance with low jitter for various Financials, Trade and Matching use cases.

4.2.1 The /proc filesystem

The /proc filesystem interface can expose per processor information such as the internal kernel data, the kernel subsystems, and system devices. To realize Low Latency performance, configure the SUT with the following kernel parameters.

4.2.1.1 Tuning Kernel Parameters Using sysctl

The settings of these parameters have been determined through various iterations and experiments, and they can be made persistent by writing the values into a custom sysctl configuration file and added it to the /etc/sysctl.d directory.

The below table contains experimented system kernel parameters to override default kernel parameter values written into a file called 100-lowlatency.conf to help achieve the jitter-less system. 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:

These settings are read from the /etc/sysctl.d/100-lowlatency.conf file at boot time.

1. Create a file **`/etc/sysctl.d/100-lowlatency.conf`** and populate it with the following parameters

```
# vi /etc/sysctl.d/100-lowlatency.conf
```

Note: Put the below script inside (*Please watch out for any line wrapping issues*)

```
kernel.hung_task_timeout_secs = 600
kernel.numa_balancing = 0
kernel.numa_balancing_scan_delay_ms = 1000
kernel.numa_balancing_scan_period_max_ms = 60000
kernel.numa_balancing_scan_period_min_ms = 1000
kernel.numa_balancing_scan_size_mb = 256
kernel.sched_latency_ns = 24000000
kernel.sched_migration_cost_ns = 50000000
kernel.sched_min_granularity_ns = 100000000
kernel.sched_rt_runtime_us = -1
kernel.timer_migration = 1
kernel.watchdog = 0
kernel.watchdog_cpumask = 0,8
kernel.watchdog_thresh = 10
vm.dirty_background_ratio = 3
vm.dirty_ratio = 10
vm.stat_interval = 3600
vm.swappiness = 0
vm.zone_reclaim_mode = 0
net.ipv4.conf.all.rp_filter =  0
```

To have the sysctl conf file read and the settings put in effect immediately, the following command can be used. Run the following command as root to apply these new parameters and make sure there are no errors.

```
# sysctl -p /etc/sysctl.d/100-lowlatency.conf
```

4.3 Tuning the system with Linux Tuned

You can tune additional parameters using Linux TUNED. Linux TUNED is a customizable application that helps to tune and optimize your system to reach its best performance. The TUNED application includes profiles with predefined tuning options such as throughput-performance, latency-performance, and cpu-partitioning. You can use these tuned-adm profiles instead of writing directly to the /proc filesystem, using **`sysctl`**, and applying boot command options. However, if you stopped or disabled the Tuned daemon, you must restart it to apply Tuned profiles.

4.3.1 Setting up the Custom HPELowLatency Profile using Linux TUNED

This section describes the process of preparing your server using a customized TUNED profile for low latency tuning. The HPELowLatency profile modifies several system kernel settings as well as configuring boot time options and performing other tuning operations. In addition, a “one-time" Linux Service is added to the System startup process, executing a “*one-shot.sh*” shell script during system boot. You can create the indicated files using the steps below and apply them without any additional modifications to your HPE ProLiant Gen10 Plus server with **two** AMD EPYC High Frequency processors, such as the 7F32.

> When you copy and paste, be sure to inspect the code. The code page lines are wrapped due to the page size of this document, so after copying check for any unwanted spaces, comments, and line breaks, and resolve those issues before executing.

Be sure that TUNED is configured for static settings. To reduce jitter and attain Low Latency performance, instead of using a predefined profile, we are creating a customized *HPELowLatency* Tuned profile.

Follow the steps described below to create the custom **HPELowLatency** tuned profile and modify the **tuned.conf** and **cpu-partitioning.conf** files.

4.3.2 Preparing for HPELowLatency using TUNED Custom Profile

4.3.2.1 Configuration 1 – HPELowLatency

4.3.2.1.1 Add the configuration settings in "*tuned.conf*" under /etc/tuned/HPELowLatency directory

```
# mkdir /etc/tuned/HPELowLatency
# vi /etc/tuned/HPELowLatency/tuned.conf
```

```
Note: Put the below script inside (Please watch out for any line wrapping issues)
```

```
##############################START########################################
#! HPELowLatency includes several files:\
#! This file, /etc/tuned/HPELowLatency/tuned.conf, which is a tuned profile based on
the cpu-partitioning profile
#! A helper script file /etc/tuned/HPELowLatency/script.sh that is used by cpu-
partitioning-variables.conf, /usr/local/bin/oneshot_script.sh and /opt/run_sysjit.sh
script to Run Sysjitter program in Chapter 5
#! The global tuned configuration file /etc/tuned/tuned-main.conf
#! /etc/tuned/cpu-partitioning-variables.conf, which sets variable for the cpu-
partitioning profile
#!
#! In addition to the "tuned" files are these systemd startup files
#!   /etc/systemd/system/one-time.service that defines a service to execute commands
at startup
#!   /usr/local/bin/oneshot_script.sh, which is a script to run at startup to affect
additional changes.

[main]
summary=Related to HPELowLatency modifications and its cpu-partitioning profile
description=Modifies the cpu-partitioning profile to leave the first CPU of each
NUMAnode for housekeeping, allow balancing on the next few, and disable balancing on
the rest.

include=cpu-partitioning

[disk]
elevator=none

[bootloader]
#! A problem in tuned and/or grub is causing the first word to get eaten up.
#! The recommendation is as follows, quoting my Red Hat contact:
#! The workaround used is to just add the truncated boot flags you're
#! missing to the GRUB_CMDLINE_LINUX variable in the /etc/default/grub file.
#! Then run "grub2-mkconfig -o /boot/grub2/grub.cfg" to pick it up, and reboot.
#! Using a workaround of a throw-away first word, add "quiet" as the first option to
#! cmdline. Please make sure that the below cmdline is a one line entry and watch out
#! for line wrapping.

cmdline=quiet selinux=0 mce=ignore_ce ipv6.disable=1 audit=0 nmi_watchdog=0
hugepagesz=2MB hugepages=6000 default_hugepagesz=2MB transparent_hugepage=never
tsc=reliable pcie_aspm=off cpuidle.off=1 rcu_nocb_poll idle=poll processor.max_cstate=0

#! Reference from https://tuned-project.org/docs/tuned_devconf_2019.pdf
[scheduler]
group.ksoftirqd=${f:exec:/etc/tuned/HPELowLatency/script.sh:ksoftirqd}
group.rcub=${f:exec:/etc/tuned/HPELowLatency/script.sh:rcub}
##############################END########################################
# Note:
# a.   Kernel Boot Parameters "cmdline" are implemented with tuned.conf file and
# experiment your preferred boot time arguments as permanent.
# b.   cat /etc/default/grub and make sure your changes are in effect after the
# final reboot.
##############################END########################################
```

This new **HPELowLatency** Tuned profile also includes the "cpu-partitioning" profile and its associated file **cpu-partitioning-variables.conf (See Section 4.3.2.1.3)**, which sets the housekeeping and isolated cores appropriately for the installed AMD EPYC High Frequency processors.

4.3.2.1.2 Add the Tuned "*script.sh*" under /etc/tuned/HPELowLatency directory

```
# vi /etc/tuned/HPELowLatency/script.sh
```

Note: Put the below script (*Please watch out for any line wrapping issues*)

```
#!/bin/bash
##############################START#########################################
function range_expand () {
  eval /usr/bin/printf "%s" $(/usr/bin/printf $*| \
  sed -r 's:.*:{&}:;s:([0-9]+)-([0-9]+):{\1..\2}:g;s:^\{([^,]*)\}$:\1:');
}

#! Not used
function getPopulatedNodes () {
  local NodeList
  local Node

  NodeList=($(range_expand $(cat /sys/devices/system/node/has_cpu)))
  if [ "${NodeList}" = "" ]; then
    for Node in \
        $(cd /sys/devices/system/node/; ls -d node*| sed '/node[0-9]\{1,\}/s/node//')
    do
     NodeList=(${NodeList[*]} $(/usr/bin/echo $(($(/usr/bin/echo \
     "16 i $(sed 's/,//g;s/.*/\U&/' /sys/devices/system/node/node${Node}/cpumap) p" | \
     dc) == 0 ? 0 : $Node))))
    done
  fi
  /usr/bin/printf "${NodeList[*]}"
}

function _HousekeepingCpus () {
  local Params
  local List
  local HT
  local Cores
  local Processors
  local Count

#! "Params" gets ThreadsPerCore CoresPerProcessor Processors
  Params=($(/usr/bin/lscpu | \
  /usr/bin/awk \
'/(^Thread\(s\) per core:|^Core\(s\) per socket:|^[CPU ]*[sS]ocket\(s\):)/{print $NF}'))
  unset List
  for HT in $(seq 0 $((${Params[0]}-1))); do
    for Cores in ${Params[1]}; do
      for Processors in $(seq 0 $((${Params[2]}-1))); do
        for Count in $(seq 0 $((${HousekeepingCores}-1))); do
List=(${List[*]} $((${HT}*${Params[1]}*${Params[2]}+${Processors}*${Cores}+${Count})))
        done
      done
    done
  done
  /usr/bin/printf "$(echo ${List[*]}| sed 's/ /,/g')"
}

function _NoBalanceCpus () {
  local Params
  local List
```

```
  local HT
  local Cores
  local Processors
  local Count

  Params=($(/usr/bin/lscpu | \
  /usr/bin/awk \
'/(^Thread\(s\) per core:|^Core\(s\) per socket:|^[CPU ]*[sS]ocket\(s\):)/{print $NF}'))
  unset List
  for HT in $(seq 0 $((${Params[0]}-1))); do
    for Cores in ${Params[1]}; do
      for Processors in $(seq 0 $((${Params[2]}-1))); do
        for Count in $(seq ${HousekeepingCores} ${NoBalanceCores}); do
List=(${List[*]} $((${HT}*${Params[1]}*${Params[2]}+${Processors}*${Cores}+${Count})))
        done
      done
    done
  done
  /usr/bin/printf "$(/usr/bin/echo ${List[*]}| sed 's/ /,/g')"
}

function _IsolatedCpus () {
  _NoBalanceCpus
}

function _HousekeepingMask () {
  local Params
  local Mask=0
  local HT
  local Cores
  local Processors
  local Count

#! "Params" gets ThreadsPerCore CoresPerProcessor Processors
  Params=($(/usr/bin/lscpu | \
  /usr/bin/awk \
'/(^Thread\(s\) per core:|^Core\(s\) per socket:|^[CPU ]*[sS]ocket\(s\):)/{print $NF}'))
  for HT in $(seq 0 $((${Params[0]}-1))); do
    for Cores in ${Params[1]}; do
      for Processors in $(seq 0 $((${Params[2]}-1))); do
        for Count in $(seq 0 $((${HousekeepingCores}-1))); do
Mask=$((Mask|=1<<(${HT}*${Params[1]}*${Params[2]}+${Processors}*${Cores}+${Count})))
        done
      done
    done
  done
  /usr/bin/printf "%x" $Mask
}

function _NoBalanceMask () {
  local HKMask=0x$(_HousekeepingMask)
  local AllMask=0x$((-1<<$(grep -c processor /proc/cpuinfo)))
  local Mask=$((~$HKMask^$AllMask))
  /usr/bin/printf "%x" $Mask
}


function _IsolatedMask () {
  _NoBalanceMask
}

function _ksoftirqd () {
  local Mask=$(_HousekeepingMask)
```

```
  /usr/bin/printf '0:f:2:%s:ksoftirqd.*' $Mask
}

function _rcub () {
  local Mask=$(_HousekeepingMask)
  /usr/bin/printf '0:f:4:%s:rcub.*' $Mask
}

#! function main () {
#! These are the number of CPUs with this designation.  A negative number means "all but"
#! They are per NUMAnode:
  HousekeepingCores=1
  NoBalanceCores=$(($(/usr/bin/lscpu | \
  /usr/bin/awk '/^Core\(s\) per socket:/{print $NF}')-${HousekeepingCores}))
  if [ "${DefineFunctionsOnly}" != "1" ]; then
    Workload=_${1}
    shift
    if [ "$(type -t ${Workload})" = "function" ] ; then
     ${Workload} $*
    fi
  fi
#! }
##############################END#######################################
```

```
# chmod +x /etc/tuned/HPELowLatency/script.sh
```

Be aware of the following:

- Due to a GRUB generation bug, the first token in “cmdline” gets consumed and lost. As a workaround, add the single-token option “quiet” as the first token.
- Kernel Boot Parameters “cmdline” are implemented with **tuned.conf** file. Specify your desired boot time options in that section.
- Before rebooting the server, check the TUNED_BOOT_CMDLINE in the file **`/etc/tuned/bootcmdline`** after applying the tuned profile.
- After the boot please look at **`/proc/cmdline`** where the reboot has taken the Tuned parameters into account.

4.3.2.1.3 Add the Tuned "*cpu-partitioning-variables.conf*" under /etc/tuned directory

Edit the existing **cpu-partitioning-variables.conf** file and add/modify the below entries to reflect Isolated CPU(s).

```
# vi /etc/tuned/cpu-partitioning-variables.conf
```

```
Note: Put the below script (Please watch out for any line wrapping issues)
```

```
#############################START########################################
#! This file is included in the [variables] section of
# /usr/lib/tuned/cpu-partitioning/tuned.conf
#
housekeeping_cpus=${f:exec:/etc/tuned/HPELowLatency/script.sh:HousekeepingCpus}
isolated_cores=${f:cpulist_invert:${housekeeping_cpus}}
no_balance_cores=${f:exec:/etc/tuned/HPELowLatency/script.sh:NoBalanceCpus}
#
##############################END########################################
```

4.3.3 Preparing and Setting up one-time.service script

Execute the BASH script, which has been tailored for a server with two AMD EPYC High Frequency processors such as 7F32 (8 Core) processors, during the boot process to set the required Linux kernel parameters.

4.3.3.1 Configuration 2 → Setting up one-time.service and oneshot_script.sh

4.3.3.1.1 Add the systemctl service settings in "one-time.service" under /etc/systemd/system/ directory

Below command will open a new service file called "*one-time.service*" in VI editor and add the below configuration settings

```
# SYSTEMD_EDITOR=/bin/vi /usr/bin/systemctl edit --force --full one-time.service
```

```
Note: Put the below script (Please watch out for any line wrapping issues)
```

```
##############################START########################################

[Unit]
Description=Execute this script to setup the Low Jitter environment from CPU cores to Network
After=network.target

[Service]
Type=oneshot
ExecStart=/usr/local/bin/oneshot_script.sh
TimeoutStartSec=0

[Install]
WantedBy=default.target

##############################END########################################
```

4.3.3.1.2 Add the /usr/local/bin/oneshot_script.sh under /usr/local/bin directory

Follow the steps below to complete reconfiguration and to reboot the system:

```
# vi /usr/local/bin/oneshot_script.sh
```

Note: Put the below script (*Please watch out for any line wrapping issues*)

```
#!/bin/bash
########################################START##########################################
#
#
# Script Written by
# Sylvester Rajasekaran, DEAE, AMD Inc.,  Chuck Newman, HPE
# October 30th, 2020
#
#
#######################################################################################
#
# /usr/local/bin/oneshot_script.sh is an environment setup script for an
# AMD EPYC Hi-Frequency Processors based server (e.g.) 7F32 (2 Socket x 8 Cores).
# This automatic script would help customers in HFT/FSI domains to observe
# Jitters via any Jitter observing Programs such as "sysjitter"
#
################################ T  O  P  O  L  G  Y #################################
#
# Example: In this System Topology scenario there are
# (a) 2 Sockets 7F32 (2 Socket x 8 Cores)
# (b) 512GB DRAM (Memory) with a NUMA Topology setting in BIOS as NPS=1
#
# ### NUMA - inventory of available nodes on the system ###############################
# numactl -H
#               available: 2 nodes (0-1)
#               node 0 cpus: 0 1 2 3 4 5 6 7
#               node 0 size: 257578 MB
#               node 0 free: 254656 MB
#               node 1 cpus: 8 9 10 11 12 13 14 15
#               node 1 size: 258043 MB
#               node 1 free: 255865 MB
#               node distances:
#               node   0   1
#               0:  10  32
#               1:  32  10
#
# ### NUMA - NUMA policy settings
# numactl -s
#
#               policy: default
#               preferred node: current
#               physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
#               cpubind:  0 1
#               nodebind: 0 1
#               membind:  0 1
#
################################ S C R I P T   D E T A I L S ##########################
#
# This script gets executed during boot time as part of systemd
# (systemctl {enable|startup|stop|status} one-time.service)
#  called by the below service script to execute
#               /etc/systemd/system/one-time.service
#                                   ==> /usr/local/bin/oneshot_script.sh
```

```
#
#######################################################################################
#
#       Reference: https://access.redhat.com/solutions/3152271
#
#       Example: AMD EPYC 7F32 (8 Cores) Dual Socket system
#
#               Socket 1                Socket 0
#               0 0 0 0 0 0 0 0         0 0 0 0 0 0 0 0 --> All are ALLOWED by default
#               1 1 1 1 1 1 1 0         1 1 1 1 1 1 1 0 -->
#               (where Core# 0 and 8 are Housekeeping Cores dedicated for Linux OS and
#                leaving the rest of the 14 Cores as ISOLATED Cores for Jitter
#                observation)
#               which turns in to (1111111011111110)    ===> AllowedOSBINCores="0101"
#CPU HEX MASK           fe                    fe        ===> AllowedCPUMask="fefe"
#CPU CORES LIST FOR OS        8                        0 ===> AllowedOSCPUCores="0,8"
#
#
#######################################################################################
#
#
mpid=$$
LOGDIR=/opt/$(date +%m%d%Y_%H%M%S)
mkdir -p ${LOGDIR}
#
#+++ IMPORTANT NOTE +++
# Example:
# AllowedCPUMask="fefe"
# AllowedOSCPUCores="0,8"
# AllowedOSBINCores="0101"
#
# Variable Assignments are automatically assigned based on the type of AMD EPYC Processor SKU
# by calling /etc/tuned/HPELowLatency/script.sh shell script to get
# the individual parameters and pass it on to the respective below variables
#               (AllowedCPUMask, AllowedOSCPUCores, AllowedOSBINCores)
#

DefineFunctionsOnly=1; . /etc/tuned/HPELowLatency/script.sh
AllowedCPUMask="$(_IsolatedMask)"
AllowedOSCPUCores="$(_HousekeepingCpus)"
AllowedOSBINCores="$(_HousekeepingMask)"
echo -e "Allowed CPU MASK (Hex):         ${AllowedCPUMask}"
echo -e "Allowed CPU Cores for OS:       ${AllowedOSCPUCores}"
echo -e "Allowed CPU Cores for OS (BIN): ${AllowedOSBINCores}"
#
# 0. Set and Read New Sysctl values which are required for Less Jitter
#
sysctl -p /etc/sysctl.d/100-lowlatency.conf
#
# 1. Checking whether the System's Tuned Profile set to "HPELowLatency"
#    which includes "cpu-partitioning"
echo "Checking current tuned profile"
if [ "${CurrentProfile}" != "HPELowLatency" ]
then
   tuned-adm profile HPELowLatency
   tuned-adm active | tee -a ${LOGDIR}/tuned_profile_active_${mpid}.log
   cat ${LOGDIR}/tuned_profile_active_${mpid}.log
fi
#
# 2. Set affinity for the writeback threads. These threads should be moved to only the
#    housekeeping cpus in our case ${AllowedOSBINCores}
#
echo 0 > /sys/bus/workqueue/devices/writeback/numa
```

```
cat /sys/bus/workqueue/devices/writeback/numa
echo ${AllowedOSBINCores} > /sys/bus/workqueue/devices/writeback/cpumask
cat /sys/bus/workqueue/devices/writeback/cpumask | tee -a ${LOGDIR}/cpumask_${mpid}.log
#
# 3. Setting All CPU Cores Governor to Performance
NP=$(grep -c processor /proc/cpuinfo)
if [ -e /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor ]; then
        for sg in $(seq 0 $((${NP}-1))); do
           echo performance > /sys/devices/system/cpu/cpu${sg}/cpufreq/scaling_governor
        done
        for sg in $(seq 0 $((${NP}-1))); do
           cat /sys/devices/system/cpu/cpu${sg}/cpufreq/scaling_governor
        done | tee -a ${LOGDIR}/cpu_scaling_gov_${mpid}.log
fi
#
# 4. Switching OFF and ON of the Cores (Except Housekeeping Core 0 and 8)
echo -e "It takes few more seconds... Please wait..."
NP=$(grep -c processor /proc/cpuinfo)
for OffOn in 0 1; do
        for Cpu in $(seq 1 $((${NP}-1))) $(seq $((${NP}/2+1)) $((${NP}-1))); do
           echo ${OffOn} > /sys/devices/system/cpu/cpu${Cpu}/online
           sleep 1
        done
done
if [ -e /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor ]; then
        for sg in $(seq 1 $((${NP}-1))); do
           cat /sys/devices/system/cpu/cpu${sg}/cpufreq/scaling_governor
        done | tee -a ${LOGDIR}/cpu_scaling_gov_${mpid}.log
fi
#
# 5. Flush/Clear unwanted IPV4/IPV6 Network related stuff
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 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 ipv6t_REJECT xt_LOG xt_multiport \
            nf_conntrack
#
# 6. RCU (READ COPY UPDATE) to use only the Housekeeping Cores
# We have already set the cpu-partitioning profile in Step 1 to
# exclude (in this case Core 0 and 8) for rcu tasks at post-boot,
# ${AllowedOSCPUCores} for our housekeeping CPUs
#
for i in `pgrep rcu`; do taskset -pc ${AllowedOSCPUCores} $i; done
#
# 7. Modifying by forcing the IRQ Values to use ${AllowedOSBINCores}
#
for irq in /proc/irq/*/smp_affinity; do
        echo ${AllowedOSBINCores} > $irq 2>/dev/null
done | tee -a ${LOGDIR}/IRQ_new_values_${mpid}.log
# cat ${LOGDIR}/IRQ_new_values_${mpid}.log
#
# 8. Modify kswapd to use only the OS Cores
for i in `pgrep kswapd?`; do taskset -p ${AllowedOSBINCores} $i; done
for i in `pgrep kswapd?`; do
   taskset -cp $i
done | tee -a ${LOGDIR}/kswapd_new_values_${mpid}.log
```

```
# 9. Switching OFF unwanted Systemd services
for SERVICE in  avahi-daemon.service bluetooth.service chronyd.service \
                crond.service dbus.service dnsmasq.service dnsmasq.service \
                firewalld.service firewalld.service iprdump.service iprinit.service \
                iprupdate.service kdump.service ksm.service libstoragemgmt.service \
                libvirtd.service lvm2-monitor.service mcelog.service \
                mdmonitor.service messagebus.service ModemManager.service \
                nfs-client.target postfix.service rhnsd.service rhsmcertd.service \
                rpcbind.service rpcbind.socket systemd-journald.service tuned.service \
                systemd-journald.socket virtlogd.socket wpa_supplicant.service
do
        systemctl stop $SERVICE
        systemctl is-active $SERVICE | \
                tee -a ${LOGDIR}/inactive_service_status_${mpid}.log
done
########################################END########################################
```

```
# chmod +x /etc/systemd/system/one-time.service
# chmod +x /usr/local/bin/oneshot_script.sh
# systemctl daemon-reload
# systemctl enable one-time.service
# systemctl list-unit-files one-time.service --state=enabled
UNIT FILE        STATE
one-time.service enabled
1 unit files listed.
```

```
# systemctl reboot
```

The one-time.service will call the oneshot_script.sh when the system boots and comes back online and implements all the required settings as per the script.

Once the system is back online, login as root and check whether the user PID(s) are associated with only the first CPU of each processor.

```
# taskset -cp $$
pid 7032's current affinity list: 0,8
```

4.3.4 Verify the Default Boot Kernel to RHEL 8 (8.1) (Example)

```
# grubby --default-kernel
/boot/vmlinuz-4.18.0-147.5.1.el8_1.x86_64
```

4.3.5 Regenerated Grubby Information of RHEL 8 (8.1) (Example)

```
# grubby --info /boot/vmlinuz-4.18.0-147.5.1.el8_1.x86_64
index=1
kernel="/boot/vmlinuz-4.18.0-147.5.1.el8_1.x86_64"
args="ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet
kernelopts=root=/dev/mapper/rhel-root ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap rhgb quiet $tuned_params quiet selinux=0 mce=ignore_ce ipv6.disable=1 audit=0
nmi_watchdog=0 hugepagesz=2MB hugepages=6000 default_hugepagesz=2MB transparent_hugepage=never
tsc=reliable pcie_aspm=off cpuidle.off=1 rcu_nocb_poll idle=poll processor.max_cstate=0 nohz=on
nohz_full=1,2,3,4,5,6,7,9,10,11,12,13,14,15 rcu_nocbs=1,2,3,4,5,6,7,9,10,11,12,13,14,15
tuned.non_isolcpus=00000101 intel_pstate=disable nosoftlockup $tuned_params"
root="/dev/mapper/rhel-root"
initrd="/boot/initramfs-4.18.0-147.5.1.el8_1.x86_64.img $tuned_initrd"
title="Red Hat Enterprise Linux (4.18.0-147.5.1.el8_1.x86_64) 8.1 (Ootpa)"
id="16699b9231234d6f83e12e4549b18673-4.18.0-147.5.1.el8_1.x86_64"
```

Now reboot the system using the following command:

```
# systemctl reboot
```

4.4 Verifying Boot Parameter Configuration After Reboot

4.4.1 Boot Parameters

After rebooting, check the running kernel to verify that the required settings including the boot parameters options set by HPELowLatency Profile in TUNED are enabled.

```
# cat /proc/cmdline

Note: This command line information shows the server with 2 AMD EPYC High Frequency
processors such as the AMD EPYC 7F32 (8 core each) and how the cores are isolated
according to the HPELowLatency Tuned Profile.
```

```
BOOT_IMAGE=(hd1,gpt2)/vmlinuz-4.18.0-147.5.1.el8_1.x86_64 root=/dev/mapper/rhel-root ro crashkernel=auto
resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet selinux=0 mce=ignore_ce
ipv6.disable=1 audit=0 nmi_watchdog=0 hugepagesz=2MB hugepages=6000 default_hugepagesz=2MB
transparent_hugepage=never tsc=reliable pcie_aspm=off cpuidle.off=1 rcu_nocb_poll idle=poll
processor.max_cstate=0 nohz=on nohz_full=1,2,3,4,5,6,7,9,10,11,12,13,14,15
rcu_nocbs=1,2,3,4,5,6,7,9,10,11,12,13,14,15 tuned.non_isolcpus=00000101 intel_pstate=disable nosoftlockup
```

Chapter 5 Sysjitter

Solarflare' s sysjitter utility measures the extent to which the system introduces jitter and so impacts on the user level process. Sysjitter runs a thread on each processor core and when the thread is interrupted from the core it measures for how long. A common cause of such jitter is when another task is scheduled on the core, although shorter duration events such as cache misses can also be detected if the jitter threshold is set low enough. Examples of tasks that induce jitter are other user processes that the scheduler schedules on the core, and even the scheduler itself will induce jitter.

Sysjitter produces summary statistics for each processor core. The sysjitter utility can be downloaded from www.openonload.org

5.1 Installing Sysjitter

The sysjitter utility can be downloaded from https://www.openonload.org/download/sysjitter/. As of the time this document was written, the latest version of sysjitter is version 1.4. See the sysjitter README file for details on building and running sysjitter.

For Example, in our case we have installed the sysjitter tool on */opt/sysjitter-1.4* directory.

5.2 Running Sysjitter

After installing sysjitter, you need to create a Bash script as below and call it as /opt/run_sysjit.sh. See the below Sysjitter execution Script for any AMD EPYC High Frequency processors such as the 7F32.

```
# vi /opt/run_sysjit.sh
# chmod +x /opt/run_sysjit.sh
```

Note: Put the below script (*Please watch out for any line wrapping issues*)

```
#!/bin/bash
##############################START########################################
tput clear
myhost=$(hostname -s)
mydate=$(date +%Y%m%d%H%M%S%Z)
repdir=/lowlatency/${mydate}
echo -e "Target Directory: ${repdir}/${myhost}.${mydate}"
mkdir -p ${repdir}
jitfile="${repdir}/sysjitter.${myhost}.${mydate}"
#
#
# Function to get the # of NoBalancCpus
#
# Variable Assignment Automatically based on Type of CPU and getting the cores by
# calling /etc/tuned/HPELowLatency/script.sh BASH script
# to get the # of Cores of the Processor which are
# NOT HouseKeeping Cores (_NoBalanceCpus) parameter and pass it on to lim_cores
variable
#
# Example:
#       On a 2 Socket AMD EPYC 7F32 x 8 Cores without HT = 16 Cores Total
#       OS Housekeeping Cores are 0 and 8
#       Thus, observing jitter in these cores 1,2,3,4,5,6,7,9,10,11,12,13,14,15
#
DefineFunctionsOnly=1; . /etc/tuned/HPELowLatency/script.sh
lim_cores=$(_NoBalanceCpus)
printf "These are the List of Cores will be observed by Sysjitter \n${lim_cores}\n"
#
# Collect Jitter Data
#
echo -e ""
echo -e "Jitter Data Collection has started using /opt/run_sysjit.sh script"
echo -e "using sysjitter program for Cores ${lim_cores}"
echo -e "with ${1} Threshold for ${2} seconds..."
echo -e ""
echo -e "\tPlease wait..."
read a
time /opt/sysjitter-1.4/sysjitter \
        --raw ${jitfile} --runtime ${2} --cores ${lim_cores} ${1} > ${jitfile}.txt
column -t ${jitfile}.txt | tee -a ${jitfile}.tab
echo -e ""
echo -e ""
echo -e "Jitter Data Collection is now complete"
echo -e "using sysjitter for Cores ${lim_cores} with ${1} Threshold for ${2}
seconds..."
echo -e "Jitter Data Collection is now complete for Cores ${lim_cores}"
echo -e "with ${1} Threshold for ${2} seconds...\nCheck file at ${jitfile}.tab file"
echo -e ""
echo -e ""
####################################END########################################
```

5.2.1 Preparation and Checkup

Before running Sysjitter make sure your System is healthy and ready for the Jitter observation.

- Load Average values are close to 0.00 (e.g.) see below and good to execute the script

```
# uptime
 12:17:08 up 46 days,  1:04,  2 users,  load average: 0.00, 0.00, 0.00

It should not be like the one below (Wait for some more minutes when the system load
goes to near 0.00)

# uptime
 12:36:21 up 46 days,  1:24,  2 users,  load average: 7.22, 2.61, 0.93
```

- Check and confirm your system are set with "HPELowLatency" Tuned Profile and your Terminal/Console where you are running Sysjitter program is currently set with Housekeeping Cores only.

```
# tuned-adm active
It seems that tuned daemon is not running, preset profile is not activated.
Preset profile: HPELowLatency
Note:
•     The above message shows that the system's TUNED Profile is now set with
HPELowLatency.
•     It is intentional that the TUNED Daemon is not running as we have switched off
the System Level Daemon through our startup Bash script (oneshot_script.sh) on Step
# 9
# taskset -cp $$
pid 16382's current affinity list: 0,8
```

5.3 Launching Sysjitter

As a root user, go to the Sysjitter directory /opt/sysjitter-1.4. where you have created the Bash, script called (***run_sysjit.sh***)

```
# cd /opt/sysjitter-1.4
# ./run_sysjit.sh 100 610
```

Where,

1. CONSTANT 100 (Preferred) → is the threshold in (*ns*) *ignore any interrupts shorter than this period.*
2. VARIABLE 610 → # of Seconds (10 Minutes and plus 10 seconds adding 1 second extra for every minute).

When the program completes execution, it generates a CSV file as output into a directory, with current date and time created through the script.

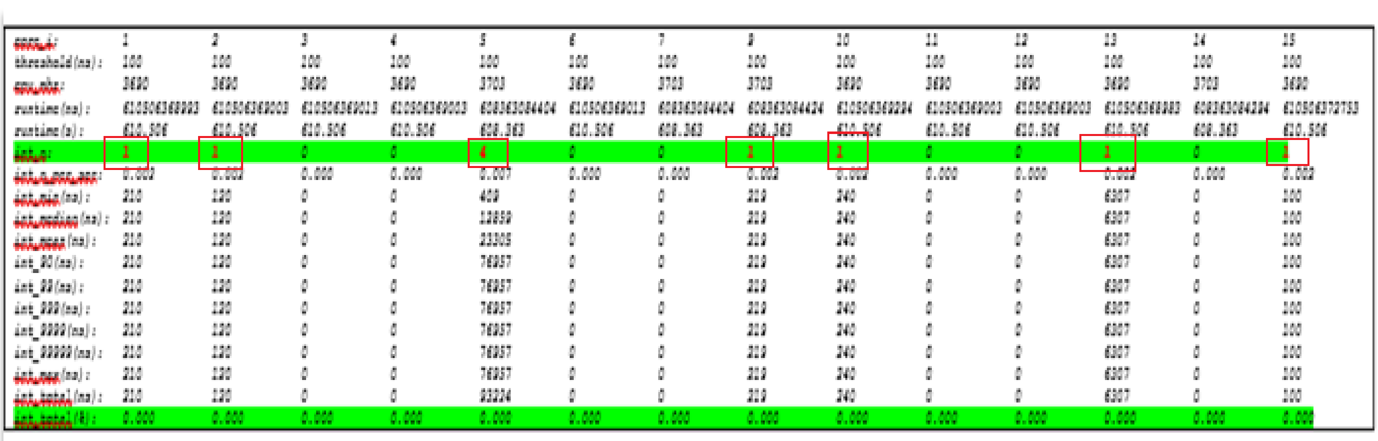
5.4 Jitter Analysis from Sysjitter Output

5.4.1 Individual Core Statistics by Example on AMD EPYC 7F32 (2 Sockets)

The following is a sample output of the individual core statistics files and the formatted Tab file:

```
-rw-r--r-- 1 root root   386 Jun 22 17:33 sysjitter.hpe-lowlat.20200622172324PDT.15
-rw-r--r-- 1 root root    56 Jun 22 17:33 sysjitter.hpe-lowlat.20200622172324PDT.14
-rw-r--r-- 1 root root   388 Jun 22 17:33 sysjitter.hpe-lowlat.20200622172324PDT.13
-rw-r--r-- 1 root root    56 Jun 22 17:33 sysjitter.hpe-lowlat.20200622172324PDT.12
-rw-r--r-- 1 root root    56 Jun 22 17:33 sysjitter.hpe-lowlat.20200622172324PDT.11
-rw-r--r-- 1 root root   386 Jun 22 17:33 sysjitter.hpe-lowlat.20200622172324PDT.10
-rw-r--r-- 1 root root   386 Jun 22 17:33 sysjitter.hpe-lowlat.20200622172324PDT.09
-rw-r--r-- 1 root root    56 Jun 22 17:33 sysjitter.hpe-lowlat.20200622172324PDT.07
-rw-r--r-- 1 root root    56 Jun 22 17:33 sysjitter.hpe-lowlat.20200622172324PDT.06
-rw-r--r-- 1 root root   539 Jun 22 17:33 sysjitter.hpe-lowlat.20200622172324PDT.05
-rw-r--r-- 1 root root    56 Jun 22 17:33 sysjitter.hpe-lowlat.20200622172324PDT.04
-rw-r--r-- 1 root root    56 Jun 22 17:33 sysjitter.hpe-lowlat.20200622172324PDT.03
-rw-r--r-- 1 root root   386 Jun 22 17:33 sysjitter.hpe-lowlat.20200622172324PDT.02
-rw-r--r-- 1 root root   386 Jun 22 17:33 sysjitter.hpe-lowlat.20200622172324PDT.01
-rw-r--r-- 1 root root  3669 Jun 22 17:33 sysjitter.hpe-lowlat.20200622172324PDT.tab
```

The following is an example of formatted output for easier readability. Here the output tab file **`sysjitter.hpe-lowlat.20200622172324PDT.tab`** was formatted by the Linux command (*`column -t`*).



core_i:	1	2	3	4	5	6	7	9	10	11	12	13	14	15
threshold(ns):	100	100	100	100	100	100	100	100	100	100	100	100	100	100
cpu_mhz:	3690	3690	3690	3690	3703	3690	3703	3703	3690	3690	3690	3690	3703	3690
runtime(ns):	610506368993	610506369003	610506369013	610506369003	608363084404	610506369013	608363084404	608363084424	610506369394	610506369003	610506369003	610506368983	608363084394	610506372753
runtime(s):	610.506	610.506	610.506	610.506	608.363	610.506	608.363	608.363	610.506	610.506	610.506	610.506	608.363	610.506
int_n:	1	1	0	0	4	0	0	1	1	0	0	1	0	1
int_n_per_sec:	0.002	0.002	0.000	0.000	0.007	0.000	0.000	0.002	0.002	0.000	0.000	0.002	0.000	0.002
int_min(ns):	210	120	0	0	409	0	0	219	240	0	0	6307	0	100
int_median(ns):	210	120	0	0	12859	0	0	219	240	0	0	6307	0	100
int_mean(ns):	210	120	0	0	23305	0	0	219	240	0	0	6307	0	100
int_90(ns):	210	120	0	0	76957	0	0	219	240	0	0	6307	0	100
int_99(ns):	210	120	0	0	76957	0	0	219	240	0	0	6307	0	100
int_999(ns):	210	120	0	0	76957	0	0	219	240	0	0	6307	0	100
int_9999(ns):	210	120	0	0	76957	0	0	219	240	0	0	6307	0	100
int_99999(ns):	210	120	0	0	76957	0	0	219	240	0	0	6307	0	100
int_max(ns):	210	120	0	0	76957	0	0	219	240	0	0	6307	0	100
int_total(ns):	210	120	0	0	93224	0	0	219	240	0	0	6307	0	100
int_total(%):	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Figure 2 Sysjitter results

5.5 Quick analysis

From the above formatted tab file sysjitter.hpe-lowlat.20200622172324PDT.tab output shows various statistics related to Interrupts observed by sysjitter on the 14 isolated cores on the 2 AMD EPYC 7F32 processors (1,2,3,4,5,6,7,9,10,11,12,13,14,15) during the 610 seconds run. In this example there are a total of 10 Interrupts encountered, marked in red rectangles in Figure 2, during the 10 minutes run. It is strongly recommended that you also peruse the sysjitter per-core output files, which show the time-history of jitter events during the sysjitter execution time window.

These results are very good, showing that the jitter is reduced by the BIOS, RHEL OS and TUNED settings/tunings, which demonstrate this HPE ProLiant DL385 Gen10 Plus server with two AMD EPYC 7F32 processors to be a well-tuned server with negligible jitter. As such, this server is an excellent choice for low latency workloads in the High Frequency Trading (HFT) submarket of the Financial Services industry, which is exceptionally demanding on computer hardware. Great care must be taken in architecting these solutions to ensure that deployments perform as required. Hewlett Packard Enterprise, working closely with industry-leading partners, has a long history of configuring and marketing servers to enable many of the world's leading financial institutions to meet this challenge.

5.6 Deeper analysis

Should you decide to go for a deeper analysis of this Low Latency Performance Tuning, contact your AMD or HPE representative for further assistance.