



Red Hat Enterprise Linux 9.0

Monitoring and managing system status and performance

Optimizing system throughput, latency, and power consumption

Red Hat Enterprise Linux 9.0 Monitoring and managing system status and performance

Optimizing system throughput, latency, and power consumption

Legal Notice

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Monitor and optimize the throughput, latency, and power consumption of Red Hat Enterprise Linux 9 in different scenarios.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE	11
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	12
CHAPTER 1. GETTING STARTED WITH TUNED	13
1.1. THE PURPOSE OF TUNED	13
1.2. TUNED PROFILES	13
Syntax of profile configuration	13
1.3. THE DEFAULT TUNED PROFILE	14
1.4. MERGED TUNED PROFILES	14
1.5. THE LOCATION OF TUNED PROFILES	15
1.6. TUNED PROFILES DISTRIBUTED WITH RHEL	15
1.7. TUNED CPU-PARTITIONING PROFILE	17
1.8. USING THE TUNED CPU-PARTITIONING PROFILE FOR LOW-LATENCY TUNING	18
1.9. CUSTOMIZING THE CPU-PARTITIONING TUNED PROFILE	19
1.10. REAL-TIME TUNED PROFILES DISTRIBUTED WITH RHEL	20
1.11. STATIC AND DYNAMIC TUNING IN TUNED	20
1.12. TUNED NO-DAEMON MODE	21
1.13. INSTALLING AND ENABLING TUNED	21
1.14. LISTING AVAILABLE TUNED PROFILES	22
1.15. SETTING A TUNED PROFILE	23
1.16. DISABLING TUNED	24
CHAPTER 2. CUSTOMIZING TUNED PROFILES	25
2.1. TUNED PROFILES	25
Syntax of profile configuration	25
2.2. THE DEFAULT TUNED PROFILE	25
2.3. MERGED TUNED PROFILES	26
2.4. THE LOCATION OF TUNED PROFILES	26
2.5. INHERITANCE BETWEEN TUNED PROFILES	27
2.6. STATIC AND DYNAMIC TUNING IN TUNED	27
2.7. TUNED PLUG-INS	28
Syntax for plug-ins in TuneD profiles	29
Short plug-in syntax	29
Conflicting plug-in definitions in a profile	30
2.8. AVAILABLE TUNED PLUG-INS	30
Monitoring plug-ins	30
Tuning plug-ins	30
2.9. FUNCTIONALITIES OF THE SCHEDULER TUNED PLUGIN	35
2.10. VARIABLES IN TUNED PROFILES	40
2.11. BUILT-IN FUNCTIONS IN TUNED PROFILES	40
2.12. BUILT-IN FUNCTIONS AVAILABLE IN TUNED PROFILES	41
2.13. CREATING NEW TUNED PROFILES	42
2.14. MODIFYING EXISTING TUNED PROFILES	43
2.15. SETTING THE DISK SCHEDULER USING TUNED	44
CHAPTER 3. REVIEWING A SYSTEM USING TUNA INTERFACE	47
3.1. INSTALLING TUNA TOOL	47
3.2. VIEWING THE SYSTEM STATUS USING TUNA TOOL	47
3.3. TUNING CPUS USING TUNA TOOL	48
3.4. TUNING IRQS USING TUNA TOOL	50

CHAPTER 4. MONITORING PERFORMANCE USING RHEL SYSTEM ROLES	52
4.1. PREPARING A CONTROL NODE AND MANAGED NODES TO USE RHEL SYSTEM ROLES	52
4.1.1. Preparing a control node on RHEL 9	52
4.1.2. Preparing a managed node	54
4.2. INTRODUCTION TO THE METRICS SYSTEM ROLE	56
4.3. USING THE METRICS SYSTEM ROLE TO MONITOR YOUR LOCAL SYSTEM WITH VISUALIZATION	57
4.4. USING THE METRICS SYSTEM ROLE TO SET UP A FLEET OF INDIVIDUAL SYSTEMS TO MONITOR THEMSELVES	58
4.5. USING THE METRICS SYSTEM ROLE TO MONITOR A FLEET OF MACHINES CENTRALLY VIA YOUR LOCAL MACHINE	59
4.6. SETTING UP AUTHENTICATION WHILE MONITORING A SYSTEM USING THE METRICS SYSTEM ROLE	60
4.7. USING THE METRICS SYSTEM ROLE TO CONFIGURE AND ENABLE METRICS COLLECTION FOR SQL SERVER	61
CHAPTER 5. SETTING UP PCP	63
5.1. OVERVIEW OF PCP	63
5.2. INSTALLING AND ENABLING PCP	63
5.3. DEPLOYING A MINIMAL PCP SETUP	64
5.4. SYSTEM SERVICES DISTRIBUTED WITH PCP	65
5.5. TOOLS DISTRIBUTED WITH PCP	66
5.6. PCP DEPLOYMENT ARCHITECTURES	68
5.7. RECOMMENDED DEPLOYMENT ARCHITECTURE	72
5.8. SIZING FACTORS	72
5.9. CONFIGURATION OPTIONS FOR PCP SCALING	73
5.10. EXAMPLE: ANALYZING THE CENTRALIZED LOGGING DEPLOYMENT	73
5.11. EXAMPLE: ANALYZING THE FEDERATED SETUP DEPLOYMENT	74
5.12. ESTABLISHING SECURE PCP CONNECTIONS	75
5.12.1. Secure PCP connections	75
5.12.2. Configuring secure connections for PCP collector components	75
5.12.3. Configuring secure connections for PCP monitoring components	76
5.13. TROUBLESHOOTING HIGH MEMORY USAGE	77
CHAPTER 6. LOGGING PERFORMANCE DATA WITH PMLOGGER	80
6.1. MODIFYING THE PMLOGGER CONFIGURATION FILE WITH PMLOGCONF	80
6.2. EDITING THE PMLOGGER CONFIGURATION FILE MANUALLY	80
6.3. ENABLING THE PMLOGGER SERVICE	81
6.4. SETTING UP A CLIENT SYSTEM FOR METRICS COLLECTION	82
6.5. SETTING UP A CENTRAL SERVER TO COLLECT DATA	83
6.6. REPLAYING THE PCP LOG ARCHIVES WITH PMREP	84
6.7. ENABLING PCP VERSION 3 ARCHIVES	85
CHAPTER 7. MONITORING PERFORMANCE WITH PERFORMANCE CO-PILOT	87
7.1. MONITORING POSTFIX WITH PMDA-POSTFIX	87
7.2. VISUALLY TRACING PCP LOG ARCHIVES WITH THE PCP CHARTS APPLICATION	88
7.3. COLLECTING DATA FROM SQL SERVER USING PCP	90
7.4. GENERATING PCP ARCHIVES FROM SADC ARCHIVES	92
CHAPTER 8. PERFORMANCE ANALYSIS OF XFS WITH PCP	93
8.1. INSTALLING XFS PMDA MANUALLY	93
8.2. EXAMINING XFS PERFORMANCE METRICS WITH PMINFO	94
8.3. RESETTING XFS PERFORMANCE METRICS WITH PMSTORE	95
8.4. PCP METRIC GROUPS FOR XFS	96
8.5. PER-DEVICE PCP METRIC GROUPS FOR XFS	97

CHAPTER 9. SETTING UP GRAPHICAL REPRESENTATION OF PCP METRICS	99
9.1. SETTING UP PCP WITH PCP-ZEROCONF	99
9.2. SETTING UP A GRAFANA-SERVER	99
9.3. ACCESSING THE GRAFANA WEB UI	100
9.4. CONFIGURING SECURE CONNECTIONS FOR GRAFANA	102
9.5. CONFIGURING PCP REDIS	103
9.6. CONFIGURING SECURE CONNECTIONS FOR PCP REDIS	104
9.7. CREATING PANELS AND ALERT IN PCP REDIS DATA SOURCE	105
9.8. ADDING NOTIFICATION CHANNELS FOR ALERTS	107
9.9. SETTING UP AUTHENTICATION BETWEEN PCP COMPONENTS	108
9.10. INSTALLING PCP BPFTRACE	109
9.11. VIEWING THE PCP BPFTRACE SYSTEM ANALYSIS DASHBOARD	110
9.12. INSTALLING PCP VECTOR	112
9.13. VIEWING THE PCP VECTOR CHECKLIST	112
9.14. TROUBLESHOOTING GRAFANA ISSUES	114
CHAPTER 10. OPTIMIZING THE SYSTEM PERFORMANCE USING THE WEB CONSOLE	116
10.1. PERFORMANCE TUNING OPTIONS IN THE WEB CONSOLE	116
10.2. SETTING A PERFORMANCE PROFILE IN THE WEB CONSOLE	116
10.3. MONITORING PERFORMANCE ON THE LOCAL SYSTEM USING THE WEB CONSOLE	117
10.4. MONITORING PERFORMANCE ON SEVERAL SYSTEMS USING THE WEB CONSOLE AND GRAFANA	118
CHAPTER 11. SETTING THE DISK SCHEDULER	121
11.1. AVAILABLE DISK SCHEDULERS	121
11.2. DIFFERENT DISK SCHEDULERS FOR DIFFERENT USE CASES	122
11.3. THE DEFAULT DISK SCHEDULER	122
11.4. DETERMINING THE ACTIVE DISK SCHEDULER	122
11.5. SETTING THE DISK SCHEDULER USING TUNED	123
11.6. SETTING THE DISK SCHEDULER USING UDEV RULES	125
11.7. TEMPORARILY SETTING A SCHEDULER FOR A SPECIFIC DISK	126
CHAPTER 12. TUNING THE PERFORMANCE OF A SAMBA SERVER	127
12.1. SETTING THE SMB PROTOCOL VERSION	127
12.2. TUNING SHARES WITH DIRECTORIES THAT CONTAIN A LARGE NUMBER OF FILES	127
12.3. SETTINGS THAT CAN HAVE A NEGATIVE PERFORMANCE IMPACT	128
CHAPTER 13. OPTIMIZING VIRTUAL MACHINE PERFORMANCE	129
13.1. WHAT INFLUENCES VIRTUAL MACHINE PERFORMANCE	129
The impact of virtualization on system performance	129
Reducing VM performance loss	129
13.2. OPTIMIZING VIRTUAL MACHINE PERFORMANCE USING TUNED	130
13.3. OPTIMIZING LIBVIRT DAEMONS	131
13.3.1. Types of libvirt daemons	131
13.3.2. Enabling modular libvirt daemons	132
13.4. CONFIGURING VIRTUAL MACHINE MEMORY	133
13.4.1. Adding and removing virtual machine memory using the web console	133
13.4.2. Adding and removing virtual machine memory using the command-line interface	135
13.4.3. Adding and removing virtual machine memory by using virtio-mem	137
13.4.3.1. Prerequisites	137
13.4.3.2. Overview of virtio-mem	137
13.4.3.3. Configuring memory onlining in virtual machines	139
13.4.3.4. Attaching a virtio-mem device to virtual machines	141
13.4.4. Additional resources	145

13.5. OPTIMIZING VIRTUAL MACHINE I/O PERFORMANCE	145
13.5.1. Tuning block I/O in virtual machines	145
13.5.2. Disk I/O throttling in virtual machines	146
13.5.3. Enabling multi-queue virtio-scsi	147
13.6. OPTIMIZING VIRTUAL MACHINE CPU PERFORMANCE	147
13.6.1. Adding and removing virtual CPUs using the command-line interface	148
13.6.2. Managing virtual CPUs using the web console	149
13.6.3. Configuring NUMA in a virtual machine	150
13.6.4. Sample vCPU performance tuning scenario	152
13.6.5. Managing kernel same-page merging	157
13.7. OPTIMIZING VIRTUAL MACHINE NETWORK PERFORMANCE	159
13.8. VIRTUAL MACHINE PERFORMANCE MONITORING TOOLS	160
13.9. ADDITIONAL RESOURCES	162
CHAPTER 14. IMPORTANCE OF POWER MANAGEMENT	163
14.1. POWER MANAGEMENT BASICS	163
14.2. AUDIT AND ANALYSIS OVERVIEW	164
14.3. TOOLS FOR AUDITING	165
CHAPTER 15. MANAGING POWER CONSUMPTION WITH POWERTOP	169
15.1. THE PURPOSE OF POWERTOP	169
15.2. USING POWERTOP	169
15.2.1. Starting PowerTOP	169
15.2.2. Calibrating PowerTOP	169
15.2.3. Setting the measuring interval	170
15.2.4. Additional resources	170
15.3. POWERTOP STATISTICS	170
15.3.1. The Overview tab	170
15.3.2. The Idle stats tab	171
15.3.3. The Device stats tab	171
15.3.4. The Tunables tab	171
15.3.5. The WakeUp tab	171
15.4. WHY POWERTOP DOES NOT DISPLAY FREQUENCY STATS VALUES IN SOME INSTANCES	172
15.5. GENERATING AN HTML OUTPUT	173
15.6. OPTIMIZING POWER CONSUMPTION	173
15.6.1. Optimizing power consumption using the powertop service	173
15.6.2. The powertop2tuned utility	173
15.6.3. Optimizing power consumption using the powertop2tuned utility	173
15.6.4. Comparison of powertop.service and powertop2tuned	174
CHAPTER 16. GETTING STARTED WITH PERF	175
16.1. INTRODUCTION TO PERF	175
16.2. INSTALLING PERF	175
16.3. COMMON PERF COMMANDS	175
CHAPTER 17. PROFILING CPU USAGE IN REAL TIME WITH PERF TOP	177
17.1. THE PURPOSE OF PERF TOP	177
17.2. PROFILING CPU USAGE WITH PERF TOP	177
17.3. INTERPRETATION OF PERF TOP OUTPUT	178
17.4. WHY PERF DISPLAYS SOME FUNCTION NAMES AS RAW FUNCTION ADDRESSES	178
17.5. ENABLING DEBUG AND SOURCE REPOSITORIES	178
17.6. GETTING DEBUGINFO PACKAGES FOR AN APPLICATION OR LIBRARY USING GDB	179
CHAPTER 18. COUNTING EVENTS DURING PROCESS EXECUTION WITH PERF STAT	181

18.1. THE PURPOSE OF PERF STAT	181
18.2. COUNTING EVENTS WITH PERF STAT	181
18.3. INTERPRETATION OF PERF STAT OUTPUT	182
18.4. ATTACHING PERF STAT TO A RUNNING PROCESS	183
CHAPTER 19. RECORDING AND ANALYZING PERFORMANCE PROFILES WITH PERF	184
19.1. THE PURPOSE OF PERF RECORD	184
19.2. RECORDING A PERFORMANCE PROFILE WITHOUT ROOT ACCESS	184
19.3. RECORDING A PERFORMANCE PROFILE WITH ROOT ACCESS	184
19.4. RECORDING A PERFORMANCE PROFILE IN PER-CPU MODE	185
19.5. CAPTURING CALL GRAPH DATA WITH PERF RECORD	185
19.6. ANALYZING PERF.DATA WITH PERF REPORT	186
19.7. INTERPRETATION OF PERF REPORT OUTPUT	187
19.8. GENERATING A PERF.DATA FILE THAT IS READABLE ON A DIFFERENT DEVICE	187
19.9. ANALYZING A PERF.DATA FILE THAT WAS CREATED ON A DIFFERENT DEVICE	188
19.10. WHY PERF DISPLAYS SOME FUNCTION NAMES AS RAW FUNCTION ADDRESSES	189
19.11. ENABLING DEBUG AND SOURCE REPOSITORIES	189
19.12. GETTING DEBUGINFO PACKAGES FOR AN APPLICATION OR LIBRARY USING GDB	190
CHAPTER 20. INVESTIGATING BUSY CPUS WITH PERF	191
20.1. DISPLAYING WHICH CPU EVENTS WERE COUNTED ON WITH PERF STAT	191
20.2. DISPLAYING WHICH CPU SAMPLES WERE TAKEN ON WITH PERF REPORT	191
20.3. DISPLAYING SPECIFIC CPUS DURING PROFILING WITH PERF TOP	192
20.4. MONITORING SPECIFIC CPUS WITH PERF RECORD AND PERF REPORT	192
CHAPTER 21. MONITORING APPLICATION PERFORMANCE WITH PERF	194
21.1. ATTACHING PERF RECORD TO A RUNNING PROCESS	194
21.2. CAPTURING CALL GRAPH DATA WITH PERF RECORD	194
21.3. ANALYZING PERF.DATA WITH PERF REPORT	195
CHAPTER 22. CREATING UPROBES WITH PERF	197
22.1. CREATING UPROBES AT THE FUNCTION LEVEL WITH PERF	197
22.2. CREATING UPROBES ON LINES WITHIN A FUNCTION WITH PERF	197
22.3. PERF SCRIPT OUTPUT OF DATA RECORDED OVER UPROBES	198
CHAPTER 23. PROFILING MEMORY ACCESSES WITH PERF MEM	199
23.1. THE PURPOSE OF PERF MEM	199
23.2. SAMPLING MEMORY ACCESS WITH PERF MEM	199
23.3. INTERPRETATION OF PERF MEM REPORT OUTPUT	201
CHAPTER 24. DETECTING FALSE SHARING	203
24.1. THE PURPOSE OF PERF C2C	203
24.2. DETECTING CACHE-LINE CONTENTION WITH PERF C2C	203
24.3. VISUALIZING A PERF.DATA FILE RECORDED WITH PERF C2C RECORD	204
24.4. INTERPRETATION OF PERF C2C REPORT OUTPUT	206
24.5. DETECTING FALSE SHARING WITH PERF C2C	207
CHAPTER 25. GETTING STARTED WITH FLAMEGRAPHS	210
25.1. INSTALLING FLAMEGRAPHS	210
25.2. CREATING FLAMEGRAPHS OVER THE ENTIRE SYSTEM	210
25.3. CREATING FLAMEGRAPHS OVER SPECIFIC PROCESSES	211
25.4. INTERPRETING FLAMEGRAPHS	212
CHAPTER 26. MONITORING PROCESSES FOR PERFORMANCE BOTTLENECKS USING PERF CIRCULAR BUFFERS	214

26.1. CIRCULAR BUFFERS AND EVENT-SPECIFIC SNAPSHOTS WITH PERF	214
26.2. COLLECTING SPECIFIC DATA TO MONITOR FOR PERFORMANCE BOTTLENECKS USING PERF CIRCULAR BUFFERS	214
CHAPTER 27. ADDING AND REMOVING TRACEPOINTS FROM A RUNNING PERF COLLECTOR WITHOUT STOPPING OR RESTARTING PERF	216
27.1. ADDING TRACEPOINTS TO A RUNNING PERF COLLECTOR WITHOUT STOPPING OR RESTARTING PERF	216
27.2. REMOVING TRACEPOINTS FROM A RUNNING PERF COLLECTOR WITHOUT STOPPING OR RESTARTING PERF	217
CHAPTER 28. PROFILING MEMORY ALLOCATION WITH NUMASTAT	218
28.1. DEFAULT NUMASTAT STATISTICS	218
28.2. VIEWING MEMORY ALLOCATION WITH NUMASTAT	218
CHAPTER 29. CONFIGURING AN OPERATING SYSTEM TO OPTIMIZE CPU UTILIZATION	220
29.1. TOOLS FOR MONITORING AND DIAGNOSING PROCESSOR ISSUES	220
29.2. TYPES OF SYSTEM TOPOLOGY	221
29.2.1. Displaying system topologies	221
29.3. CONFIGURING KERNEL TICK TIME	223
29.4. OVERVIEW OF AN INTERRUPT REQUEST	225
29.4.1. Balancing interrupts manually	225
29.4.2. Setting the smp_affinity mask	226
CHAPTER 30. TUNING SCHEDULING POLICY	228
30.1. CATEGORIES OF SCHEDULING POLICIES	228
30.2. STATIC PRIORITY SCHEDULING WITH SCHED_FIFO	228
30.3. ROUND ROBIN PRIORITY SCHEDULING WITH SCHED_RR	229
30.4. NORMAL SCHEDULING WITH SCHED_OTHER	229
30.5. SETTING SCHEDULER POLICIES	229
30.6. POLICY OPTIONS FOR THE CHRT COMMAND	230
30.7. CHANGING THE PRIORITY OF SERVICES DURING THE BOOT PROCESS	231
30.8. PRIORITY MAP	232
30.9. TUNED CPU-PARTITIONING PROFILE	233
30.10. USING THE TUNED CPU-PARTITIONING PROFILE FOR LOW-LATENCY TUNING	234
30.11. CUSTOMIZING THE CPU-PARTITIONING TUNED PROFILE	235
CHAPTER 31. TUNING THE NETWORK PERFORMANCE	237
31.1. TUNING NETWORK ADAPTER SETTINGS	237
31.1.1. Increasing the ring buffers to reduce a high packet drop rate	237
31.1.2. Tuning the network device backlog queue to avoid packet drops	238
31.1.3. Increasing the transmit queue length of a NIC to reduce the number of transmit errors	240
31.2. TUNING IRQ BALANCING	241
31.2.1. Interrupts and interrupt handlers	241
31.2.2. Software interrupt requests	241
31.2.3. NAPI Polling	242
31.2.4. The irqbalance service	242
31.2.5. Increasing the time SoftIRQs can run on the CPU	243
31.3. IMPROVING THE NETWORK LATENCY	244
31.3.1. How the CPU power states influence the network latency	244
31.3.2. C-state settings in the EFI firmware	245
31.3.3. Disabling C-states by using a custom Tuned profile	245
31.3.4. Disabling C-states by using a kernel command line option	246
31.4. IMPROVING THE THROUGHPUT OF LARGE AMOUNTS OF CONTIGUOUS DATA STREAMS	248
31.4.1. Considerations before configuring jumbo frames	248

31.4.2. Configuring the MTU in an existing NetworkManager connection profile	249
31.5. TUNING TCP CONNECTIONS FOR HIGH THROUGHPUT	250
31.5.1. Testing the TCP throughput using iperf3	250
31.5.2. The system-wide TCP socket buffer settings	252
31.5.3. Increasing the system-wide TCP socket buffers	252
31.5.4. TCP Window Scaling	254
31.5.5. How TCP SACK reduces the packet drop rate	255
31.6. TUNING UDP CONNECTIONS	255
31.6.1. Detecting packet drops	256
31.6.2. Testing the UDP throughput using iperf3	257
31.6.3. Impact of the MTU size on UDP traffic throughput	259
31.6.4. Impact of the CPU speed on UDP traffic throughput	259
31.6.5. Increasing the system-wide UDP socket buffers	260
31.7. IDENTIFYING APPLICATION READ SOCKET BUFFER BOTTLENECKS	261
31.7.1. Identifying receive buffer collapsing and pruning	261
31.8. TUNING APPLICATIONS WITH A LARGE NUMBER OF INCOMING REQUESTS	262
31.8.1. Tuning the TCP listen backlog to process a high number of TCP connection attempts	262
31.9. AVOIDING LISTEN QUEUE LOCK CONTENTION	264
31.9.1. Avoiding RX queue lock contention: The SO_REUSEPORT and SO_REUSEPORT_BPF socket options	264
31.9.2. Avoiding TX queue lock contention: Transmit packet steering	265
31.9.3. Disabling the Generic Receive Offload feature on servers with high UDP traffic	267
31.10. TUNING THE DEVICE DRIVER AND NIC	268
31.10.1. Configuring custom NIC driver parameters	269
31.11. CONFIGURING NETWORK ADAPTER OFFLOAD SETTINGS	270
31.11.1. Temporarily setting an offload feature	271
31.11.2. Permanently setting an offload feature	272
31.12. TUNING INTERRUPT COALESCENCE SETTINGS	273
31.12.1. Optimizing RHEL for latency or throughput-sensitive services	273
31.13. BENEFITS OF TCP TIMESTAMPS	277
31.14. FLOW CONTROL FOR ETHERNET NETWORKS	277
CHAPTER 32. FACTORS AFFECTING I/O AND FILE SYSTEM PERFORMANCE	279
32.1. TOOLS FOR MONITORING AND DIAGNOSING I/O AND FILE SYSTEM ISSUES	279
32.2. AVAILABLE TUNING OPTIONS FOR FORMATTING A FILE SYSTEM	281
32.3. AVAILABLE TUNING OPTIONS FOR MOUNTING A FILE SYSTEM	282
32.4. TYPES OF DISCARDING UNUSED BLOCKS	283
32.5. SOLID-STATE DISKS TUNING CONSIDERATIONS	283
32.6. GENERIC BLOCK DEVICE TUNING PARAMETERS	284
CHAPTER 33. USING SYSTEMD TO MANAGE RESOURCES USED BY APPLICATIONS	286
33.1. ALLOCATING SYSTEM RESOURCES USING SYSTEMD	286
33.2. ROLE OF SYSTEMD IN RESOURCE MANAGEMENT	287
33.3. OVERVIEW OF SYSTEMD HIERARCHY FOR CGROUPS	287
33.4. LISTING SYSTEMD UNITS	289
33.5. VIEWING SYSTEMD CONTROL GROUP HIERARCHY	290
33.6. VIEWING CGROUPS OF PROCESSES	292
33.7. MONITORING RESOURCE CONSUMPTION	293
33.8. USING SYSTEMD UNIT FILES TO SET LIMITS FOR APPLICATIONS	293
33.9. USING SYSTEMCTL COMMAND TO SET LIMITS TO APPLICATIONS	294
33.10. SETTING GLOBAL DEFAULT CPU AFFINITY THROUGH MANAGER CONFIGURATION	295
33.11. CONFIGURING NUMA POLICIES USING SYSTEMD	296
33.12. NUMA POLICY CONFIGURATION OPTIONS FOR SYSTEMD	297

33.13. CREATING TRANSIENT CGROUPS USING SYSTEMD-RUN COMMAND	297
33.14. REMOVING TRANSIENT CONTROL GROUPS	298
CHAPTER 34. UNDERSTANDING CONTROL GROUPS	300
34.1. INTRODUCING CONTROL GROUPS	300
34.2. INTRODUCING KERNEL RESOURCE CONTROLLERS	301
34.3. INTRODUCING NAMESPACES	303
CHAPTER 35. USING CGROUPFS TO MANUALLY MANAGE CGROUPS	304
35.1. CREATING CGROUPS AND ENABLING CONTROLLERS IN CGROUPS-V2 FILE SYSTEM	304
35.2. CONTROLLING DISTRIBUTION OF CPU TIME FOR APPLICATIONS BY ADJUSTING CPU WEIGHT	306
35.3. MOUNTING CGROUPS-V1	309
35.4. SETTING CPU LIMITS TO APPLICATIONS USING CGROUPS-V1	311
CHAPTER 36. ANALYZING SYSTEM PERFORMANCE WITH BPF COMPILER COLLECTION	315
36.1. INSTALLING THE BCC-TOOLS PACKAGE	315
36.2. USING SELECTED BCC-TOOLS FOR PERFORMANCE ANALYSES	315
Using execsnoop to examine the system processes	315
Using opensnoop to track what files a command opens	316
Using btop to examine the I/O operations on the disk	317
Using xfs slower to expose unexpectedly slow file system operations	318
CHAPTER 37. CONFIGURING AN OPERATING SYSTEM TO OPTIMIZE MEMORY ACCESS	320
37.1. TOOLS FOR MONITORING AND DIAGNOSING SYSTEM MEMORY ISSUES	320
37.2. OVERVIEW OF A SYSTEM'S MEMORY	320
37.3. VIRTUAL MEMORY PARAMETERS	321
37.4. FILE SYSTEM PARAMETERS	324
37.5. KERNEL PARAMETERS	324
37.6. SETTING MEMORY-RELATED KERNEL PARAMETERS	325
CHAPTER 38. CONFIGURING HUGE PAGES	326
38.1. AVAILABLE HUGE PAGE FEATURES	326
38.2. PARAMETERS FOR RESERVING HUGETLB PAGES AT BOOT TIME	327
38.3. CONFIGURING HUGETLB AT BOOT TIME	327
38.4. PARAMETERS FOR RESERVING HUGETLB PAGES AT RUN TIME	329
38.5. CONFIGURING HUGETLB AT RUN TIME	329
38.6. ENABLING TRANSPARENT HUGE PAGES	330
38.7. DISABLING TRANSPARENT HUGE PAGES	331
38.8. IMPACT OF PAGE SIZE ON TRANSLATION LOOKASIDE BUFFER SIZE	331
CHAPTER 39. GETTING STARTED WITH SYSTEMTAP	333
39.1. THE PURPOSE OF SYSTEMTAP	333
39.2. INSTALLING SYSTEMTAP	333
39.3. PRIVILEGES TO RUN SYSTEMTAP	334
39.4. RUNNING SYSTEMTAP SCRIPTS	335
CHAPTER 40. CROSS-INSTRUMENTATION OF SYSTEMTAP	336
40.1. SYSTEMTAP CROSS-INSTRUMENTATION	336
40.2. INITIALIZING CROSS-INSTRUMENTATION OF SYSTEMTAP	337
CHAPTER 41. MONITORING NETWORK ACTIVITY WITH SYSTEMTAP	339
41.1. PROFILING NETWORK ACTIVITY WITH SYSTEMTAP	339
41.2. TRACING FUNCTIONS CALLED IN NETWORK SOCKET CODE WITH SYSTEMTAP	340
41.3. MONITORING NETWORK PACKET DROPS WITH SYSTEMTAP	341
CHAPTER 42. PROFILING KERNEL ACTIVITY WITH SYSTEMTAP	342

42.1. COUNTING FUNCTION CALLS WITH SYSTEMTAP	342
42.2. TRACING FUNCTION CALLS WITH SYSTEMTAP	343
42.3. DETERMINING TIME SPENT IN KERNEL AND USER SPACE WITH SYSTEMTAP	344
42.4. MONITORING POLLING APPLICATIONS WITH SYSTEMTAP	345
42.5. TRACKING MOST FREQUENTLY USED SYSTEM CALLS WITH SYSTEMTAP	346
42.6. TRACKING SYSTEM CALL VOLUME PER PROCESS WITH SYSTEMTAP	346
CHAPTER 43. MONITORING DISK AND I/O ACTIVITY WITH SYSTEMTAP	348
43.1. SUMMARIZING DISK READ/WRITE TRAFFIC WITH SYSTEMTAP	348
43.2. TRACKING I/O TIME FOR EACH FILE READ OR WRITE WITH SYSTEMTAP	349
43.3. TRACKING CUMULATIVE I/O WITH SYSTEMTAP	349
43.4. MONITORING I/O ACTIVITY ON A SPECIFIC DEVICE WITH SYSTEMTAP	350
43.5. MONITORING READS AND WRITES TO A FILE WITH SYSTEMTAP	351

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting comments on specific passages

1. View the documentation in the **Multi-page HTML** format and ensure that you see the **Feedback** button in the upper right corner after the page fully loads.
2. Use your cursor to highlight the part of the text that you want to comment on.
3. Click the **Add Feedback** button that appears near the highlighted text.
4. Add your feedback and click **Submit**.

Submitting feedback through Jira (account required)

1. Log in to the [Jira](#) website.
2. Click **Create** in the top navigation bar
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Create** at the bottom of the dialogue.

CHAPTER 1. GETTING STARTED WITH TUNED

As a system administrator, you can use the **Tuned** application to optimize the performance profile of your system for a variety of use cases.

1.1. THE PURPOSE OF TUNED

Tuned is a service that monitors your system and optimizes the performance under certain workloads. The core of **Tuned** are *profiles*, which tune your system for different use cases.

Tuned is distributed with a number of predefined profiles for use cases such as:

- High throughput
- Low latency
- Saving power

It is possible to modify the rules defined for each profile and customize how to tune a particular device. When you switch to another profile or deactivate **Tuned**, all changes made to the system settings by the previous profile revert back to their original state.

You can also configure **Tuned** to react to changes in device usage and adjusts settings to improve performance of active devices and reduce power consumption of inactive devices.

1.2. TUNED PROFILES

A detailed analysis of a system can be very time-consuming. **Tuned** provides a number of predefined profiles for typical use cases. You can also create, modify, and delete profiles.

The profiles provided with **Tuned** are divided into the following categories:

- Power-saving profiles
- Performance-boosting profiles

The performance-boosting profiles include profiles that focus on the following aspects:

- Low latency for storage and network
- High throughput for storage and network
- Virtual machine performance
- Virtualization host performance

Syntax of profile configuration

The **tuned.conf** file can contain one **[main]** section and other sections for configuring plug-in instances. However, all sections are optional.

Lines starting with the hash sign (**#**) are comments.

Additional resources

- **tuned.conf(5)** man page.

1.3. THE DEFAULT TUNED PROFILE

During the installation, the best profile for your system is selected automatically. Currently, the default profile is selected according to the following customizable rules:

Environment	Default profile	Goal
Compute nodes	throughput-performance	The best throughput performance
Virtual machines	virtual-guest	The best performance. If you are not interested in the best performance, you can change it to the balanced or powersave profile.
Other cases	balanced	Balanced performance and power consumption

Additional resources

- **tuned.conf(5)** man page.

1.4. MERGED TUNED PROFILES

As an experimental feature, it is possible to select more profiles at once. **Tuned** will try to merge them during the load.

If there are conflicts, the settings from the last specified profile takes precedence.

Example 1.1. Low power consumption in a virtual guest

The following example optimizes the system to run in a virtual machine for the best performance and concurrently tunes it for low power consumption, while the low power consumption is the priority:

```
# tuned-adm profile virtual-guest powersave
```



WARNING

Merging is done automatically without checking whether the resulting combination of parameters makes sense. Consequently, the feature might tune some parameters the opposite way, which might be counterproductive: for example, setting the disk for high throughput by using the **throughput-performance** profile and concurrently setting the disk spindown to the low value by the **spindown-disk** profile.

Additional resources

***tuned-adm** man page. * **tuned.conf(5)** man page.

1.5. THE LOCATION OF TUNED PROFILES

TuneD stores profiles in the following directories:

/usr/lib/tuned/

Distribution-specific profiles are stored in the directory. Each profile has its own directory. The profile consists of the main configuration file called **tuned.conf**, and optionally other files, for example helper scripts.

/etc/tuned/

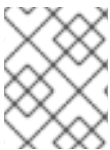
If you need to customize a profile, copy the profile directory into the directory, which is used for custom profiles. If there are two profiles of the same name, the custom profile located in **/etc/tuned/** is used.

Additional resources

- **tuned.conf(5)** man page.

1.6. TUNED PROFILES DISTRIBUTED WITH RHEL

The following is a list of profiles that are installed with **TuneD** on Red Hat Enterprise Linux.



NOTE

There might be more product-specific or third-party **TuneD** profiles available. Such profiles are usually provided by separate RPM packages.

balanced

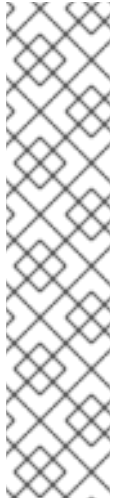
The default power-saving profile. It is intended to be a compromise between performance and power consumption. It uses auto-scaling and auto-tuning whenever possible. The only drawback is the increased latency. In the current **TuneD** release, it enables the CPU, disk, audio, and video plugins, and activates the **conservative** CPU governor. The **radeon_powersave** option uses the **dpm-balanced** value if it is supported, otherwise it is set to **auto**.

It changes the **energy_performance_preference** attribute to the **normal** energy setting. It also changes the **scaling_governor** policy attribute to either the **conservative** or **powersave** CPU governor.

powersave

A profile for maximum power saving performance. It can throttle the performance in order to minimize the actual power consumption. In the current **TuneD** release it enables USB autosuspend, WiFi power saving, and Aggressive Link Power Management (ALPM) power savings for SATA host adapters. It also schedules multi-core power savings for systems with a low wakeup rate and activates the **ondemand** governor. It enables AC97 audio power saving or, depending on your system, HDA-Intel power savings with a 10 seconds timeout. If your system contains a supported Radeon graphics card with enabled KMS, the profile configures it to automatic power saving. On ASUS Eee PCs, a dynamic Super Hybrid Engine is enabled.

It changes the **energy_performance_preference** attribute to the **powersave** or **power** energy setting. It also changes the **scaling_governor** policy attribute to either the **ondemand** or **powersave** CPU governor.



NOTE

In certain cases, the **balanced** profile is more efficient compared to the **powersave** profile.

Consider there is a defined amount of work that needs to be done, for example a video file that needs to be transcoded. Your machine might consume less energy if the transcoding is done on the full power, because the task is finished quickly, the machine starts to idle, and it can automatically step-down to very efficient power save modes. On the other hand, if you transcode the file with a throttled machine, the machine consumes less power during the transcoding, but the process takes longer and the overall consumed energy can be higher.

That is why the **balanced** profile can be generally a better option.

throughput-performance

A server profile optimized for high throughput. It disables power savings mechanisms and enables **sysctl** settings that improve the throughput performance of the disk and network IO. CPU governor is set to **performance**.

It changes the **energy_performance_preference** and **scaling_governor** attribute to the **performance** profile.

accelerator-performance

The **accelerator-performance** profile contains the same tuning as the **throughput-performance** profile. Additionally, it locks the CPU to low C states so that the latency is less than 100us. This improves the performance of certain accelerators, such as GPUs.

latency-performance

A server profile optimized for low latency. It disables power savings mechanisms and enables **sysctl** settings that improve latency. CPU governor is set to **performance** and the CPU is locked to the low C states (by PM QoS).

It changes the **energy_performance_preference** and **scaling_governor** attribute to the **performance** profile.

network-latency

A profile for low latency network tuning. It is based on the **latency-performance** profile. It additionally disables transparent huge pages and NUMA balancing, and tunes several other network-related **sysctl** parameters.

It inherits the **latency-performance** profile which changes the **energy_performance_preference** and **scaling_governor** attribute to the **performance** profile.

hpc-compute

A profile optimized for high-performance computing. It is based on the **latency-performance** profile.

network-throughput

A profile for throughput network tuning. It is based on the **throughput-performance** profile. It additionally increases kernel network buffers.

It inherits either the **latency-performance** or **throughput-performance** profile, and changes the **energy_performance_preference** and **scaling_governor** attribute to the **performance** profile.

virtual-guest

A profile designed for Red Hat Enterprise Linux 9 virtual machines and VMWare guests based on the **throughput-performance** profile that, among other tasks, decreases virtual memory swappiness and increases disk readahead values. It does not disable disk barriers.

It inherits the **throughput-performance** profile and changes the **energy_performance_preference** and **scaling_governor** attribute to the **performance** profile.

virtual-host

A profile designed for virtual hosts based on the **throughput-performance** profile that, among other tasks, decreases virtual memory swappiness, increases disk readahead values, and enables a more aggressive value of dirty pages writeback.

It inherits the **throughput-performance** profile and changes the **energy_performance_preference** and **scaling_governor** attribute to the **performance** profile.

oracle

A profile optimized for Oracle databases loads based on **throughput-performance** profile. It additionally disables transparent huge pages and modifies other performance-related kernel parameters. This profile is provided by the **tuned-profiles-oracle** package.

desktop

A profile optimized for desktops, based on the **balanced** profile. It additionally enables scheduler autogroups for better response of interactive applications.

optimize-serial-console

A profile that tunes down I/O activity to the serial console by reducing the `printk` value. This should make the serial console more responsive. This profile is intended to be used as an overlay on other profiles. For example:

```
# tuned-adm profile throughput-performance optimize-serial-console
```

mssql

A profile provided for Microsoft SQL Server. It is based on the **throughput-performance** profile.

intel-sst

A profile optimized for systems with user-defined Intel Speed Select Technology configurations. This profile is intended to be used as an overlay on other profiles. For example:

```
# tuned-adm profile cpu-partitioning intel-sst
```

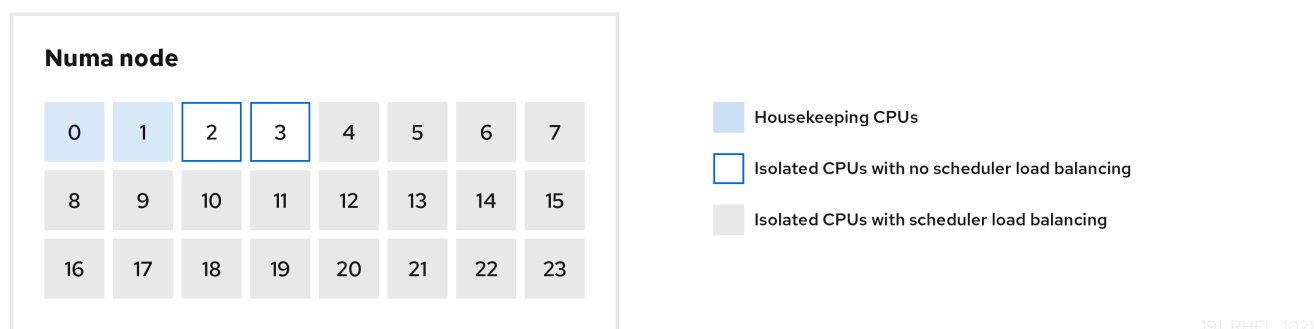
1.7. TUNED CPU-PARTITIONING PROFILE

For tuning Red Hat Enterprise Linux 9 for latency-sensitive workloads, Red Hat recommends to use the **cpu-partitioning** Tuned profile.

Prior to Red Hat Enterprise Linux 9, the low-latency Red Hat documentation described the numerous low-level steps needed to achieve low-latency tuning. In Red Hat Enterprise Linux 9, you can perform low-latency tuning more efficiently by using the **cpu-partitioning** Tuned profile. This profile is easily customizable according to the requirements for individual low-latency applications.

The following figure is an example to demonstrate how to use the **cpu-partitioning** profile. This example uses the CPU and node layout.

Figure 1.1. Figure cpu-partitioning



You can configure the cpu-partitioning profile in the `/etc/tuned/cpu-partitioning-variables.conf` file using the following configuration options:

Isolated CPUs with load balancing

In the cpu-partitioning figure, the blocks numbered from 4 to 23, are the default isolated CPUs. The kernel scheduler's process load balancing is enabled on these CPUs. It is designed for low-latency processes with multiple threads that need the kernel scheduler load balancing.

You can configure the cpu-partitioning profile in the `/etc/tuned/cpu-partitioning-variables.conf` file using the **`isolated_cores=cpu-list`** option, which lists CPUs to isolate that will use the kernel scheduler load balancing.

The list of isolated CPUs is comma-separated or you can specify a range using a dash, such as **`3-5`**. This option is mandatory. Any CPU missing from this list is automatically considered a housekeeping CPU.

Isolated CPUs without load balancing

In the cpu-partitioning figure, the blocks numbered 2 and 3, are the isolated CPUs that do not provide any additional kernel scheduler process load balancing.

You can configure the cpu-partitioning profile in the `/etc/tuned/cpu-partitioning-variables.conf` file using the **`no_balance_cores=cpu-list`** option, which lists CPUs to isolate that will not use the kernel scheduler load balancing.

Specifying the **`no_balance_cores`** option is optional, however any CPUs in this list must be a subset of the CPUs listed in the **`isolated_cores`** list.

Application threads using these CPUs need to be pinned individually to each CPU.

Housekeeping CPUs

Any CPU not isolated in the `cpu-partitioning-variables.conf` file is automatically considered a housekeeping CPU. On the housekeeping CPUs, all services, daemons, user processes, movable kernel threads, interrupt handlers, and kernel timers are permitted to execute.

Additional resources

- **`tuned-profiles-cpu-partitioning(7)`** man page

1.8. USING THE TUNED CPU-PARTITIONING PROFILE FOR LOW-LATENCY TUNING

This procedure describes how to tune a system for low-latency using the Tuned's **cpu-partitioning** profile. It uses the example of a low-latency application that can use **cpu-partitioning** and the CPU layout as mentioned in the [cpu-partitioning](#) figure.

The application in this case uses:

- One dedicated reader thread that reads data from the network will be pinned to CPU 2.
- A large number of threads that process this network data will be pinned to CPUs 4-23.
- A dedicated writer thread that writes the processed data to the network will be pinned to CPU 3.

Prerequisites

- You have installed the **cpu-partitioning** Tuned profile by using the **dnf install tuned-profiles-cpu-partitioning** command as root.

Procedure

1. Edit **/etc/tuned/cpu-partitioning-variables.conf** file and add the following information:

```
# Isolated CPUs with the kernel's scheduler load balancing:
isolated_cores=2-23
# Isolated CPUs without the kernel's scheduler load balancing:
no_balance_cores=2,3
```

2. Set the **cpu-partitioning** Tuned profile:

```
# tuned-adm profile cpu-partitioning
```

3. Reboot

After rebooting, the system is tuned for low-latency, according to the isolation in the [cpu-partitioning](#) figure. The application can use taskset to pin the reader and writer threads to CPUs 2 and 3, and the remaining application threads on CPUs 4-23.

Additional resources

- **tuned-profiles-cpu-partitioning(7)** man page

1.9. CUSTOMIZING THE CPU-PARTITIONING TUNED PROFILE

You can extend the Tuned profile to make additional tuning changes.

For example, the **cpu-partitioning** profile sets the CPUs to use **cstate=1**. In order to use the **cpu-partitioning** profile but to additionally change the CPU cstate from cstate1 to cstate0, the following procedure describes a new Tuned profile named *my_profile*, which inherits the **cpu-partitioning** profile and then sets C state-0.

Procedure

1. Create the **/etc/tuned/my_profile** directory:

```
# mkdir /etc/tuned/my_profile
```

2. Create a **tuned.conf** file in this directory, and add the following content:

```
# vi /etc/tuned/my_profile/tuned.conf
[main]
summary=Customized tuning on top of cpu-partitioning
include=cpu-partitioning
[cpu]
force_latency=cstate.id:0|1
```

3. Use the new profile:

```
# tuned-adm profile my_profile
```



NOTE

In the shared example, a reboot is not required. However, if the changes in the *my_profile* profile require a reboot to take effect, then reboot your machine.

Additional resources

- **tuned-profiles-cpu-partitioning(7)** man page

1.10. REAL-TIME TUNED PROFILES DISTRIBUTED WITH RHEL

Real-time profiles are intended for systems running the real-time kernel. Without a special kernel build, they do not configure the system to be real-time. On RHEL, the profiles are available from additional repositories.

The following real-time profiles are available:

realtime

Use on bare-metal real-time systems.

Provided by the **tuned-profiles-realtime** package, which is available from the RT or NFV repositories.

realtime-virtual-host

Use in a virtualization host configured for real-time.

Provided by the **tuned-profiles-nfv-host** package, which is available from the NFV repository.

realtime-virtual-guest

Use in a virtualization guest configured for real-time.

Provided by the **tuned-profiles-nfv-guest** package, which is available from the NFV repository.

1.11. STATIC AND DYNAMIC TUNING IN TUNED

Understanding the difference between the two categories of system tuning that **TuneD** applies, *static* and *dynamic*, is important when determining which one to use for a given situation or purpose.

Static tuning

Mainly consists of the application of predefined **sysctl** and **sysfs** settings and one-shot activation of several configuration tools such as **ethtool**.

Dynamic tuning

Watches how various system components are used throughout the uptime of your system. **TuneD** adjusts system settings dynamically based on that monitoring information.

For example, the hard drive is used heavily during startup and login, but is barely used later when the user might mainly work with applications such as web browsers or email clients. Similarly, the CPU and network devices are used differently at different times. **TuneD** monitors the activity of these components and reacts to the changes in their use.

By default, dynamic tuning is disabled. To enable it, edit the `/etc/tuned/tuned-main.conf` file and change the **dynamic_tuning** option to **1**. **TuneD** then periodically analyzes system statistics and uses them to update your system tuning settings. To configure the time interval in seconds between these updates, use the **update_interval** option.

Currently implemented dynamic tuning algorithms try to balance the performance and powersave, and are therefore disabled in the performance profiles. Dynamic tuning for individual plug-ins can be enabled or disabled in the **TuneD** profiles.

Example 1.2. Static and dynamic tuning on a workstation

On a typical office workstation, the Ethernet network interface is inactive most of the time. Only a few emails go in and out or some web pages might be loaded.

For those kinds of loads, the network interface does not have to run at full speed all the time, as it does by default. **TuneD** has a monitoring and tuning plug-in for network devices that can detect this low activity and then automatically lower the speed of that interface, typically resulting in a lower power usage.

If the activity on the interface increases for a longer period of time, for example because a DVD image is being downloaded or an email with a large attachment is opened, **TuneD** detects this and sets the interface speed to maximum to offer the best performance while the activity level is high.

This principle is used for other plug-ins for CPU and disks as well.

1.12. TUNED NO-DAEMON MODE

You can run **TuneD** in **no-daemon** mode, which does not require any resident memory. In this mode, **TuneD** applies the settings and exits.

By default, **no-daemon** mode is disabled because a lot of **TuneD** functionality is missing in this mode, including:

- D-Bus support
- Hot-plug support
- Rollback support for settings

To enable **no-daemon** mode, include the following line in the `/etc/tuned/tuned-main.conf` file:

```
daemon = 0
```

1.13. INSTALLING AND ENABLING TUNED

This procedure installs and enables the **TuneD** application, installs **TuneD** profiles, and presets a default **TuneD** profile for your system.

Procedure

1. Install the **TuneD** package:

```
# dnf install tuned
```

2. Enable and start the **TuneD** service:

```
# systemctl enable --now tuned
```

3. Optionally, install **TuneD** profiles for real-time systems:
For the **TuneD** profiles for real-time systems enable **rhel-9** repository.

```
# subscription-manager repos --enable=rhel-9-for-x86_64-nfv-beta-rpms
```

Install it.

```
# dnf install tuned-profiles-realtime tuned-profiles-nfv
```

4. Verify that a **TuneD** profile is active and applied:

```
$ tuned-adm active
```

Current active profile: *throughput-performance*



NOTE

The active profile TuneD automatically presets differs based on your machine type and system settings.

```
$ tuned-adm verify
```

Verification succeeded, current system settings match the preset profile.
See tuned log file ('/var/log/tuned/tuned.log') for details.

1.14. LISTING AVAILABLE TUNED PROFILES

This procedure lists all **TuneD** profiles that are currently available on your system.

Procedure

- To list all available **TuneD** profiles on your system, use:

```
$ tuned-adm list
```

Available profiles:

- accelerator-performance - Throughput performance based tuning with disabled higher latency STOP states

```

- balanced          - General non-specialized TuneD profile
- desktop           - Optimize for the desktop use-case
- latency-performance - Optimize for deterministic performance at the cost of increased
power consumption
- network-latency   - Optimize for deterministic performance at the cost of increased
power consumption, focused on low latency network performance
- network-throughput - Optimize for streaming network throughput, generally only
necessary on older CPUs or 40G+ networks
- powersave        - Optimize for low power consumption
- throughput-performance - Broadly applicable tuning that provides excellent performance
across a variety of common server workloads
- virtual-guest     - Optimize for running inside a virtual guest
- virtual-host      - Optimize for running KVM guests
Current active profile: balanced

```

- To display only the currently active profile, use:

```

$ tuned-adm active

Current active profile: throughput-performance

```

Additional resources

- **tuned-adm(8)** man page.

1.15. SETTING A TUNED PROFILE

This procedure activates a selected **TuneD** profile on your system.

Prerequisites

- The **TuneD** service is running. See [Installing and Enabling TuneD](#) for details.

Procedure

1. Optionally, you can let **TuneD** recommend the most suitable profile for your system:

```

# tuned-adm recommend

throughput-performance

```

2. Activate a profile:

```

# tuned-adm profile selected-profile

```

Alternatively, you can activate a combination of multiple profiles:

```

# tuned-adm profile selected-profile1 selected-profile2

```

Example 1.3. A virtual machine optimized for low power consumption

The following example optimizes the system to run in a virtual machine with the best performance and concurrently tunes it for low power consumption, while the low power consumption is the priority:

```
# tuned-adm profile virtual-guest powersave
```

3. View the current active **TuneD** profile on your system:

```
# tuned-adm active
```

```
Current active profile: selected-profile
```

4. Reboot the system:

```
# reboot
```

Verification steps

- Verify that the **TuneD** profile is active and applied:

```
$ tuned-adm verify
```

```
Verification succeeded, current system settings match the preset profile.  
See tuned log file ('/var/log/tuned/tuned.log') for details.
```

Additional resources

- **tuned-adm(8)** man page

1.16. DISABLING TUNED

This procedure disables **TuneD** and resets all affected system settings to their original state before **TuneD** modified them.

Procedure

- To disable all tunings temporarily:

```
# tuned-adm off
```

The tunings are applied again after the **TuneD** service restarts.

- Alternatively, to stop and disable the **TuneD** service permanently:

```
# systemctl disable --now tuned
```

Additional resources

- **tuned-adm(8)** man page

CHAPTER 2. CUSTOMIZING TUNED PROFILES

You can create or modify **Tuned** profiles to optimize system performance for your intended use case.

Prerequisites

- Install and enable **Tuned** as described in [Installing and Enabling Tuned](#) for details.

2.1. TUNED PROFILES

A detailed analysis of a system can be very time-consuming. **Tuned** provides a number of predefined profiles for typical use cases. You can also create, modify, and delete profiles.

The profiles provided with **Tuned** are divided into the following categories:

- Power-saving profiles
- Performance-boosting profiles

The performance-boosting profiles include profiles that focus on the following aspects:

- Low latency for storage and network
- High throughput for storage and network
- Virtual machine performance
- Virtualization host performance

Syntax of profile configuration

The **tuned.conf** file can contain one **[main]** section and other sections for configuring plug-in instances. However, all sections are optional.

Lines starting with the hash sign (**#**) are comments.

Additional resources

- **tuned.conf(5)** man page.

2.2. THE DEFAULT TUNED PROFILE

During the installation, the best profile for your system is selected automatically. Currently, the default profile is selected according to the following customizable rules:

Environment	Default profile	Goal
Compute nodes	throughput-performance	The best throughput performance
Virtual machines	virtual-guest	The best performance. If you are not interested in the best performance, you can change it to the balanced or powersave profile.

Environment	Default profile	Goal
Other cases	balanced	Balanced performance and power consumption

Additional resources

- **tuned.conf(5)** man page.

2.3. MERGED TUNED PROFILES

As an experimental feature, it is possible to select more profiles at once. **TuneD** will try to merge them during the load.

If there are conflicts, the settings from the last specified profile takes precedence.

Example 2.1. Low power consumption in a virtual guest

The following example optimizes the system to run in a virtual machine for the best performance and concurrently tunes it for low power consumption, while the low power consumption is the priority:

```
# tuned-adm profile virtual-guest powersave
```



WARNING

Merging is done automatically without checking whether the resulting combination of parameters makes sense. Consequently, the feature might tune some parameters the opposite way, which might be counterproductive: for example, setting the disk for high throughput by using the **throughput-performance** profile and concurrently setting the disk spindown to the low value by the **spindown-disk** profile.

Additional resources

***tuned-adm** man page. * **tuned.conf(5)** man page.

2.4. THE LOCATION OF TUNED PROFILES

TuneD stores profiles in the following directories:

/usr/lib/tuned/

Distribution-specific profiles are stored in the directory. Each profile has its own directory. The profile consists of the main configuration file called **tuned.conf**, and optionally other files, for example helper scripts.

/etc/tuned/

If you need to customize a profile, copy the profile directory into the directory, which is used for custom profiles. If there are two profiles of the same name, the custom profile located in **/etc/tuned/** is used.

Additional resources

- **tuned.conf(5)** man page.

2.5. INHERITANCE BETWEEN TUNED PROFILES

TuneD profiles can be based on other profiles and modify only certain aspects of their parent profile.

The **[main]** section of **TuneD** profiles recognizes the **include** option:

```
[main]
include=parent
```

All settings from the *parent* profile are loaded in this *child* profile. In the following sections, the *child* profile can override certain settings inherited from the *parent* profile or add new settings not present in the *parent* profile.

You can create your own *child* profile in the **/etc/tuned/** directory based on a pre-installed profile in **/usr/lib/tuned/** with only some parameters adjusted.

If the *parent* profile is updated, such as after a **TuneD** upgrade, the changes are reflected in the *child* profile.

Example 2.2. A power-saving profile based on balanced

The following is an example of a custom profile that extends the **balanced** profile and sets Aggressive Link Power Management (ALPM) for all devices to the maximum powersaving.

```
[main]
include=balanced

[scsi_host]
alpm=min_power
```

Additional resources

- **tuned.conf(5)** man page

2.6. STATIC AND DYNAMIC TUNING IN TUNED

Understanding the difference between the two categories of system tuning that **TuneD** applies, *static* and *dynamic*, is important when determining which one to use for a given situation or purpose.

Static tuning

Mainly consists of the application of predefined **sysctl** and **sysfs** settings and one-shot activation of several configuration tools such as **ethtool**.

Dynamic tuning

Watches how various system components are used throughout the uptime of your system. **TuneD** adjusts system settings dynamically based on that monitoring information.

For example, the hard drive is used heavily during startup and login, but is barely used later when the user might mainly work with applications such as web browsers or email clients. Similarly, the CPU and network devices are used differently at different times. **TuneD** monitors the activity of these components and reacts to the changes in their use.

By default, dynamic tuning is disabled. To enable it, edit the `/etc/tuned/tuned-main.conf` file and change the **dynamic_tuning** option to **1**. **TuneD** then periodically analyzes system statistics and uses them to update your system tuning settings. To configure the time interval in seconds between these updates, use the **update_interval** option.

Currently implemented dynamic tuning algorithms try to balance the performance and powersave, and are therefore disabled in the performance profiles. Dynamic tuning for individual plug-ins can be enabled or disabled in the **TuneD** profiles.

Example 2.3. Static and dynamic tuning on a workstation

On a typical office workstation, the Ethernet network interface is inactive most of the time. Only a few emails go in and out or some web pages might be loaded.

For those kinds of loads, the network interface does not have to run at full speed all the time, as it does by default. **TuneD** has a monitoring and tuning plug-in for network devices that can detect this low activity and then automatically lower the speed of that interface, typically resulting in a lower power usage.

If the activity on the interface increases for a longer period of time, for example because a DVD image is being downloaded or an email with a large attachment is opened, **TuneD** detects this and sets the interface speed to maximum to offer the best performance while the activity level is high.

This principle is used for other plug-ins for CPU and disks as well.

2.7. TUNED PLUG-INS

Plug-ins are modules in **TuneD** profiles that **TuneD** uses to monitor or optimize different devices on the system.

TuneD uses two types of plug-ins:

Monitoring plug-ins

Monitoring plug-ins are used to get information from a running system. The output of the monitoring plug-ins can be used by tuning plug-ins for dynamic tuning.

Monitoring plug-ins are automatically instantiated whenever their metrics are needed by any of the enabled tuning plug-ins. If two tuning plug-ins require the same data, only one instance of the monitoring plug-in is created and the data is shared.

Tuning plug-ins

Each tuning plug-in tunes an individual subsystem and takes several parameters that are populated from the **TuneD** profiles. Each subsystem can have multiple devices, such as multiple CPUs or network cards, that are handled by individual instances of the tuning plug-ins. Specific settings for individual devices are also supported.

Syntax for plug-ins in TuneD profiles

Sections describing plug-in instances are formatted in the following way:

```
[NAME]
type=TYPE
devices=DEVICES
```

NAME

is the name of the plug-in instance as it is used in the logs. It can be an arbitrary string.

TYPE

is the type of the tuning plug-in.

DEVICES

is the list of devices that this plug-in instance handles.

The **devices** line can contain a list, a wildcard (*), and negation (!). If there is no **devices** line, all devices present or later attached on the system of the *TYPE* are handled by the plug-in instance. This is same as using the **devices=*** option.

Example 2.4. Matching block devices with a plug-in

The following example matches all block devices starting with **sd**, such as **sda** or **sdb**, and does not disable barriers on them:

```
[data_disk]
type=disk
devices=sd*
disable_barriers=false
```

The following example matches all block devices except **sda1** and **sda2**:

```
[data_disk]
type=disk
devices=!sda1, !sda2
disable_barriers=false
```

If no instance of a plug-in is specified, the plug-in is not enabled.

If the plug-in supports more options, they can be also specified in the plug-in section. If the option is not specified and it was not previously specified in the included plug-in, the default value is used.

Short plug-in syntax

If you do not need custom names for the plug-in instance and there is only one definition of the instance in your configuration file, **TuneD** supports the following short syntax:

```
[TYPE]
devices=DEVICES
```

In this case, it is possible to omit the **type** line. The instance is then referred to with a name, same as the type. The previous example could be then rewritten into:

Example 2.5. Matching block devices using the short syntax

```
[disk]
devices=sdb*
disable_barriers=false
```

Conflicting plug-in definitions in a profile

If the same section is specified more than once using the **include** option, the settings are merged. If they cannot be merged due to a conflict, the last conflicting definition overrides the previous settings. If you do not know what was previously defined, you can use the **replace** Boolean option and set it to **true**. This causes all the previous definitions with the same name to be overwritten and the merge does not happen.

You can also disable the plug-in by specifying the **enabled=false** option. This has the same effect as if the instance was never defined. Disabling the plug-in is useful if you are redefining the previous definition from the **include** option and do not want the plug-in to be active in your custom profile.

NOTE

Tuned includes the ability to run any shell command as part of enabling or disabling a tuning profile. This enables you to extend **Tuned** profiles with functionality that has not been integrated into **Tuned** yet.

You can specify arbitrary shell commands using the **script** plug-in.

Additional resources

- **tuned.conf(5)** man page

2.8. AVAILABLE TUNED PLUG-INS

Monitoring plug-ins

Currently, the following monitoring plug-ins are implemented:

disk

Gets disk load (number of IO operations) per device and measurement interval.

net

Gets network load (number of transferred packets) per network card and measurement interval.

load

Gets CPU load per CPU and measurement interval.

Tuning plug-ins

Currently, the following tuning plug-ins are implemented. Only some of these plug-ins implement dynamic tuning. Options supported by plug-ins are also listed:

cpu

Sets the CPU governor to the value specified by the **governor** option and dynamically changes the Power Management Quality of Service (PM QoS) CPU Direct Memory Access (DMA) latency according to the CPU load.

If the CPU load is lower than the value specified by the **load_threshold** option, the latency is set to the value specified by the **latency_high** option, otherwise it is set to the value specified by **latency_low**.

You can also force the latency to a specific value and prevent it from dynamically changing further. To do so, set the **force_latency** option to the required latency value.

eeepc_she

Dynamically sets the front-side bus (FSB) speed according to the CPU load.

This feature can be found on some netbooks and is also known as the ASUS Super Hybrid Engine (SHE).

If the CPU load is lower or equal to the value specified by the **load_threshold_powersave** option, the plug-in sets the FSB speed to the value specified by the **she_powersave** option. If the CPU load is higher or equal to the value specified by the **load_threshold_normal** option, it sets the FSB speed to the value specified by the **she_normal** option.

Static tuning is not supported and the plug-in is transparently disabled if **TuneD** does not detect the hardware support for this feature.

net

Configures the Wake-on-LAN functionality to the values specified by the **wake_on_lan** option. It uses the same syntax as the **ethtool** utility. It also dynamically changes the interface speed according to the interface utilization.

sysctl

Sets various **sysctl** settings specified by the plug-in options.

The syntax is **name=value**, where *name* is the same as the name provided by the **sysctl** utility.

Use the **sysctl** plug-in if you need to change system settings that are not covered by other plug-ins available in **TuneD**. If the settings are covered by some specific plug-ins, prefer these plug-ins.

usb

Sets autosuspend timeout of USB devices to the value specified by the **autosuspend** parameter.

The value **0** means that autosuspend is disabled.

vm

Enables or disables transparent huge pages depending on the value of the **transparent_hugepages** option.

Valid values of the **transparent_hugepages** option are:

- "always"
- "never"
- "madvise"

audio

Sets the autosuspend timeout for audio codecs to the value specified by the **timeout** option.

Currently, the **snd_hda_intel** and **snd_ac97_codec** codecs are supported. The value **0** means that the autosuspend is disabled. You can also enforce the controller reset by setting the Boolean option **reset_controller** to **true**.

disk

Sets the disk elevator to the value specified by the **elevator** option.

It also sets:

- APM to the value specified by the **apm** option
- Scheduler quantum to the value specified by the **scheduler_quantum** option
- Disk spindown timeout to the value specified by the **spindown** option
- Disk readahead to the value specified by the **readahead** parameter
- The current disk readahead to a value multiplied by the constant specified by the **readahead_multiply** option

In addition, this plug-in dynamically changes the advanced power management and spindown timeout setting for the drive according to the current drive utilization. The dynamic tuning can be controlled by the Boolean option **dynamic** and is enabled by default.

scsi_host

Tunes options for SCSI hosts.

It sets Aggressive Link Power Management (ALPM) to the value specified by the **alpm** option.

mounts

Enables or disables barriers for mounts according to the Boolean value of the **disable_barriers** option.

script

Executes an external script or binary when the profile is loaded or unloaded. You can choose an arbitrary executable.



IMPORTANT

The **script** plug-in is provided mainly for compatibility with earlier releases. Prefer other **TuneD** plug-ins if they cover the required functionality.

TuneD calls the executable with one of the following arguments:

- **start** when loading the profile
- **stop** when unloading the profile

You need to correctly implement the **stop** action in your executable and revert all settings that you changed during the **start** action. Otherwise, the roll-back step after changing your **TuneD** profile will not work.

Bash scripts can import the **/usr/lib/tuned/functions** Bash library and use the functions defined there. Use these functions only for functionality that is not natively provided by **TuneD**. If a function name starts with an underscore, such as **_wifi_set_power_level**, consider the function private and do not use it in your scripts, because it might change in the future.

Specify the path to the executable using the **script** parameter in the plug-in configuration.

Example 2.6. Running a Bash script from a profile

To run a Bash script named **script.sh** that is located in the profile directory, use:

```
[script]
script=${i:PROFILE_DIR}/script.sh
```

sysfs

Sets various **sysfs** settings specified by the plug-in options.

The syntax is **name=value**, where *name* is the **sysfs** path to use.

Use this plugin in case you need to change some settings that are not covered by other plug-ins. Prefer specific plug-ins if they cover the required settings.

video

Sets various powersave levels on video cards. Currently, only the Radeon cards are supported.

The powersave level can be specified by using the **radeon_powersave** option. Supported values are:

- **default**
- **auto**
- **low**
- **mid**
- **high**
- **dynpm**
- **dpm-battery**
- **dpm-balanced**
- **dpm-performance**

For details, see www.x.org. Note that this plug-in is experimental and the option might change in future releases.

bootloader

Adds options to the kernel command line. This plug-in supports only the GRUB 2 boot loader. Customized non-standard location of the GRUB 2 configuration file can be specified by the **grub2_cfg_file** option.

The kernel options are added to the current GRUB configuration and its templates. The system needs to be rebooted for the kernel options to take effect.

Switching to another profile or manually stopping the **Tuned** service removes the additional options. If you shut down or reboot the system, the kernel options persist in the **grub.cfg** file.

The kernel options can be specified by the following syntax:

```
cmdline=arg1 arg2 ... argN
```

Example 2.7. Modifying the kernel command line

For example, to add the **quiet** kernel option to a **Tuned** profile, include the following lines in the **tuned.conf** file:

```
[bootloader]
cmdline=quiet
```

The following is an example of a custom profile that adds the **isolcpus=2** option to the kernel command line:

```
[bootloader]
cmdline=isolcpus=2
```

service

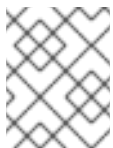
Handles various **sysvinit**, **sysv-rc**, **openrc**, and **systemd** services specified by the plug-in options. The syntax is **service.service_name=command[,file:file]**.

Supported service-handling commands are:

- **start**
- **stop**
- **enable**
- **disable**

Separate multiple commands using either a comma (,) or a semicolon (;). If the directives conflict, the **service** plugin uses the last listed one.

Use the optional **file:file** directive to install an overlay configuration file, **file**, for **systemd** only. Other init systems ignore this directive. The **service** plugin copies overlay configuration files to **/etc/systemd/system/service_name.service.d/** directories. Once profiles are unloaded, the **service** plugin removes these directories if they are empty.



NOTE

The **service** plugin only operates on the current runlevel with non- **systemd** init systems.

Example 2.8. Starting and enabling the sendmailservice service with an overlay file

```
[service]
service.sendmail=start,enable,file:${i:PROFILE_DIR}/tuned-sendmail.conf
```

The internal variable **\${i:PROFILE_DIR}** points to the directory the plugin loads the profile from.

scheduler

Offers a variety of options for the tuning of scheduling priorities, CPU core isolation, and process, thread, and IRQ affinities.

For specifics of the different options available, see [Functionalities of the scheduler TuneD plug-in](#).

2.9. FUNCTIONALITIES OF THE SCHEDULER TUNED PLUGIN

Use the **scheduler TuneD** plugin to control and tune scheduling priorities, CPU core isolation, and process, thread, and IRQ affinities.

CPU isolation

To prevent processes, threads, and IRQs from using certain CPUs, use the **isolated_cores** option. It changes process and thread affinities, IRQ affinities, and sets the **default_smp_affinity** parameter for IRQs.

The CPU affinity mask is adjusted for all processes and threads matching the **ps_whitelist** option, subject to success of the **sched_setaffinity()** system call. The default setting of the **ps_whitelist** regular expression is **.*** to match all processes and thread names. To exclude certain processes and threads, use the **ps_blacklist** option. The value of this option is also interpreted as a regular expression. Process and thread names are matched against that expression. Profile rollback enables all matching processes and threads to run on all CPUs, and restores the IRQ settings prior to the profile application.

Multiple regular expressions separated by **;** for the **ps_whitelist** and **ps_blacklist** options are supported. Escaped semicolon **\;** is taken literally.

Example 2.9. Isolate CPUs 2-4

The following configuration isolates CPUs 2-4. Processes and threads that match the **ps_blacklist** regular expression can use any CPUs regardless of the isolation:

```
[scheduler]
isolated_cores=2-4
ps_blacklist=.*pmd.*;.*PMD.*;^DPDK;.*qemu-kvm.*
```

IRQ SMP affinity

The **/proc/irq/default_smp_affinity** file contains a bitmask representing the default target CPU cores on a system for all inactive interrupt request (IRQ) sources. Once an IRQ is activated or allocated, the value in the **/proc/irq/default_smp_affinity** file determines the IRQ's affinity bitmask.

The **default_irq_smp_affinity** parameter controls what **TuneD** writes to the **/proc/irq/default_smp_affinity** file. The **default_irq_smp_affinity** parameter supports the following values and behaviors:

calc

Calculates the content of the **/proc/irq/default_smp_affinity** file from the **isolated_cores** parameter. An inversion of the **isolated_cores** parameter calculates the non-isolated cores. The intersection of the non-isolated cores and the previous content of the **/proc/irq/default_smp_affinity** file is then written to the **/proc/irq/default_smp_affinity** file.

This is the default behavior if the **default_irq_smp_affinity** parameter is omitted.

ignore

TuneD does not modify the **/proc/irq/default_smp_affinity** file.

A CPU list

Takes the form of a single number such as **1**, a comma separated list such as **1,3**, or a range such as **3-5**.

Unpacks the CPU list and writes it directly to the `/proc/irq/default_smp_affinity` file.

Example 2.10. Setting the default IRQ smp affinity using an explicit CPU list

The following example uses an explicit CPU list to set the default IRQ SMP affinity to CPUs 0 and 2:

```
[scheduler]
isolated_cores=1,3
default_irq_smp_affinity=0,2
```

Scheduling policy

To adjust scheduling policy, priority and affinity for a group of processes or threads, use the following syntax:

```
group.groupname=rule_prio:sched:prio:affinity:regex
```

where **rule_prio** defines internal **TuneD** priority of the rule. Rules are sorted based on priority. This is needed for inheritance to be able to reorder previously defined rules. Equal **rule_prio** rules should be processed in the order they were defined. However, this is Python interpreter dependent. To disable an inherited rule for **groupname**, use:

```
group.groupname=
```

sched must be one of the following:

- f**
for first in, first out (FIFO)
- b**
for batch
- r**
for round robin
- o**
for other
- ***
for do not change

affinity is CPU affinity in hexadecimal. Use ***** for no change.

prio is scheduling priority (see **chrt -m**).

regex is Python regular expression. It is matched against the output of the **ps -eo cmd** command.

Any given process name can match more than one group. In such cases, the last matching **regex** determines the priority and scheduling policy.

Example 2.11. Setting scheduling policies and priorities

The following example sets the scheduling policy and priorities to kernel threads and watchdog:


```
[scheduler]
group.kthreads=0:*:1:*:\[*\]$
group.watchdog=0:f:99:*:\[watchdog.*\]
```

The **scheduler** plugin uses a **perf** event loop to identify newly created processes. By default, it listens to **perf.RECORD_COMM** and **perf.RECORD_EXIT** events.

Setting the **perf_process_fork** parameter to **true** tells the plug-in to also listen to **perf.RECORD_FORK** events, meaning that child processes created by the **fork()** system call are processed.



NOTE

Processing **perf** events can pose a significant CPU overhead.

The CPU overhead of the scheduler plugin can be mitigated by using the scheduler **runtime** option and setting it to **0**. This completely disables the dynamic scheduler functionality and the **perf** events are not monitored and acted upon. The disadvantage of this is that the process and thread tuning will be done only at profile application.

Example 2.12. Disabling the dynamic scheduler functionality

The following example disables the dynamic scheduler functionality while also isolating CPUs 1 and 3:

```
[scheduler]
runtime=0
isolated_cores=1,3
```

The **mmaped** buffer is used for **perf** events. Under heavy loads, this buffer might overflow and as a result the plugin might start missing events and not processing some newly created processes. In such cases, use the **perf_mmap_pages** parameter to increase the buffer size. The value of the **perf_mmap_pages** parameter must be a power of 2. If the **perf_mmap_pages** parameter is not manually set, a default value of 128 is used.

Confinement using cgroups

The **scheduler** plugin supports process and thread confinement using **cgroups** v1.

The **cgroup_mount_point** option specifies the path to mount the cgroup file system, or, where **Tuned** expects it to be mounted. If it is unset, **/sys/fs/cgroup/cpuset** is expected.

If the **cgroup_groups_init** option is set to **1**, **Tuned** creates and removes all **cgroups** defined with the **cgroup*** options. This is the default behavior. If the **cgroup_mount_point** option is set to **0**, the **cgroups** must be preset by other means.

If the **cgroup_mount_point_init** option is set to **1**, **Tuned** creates and removes the cgroup mount point. It implies **cgroup_groups_init = 1**. If the **cgroup_mount_point_init** option is set to **0**, you must preset the **cgroups** mount point by other means. This is the default behavior.

The **cgroup_for_isolated_cores** option is the **cgroup** name for the **isolated_cores** option functionality. For example, if a system has 4 CPUs, **isolated_cores=1** means that **Tuned** moves all processes and threads to CPUs 0, 2, and 3. The **scheduler** plug-in isolates the specified core by writing

the calculated CPU affinity to the **cpuset.cpus** control file of the specified cgroup and moves all the matching processes and threads to this group. If this option is unset, classic cpuset affinity using **sched_setaffinity()** sets the CPU affinity.

The **cgroup.cgroup_name** option defines affinities for arbitrary **cgroups**. You can even use hierarchic cgroups, but you must specify the hierarchy in the correct order. **TuneD** does not do any sanity checks here, with the exception that it forces the **cgroup** to be in the location specified by the **cgroup_mount_point** option.

The syntax of the scheduler option starting with **group.** has been augmented to use **cgroup.cgroup_name** instead of the hexadecimal **affinity**. The matching processes are moved to the **cgroup cgroup_name**. You can also use cgroups not defined by the **cgroup.** option as described above. For example, **cgroups** not managed by **TuneD**.

All **cgroup** names are sanitized by replacing all periods (.) with slashes (/). This prevents the plugin from writing outside the location specified by the **cgroup_mount_point** option.

Example 2.13. Using cgroups v1 with the scheduler plug-in

The following example creates 2 **cgroups**, **group1** and **group2**. It sets the cgroup **group1** affinity to CPU 2 and the **cgroup group2** to CPUs 0 and 2. Given a 4 CPU setup, the **isolated_cores=1** option moves all processes and threads to CPU cores 0, 2, and 3. Processes and threads specified by the **ps_blacklist** regular expression are not moved.

```
[scheduler]
cgroup_mount_point=/sys/fs/cgroup/cpuset
cgroup_mount_point_init=1
cgroup_groups_init=1
cgroup_for_isolated_cores=group
cgroup.group1=2
cgroup.group2=0,2

group.ksoftirqd=0:f:2:cgroup.group1:ksoftirqd.*
ps_blacklist=ksoftirqd.*;rcuc.*;rcub.*;ktimersoftd.*
isolated_cores=1
```

The **cgroup_ps_blacklist** option excludes processes belonging to the specified **cgroups**. The regular expression specified by this option is matched against **cgroup** hierarchies from **/proc/PID/cgroups**. Commas (,) separate **cgroups v1** hierarchies from **/proc/PID/cgroups** before regular expression matching. The following is an example of content the regular expression is matched against:

```
10:hugetlb:/,9:perf_event:/,8:blkio:/
```

Multiple regular expressions can be separated by semicolons (;). The semicolon represents a logical 'or' operator.

Example 2.14. Excluding processes from the scheduler using cgroups

In the following example, the **scheduler** plug-in moves all processes away from core 1, except for processes which belong to cgroup **/daemons**. The **\b** string is a regular expression metacharacter that matches a word boundary.

```
[scheduler]
isolated_cores=1
cgroup_ps_blacklist=:/daemons\b
```

In the following example, the **scheduler** plugin excludes all processes which belong to a cgroup with a hierarchy-ID of 8 and controller-list **blkio**.

```
[scheduler]
isolated_cores=1
cgroup_ps_blacklist=\b8:blkio:
```

Recent kernels moved some **sched_** and **numa_balancing_** kernel run-time parameters from the **/proc/sys/kernel** directory managed by the **sysctl** utility, to **debugfs**, typically mounted under the **/sys/kernel/debug** directory. **TuneD** provides an abstraction mechanism for the following parameters via the **scheduler** plugin where, based on the kernel used, **TuneD** writes the specified value to the correct location:

- **sched_min_granularity_ns**
- **sched_latency_ns**,
- **sched_wakeup_granularity_ns**
- **sched_tunable_scaling**,
- **sched_migration_cost_ns**
- **sched_nr_migrate**
- **numa_balancing_scan_delay_ms**
- **numa_balancing_scan_period_min_ms**
- **numa_balancing_scan_period_max_ms**
- **numa_balancing_scan_size_mb**

Example 2.15. Set tasks' "cache hot" value for migration decisions.

On the old kernels, setting the following parameter meant that **sysctl** wrote a value of **500000** to the **/proc/sys/kernel/sched_migration_cost_ns** file:

```
[sysctl]
kernel.sched_migration_cost_ns=500000
```

This is, on more recent kernels, equivalent to setting the following parameter via the **scheduler** plugin:

```
[scheduler]
sched_migration_cost_ns=500000
```

Meaning **TuneD** writes a value of **500000** to the **/sys/kernel/debug/sched/migration_cost_ns** file.

2.10. VARIABLES IN TUNED PROFILES

Variables expand at run time when a **TuneD** profile is activated.

Using **TuneD** variables reduces the amount of necessary typing in **TuneD** profiles.

There are no predefined variables in **TuneD** profiles. You can define your own variables by creating the **[variables]** section in a profile and using the following syntax:

```
[variables]

variable_name=value
```

To expand the value of a variable in a profile, use the following syntax:

```
${variable_name}
```

Example 2.16. Isolating CPU cores using variables

In the following example, the **\${isolated_cores}** variable expands to **1,2**; hence the kernel boots with the **isolcpus=1,2** option:

```
[variables]
isolated_cores=1,2

[bootloader]
cmdline=isolcpus=${isolated_cores}
```

The variables can be specified in a separate file. For example, you can add the following lines to **tuned.conf**:

```
[variables]
include=/etc/tuned/my-variables.conf

[bootloader]
cmdline=isolcpus=${isolated_cores}
```

If you add the **isolated_cores=1,2** option to the **/etc/tuned/my-variables.conf** file, the kernel boots with the **isolcpus=1,2** option.

Additional resources

- **tuned.conf(5)** man page

2.11. BUILT-IN FUNCTIONS IN TUNED PROFILES

Built-in functions expand at run time when a **TuneD** profile is activated.

You can:

- Use various built-in functions together with **TuneD** variables

- Create custom functions in Python and add them to **TuneD** in the form of plug-ins

To call a function, use the following syntax:

```
${f:function_name:argument_1:argument_2}
```

To expand the directory path where the profile and the **tuned.conf** file are located, use the **PROFILE_DIR** function, which requires special syntax:

```
${i:PROFILE_DIR}
```

Example 2.17. Isolating CPU cores using variables and built-in functions

In the following example, the **\${non_isolated_cores}** variable expands to **0,3-5**, and the **cpulist_invert** built-in function is called with the **0,3-5** argument:

```
[variables]
non_isolated_cores=0,3-5

[bootloader]
cmdline=isolcpus=${f:cpulist_invert:${non_isolated_cores}}
```

The **cpulist_invert** function inverts the list of CPUs. For a 6-CPU machine, the inversion is **1,2**, and the kernel boots with the **isolcpus=1,2** command-line option.

Additional resources

- **tuned.conf(5)** man page

2.12. BUILT-IN FUNCTIONS AVAILABLE IN TUNED PROFILES

The following built-in functions are available in all **TuneD** profiles:

PROFILE_DIR

Returns the directory path where the profile and the **tuned.conf** file are located.

exec

Executes a process and returns its output.

assertion

Compares two arguments. If they *do not match*, the function logs text from the first argument and aborts profile loading.

assertion_non_equal

Compares two arguments. If they *match*, the function logs text from the first argument and aborts profile loading.

kb2s

Converts kilobytes to disk sectors.

s2kb

Converts disk sectors to kilobytes.

strip

Creates a string from all passed arguments and deletes both leading and trailing white space.

virt_check

Checks whether **TuneD** is running inside a virtual machine (VM) or on bare metal:

- Inside a VM, the function returns the first argument.
- On bare metal, the function returns the second argument, even in case of an error.

cpulist_invert

Inverts a list of CPUs to make its complement. For example, on a system with 4 CPUs, numbered from 0 to 3, the inversion of the list **0,2,3** is **1**.

cpulist2hex

Converts a CPU list to a hexadecimal CPU mask.

cpulist2hex_invert

Converts a CPU list to a hexadecimal CPU mask and inverts it.

hex2cpulist

Converts a hexadecimal CPU mask to a CPU list.

cpulist_online

Checks whether the CPUs from the list are online. Returns the list containing only online CPUs.

cpulist_present

Checks whether the CPUs from the list are present. Returns the list containing only present CPUs.

cpulist_unpack

Unpacks a CPU list in the form of **1-3,4** to **1,2,3,4**.

cpulist_pack

Packs a CPU list in the form of **1,2,3,5** to **1-3,5**.

2.13. CREATING NEW TUNED PROFILES

This procedure creates a new **TuneD** profile with custom performance rules.

Prerequisites

- The **TuneD** service is running. See [Installing and Enabling TuneD](#) for details.

Procedure

1. In the **/etc/tuned/** directory, create a new directory named the same as the profile that you want to create:

```
# mkdir /etc/tuned/my-profile
```

2. In the new directory, create a file named **tuned.conf**. Add a **[main]** section and plug-in definitions in it, according to your requirements.

For example, see the configuration of the **balanced** profile:

```
[main]
summary=General non-specialized TuneD profile
```

```
[cpu]
governor=conservative
energy_perf_bias=normal

[audio]
timeout=10

[video]
radeon_powersave=dpm-balanced, auto

[scsi_host]
alpm=medium_power
```

3. To activate the profile, use:

```
# tuned-adm profile my-profile
```

4. Verify that the **TuneD** profile is active and the system settings are applied:

```
$ tuned-adm active

Current active profile: my-profile

$ tuned-adm verify

Verification succeeded, current system settings match the preset profile.
See tuned log file ('/var/log/tuned/tuned.log') for details.
```

Additional resources

- **tuned.conf(5)** man page

2.14. MODIFYING EXISTING TUNED PROFILES

This procedure creates a modified child profile based on an existing **TuneD** profile.

Prerequisites

- The **TuneD** service is running. See [Installing and Enabling TuneD](#) for details.

Procedure

1. In the **/etc/tuned/** directory, create a new directory named the same as the profile that you want to create:

```
# mkdir /etc/tuned/modified-profile
```

2. In the new directory, create a file named **tuned.conf**, and set the **[main]** section as follows:

```
[main]
include=parent-profile
```

Replace *parent-profile* with the name of the profile you are modifying.

3. Include your profile modifications.

Example 2.18. Lowering swappiness in the throughput-performance profile

To use the settings from the **throughput-performance** profile and change the value of **vm.swappiness** to 5, instead of the default 10, use:

```
[main]
include=throughput-performance

[sysctl]
vm.swappiness=5
```

4. To activate the profile, use:

```
# tuned-adm profile modified-profile
```

5. Verify that the **TuneD** profile is active and the system settings are applied:

```
$ tuned-adm active

Current active profile: my-profile

$ tuned-adm verify

Verification succeeded, current system settings match the preset profile.
See tuned log file ('/var/log/tuned/tuned.log') for details.
```

Additional resources

- **tuned.conf(5)** man page

2.15. SETTING THE DISK SCHEDULER USING TUNED

This procedure creates and enables a **TuneD** profile that sets a given disk scheduler for selected block devices. The setting persists across system reboots.

In the following commands and configuration, replace:

- *device* with the name of the block device, for example **sdf**
- *selected-scheduler* with the disk scheduler that you want to set for the device, for example **bfq**

Prerequisites

- The **TuneD** service is installed and enabled. For details, see [Installing and enabling TuneD](#).

Procedure

1. Optional: Select an existing **TuneD** profile on which your profile will be based. For a list of available profiles, see [TuneD profiles distributed with RHEL](#). To see which profile is currently active, use:

—


```
$ tuned-adm active
```

2. Create a new directory to hold your **TunedD** profile:

```
# mkdir /etc/tuned/my-profile
```

3. Find the system unique identifier of the selected block device:

```
$ udevadm info --query=property --name=/dev/device | grep -E '(WWN|SERIAL)'

ID_WWN=0x5002538d00000000_
ID_SERIAL=Generic-SD_MMC_20120501030900000-0:0
ID_SERIAL_SHORT=20120501030900000
```



NOTE

The command in the this example will return all values identified as a World Wide Name (WWN) or serial number associated with the specified block device. Although it is preferred to use a WWN, the WWN is not always available for a given device and any values returned by the example command are acceptable to use as the *device system unique ID*.

4. Create the **/etc/tuned/my-profile/tuned.conf** configuration file. In the file, set the following options:

- a. Optional: Include an existing profile:

```
[main]
include=existing-profile
```

- b. Set the selected disk scheduler for the device that matches the WWN identifier:

```
[disk]
devices_udev_regex=IDNAME=device system unique id
elevator=selected-scheduler
```

Here:

- Replace *IDNAME* with the name of the identifier being used (for example, **ID_WWN**).
- Replace *device system unique id* with the value of the chosen identifier (for example, **0x5002538d00000000**).

To match multiple devices in the **devices_udev_regex** option, enclose the identifiers in parentheses and separate them with vertical bars:

```
devices_udev_regex=(ID_WWN=0x5002538d00000000)|
(ID_WWN=0x1234567800000000)
```

5. Enable your profile:

```
# tuned-adm profile my-profile
```

Verification steps

1. Verify that the TuneD profile is active and applied:

```
$ tuned-adm active
```

```
Current active profile: my-profile
```

```
$ tuned-adm verify
```

```
Verification succeeded, current system settings match the preset profile.  
See TuneD log file ('/var/log/tuned/tuned.log') for details.
```

2. Read the contents of the **/sys/block/*device*/queue/scheduler** file:

```
# cat /sys/block/device/queue/scheduler
```

```
[mq-deadline] kyber bfq none
```

In the file name, replace *device* with the block device name, for example **sdc**.

The active scheduler is listed in square brackets (**[]**).

Additional resources

- [Customizing TuneD profiles.](#)

CHAPTER 3. REVIEWING A SYSTEM USING TUNA INTERFACE

Use the **tuna** tool to adjust scheduler tunables, tune thread priority, IRQ handlers, and isolate CPU cores and sockets. Tuna reduces the complexity of performing tuning tasks.

The **tuna** tool performs the following operations:

- Lists the CPUs on a system
- Lists the interrupt requests (IRQs) currently running on a system
- Changes policy and priority information about threads
- Displays the current policies and priorities of a system

3.1. INSTALLING TUNA TOOL

The **tuna** tool is designed to be used on a running system. This allows application-specific measurement tools to see and analyze system performance immediately after changes have been made.

This procedure describes how to install the **tuna** tool.

Procedure

- Install the **tuna** tool:

```
# dnf install tuna
```

Verification steps

- View the available **tuna** CLI options:

```
# tuna -h
```

Additional resources

- **tuna(8)** man page

3.2. VIEWING THE SYSTEM STATUS USING TUNA TOOL

This procedure describes how to view the system status using the **tuna** command-line interface (CLI) tool.

Prerequisites

- The tuna tool is installed. For more information, see [Installing tuna tool](#).

Procedure

- To view the current policies and priorities:

```
# tuna --show_threads  
thread
```

```

pid  SCHED_ rtpri affinity      cmd
1    OTHER  0      0,1      init
2    FIFO   99      0      migration/0
3    OTHER  0      0      ksoftirqd/0
4    FIFO   99      0      watchdog/0

```

- To view a specific thread corresponding to a PID or matching a command name:

```
# tuna --threads=pid_or_cmd_list --show_threads
```

The *pid_or_cmd_list* argument is a list of comma-separated PIDs or command-name patterns.

- To tune CPUs using the **tuna** CLI, see [Tuning CPUs using tuna tool](#).
- To tune the IRQs using the **tuna** tool, see [Tuning IRQs using tuna tool](#).
- To save the changed configuration:

```
# tuna --save=filename
```

This command saves only currently running kernel threads. Processes that are not running are not saved.

Additional resources

- **tuna(8)** man page

3.3. TUNING CPUS USING TUNA TOOL

The **tuna** tool commands can target individual CPUs.

Using the tuna tool, you can:

Isolate CPUs

All tasks running on the specified CPU move to the next available CPU. Isolating a CPU makes it unavailable by removing it from the affinity mask of all threads.

Include CPUs

Allows tasks to run on the specified CPU

Restore CPUs

Restores the specified CPU to its previous configuration.

This procedure describes how to tune CPUs using the **tuna** CLI.

Prerequisites

- The tuna tool is installed. For more information, see [Installing tuna tool](#).

Procedure

- To specify the list of CPUs to be affected by a command:

```
# tuna --cpus=cpu_list [command]
```

The *cpu_list* argument is a list of comma-separated CPU numbers. For example, **--cpus=0,2**. CPU lists can also be specified in a range, for example **--cpus="1-3"**, which would select CPUs 1, 2, and 3.

To add a specific CPU to the current *cpu_list*, for example, use **--cpus=+0**.

Replace *[command]* with, for example, **--isolate**.

- To isolate a CPU:

```
# tuna --cpus=cpu_list --isolate
```

- To include a CPU:

```
# tuna --cpus=cpu_list --include
```

- To use a system with four or more processors, display how to make all the ssh threads run on CPU 0 and 1, and all the **http** threads on CPU 2 and 3:

```
# tuna --cpus=0,1 --threads=ssh\* \
--move --cpus=2,3 --threads=http\* --move
```

This command performs the following operations sequentially:

1. Selects CPUs 0 and 1.
2. Selects all threads that begin with **ssh**.
3. Moves the selected threads to the selected CPUs. Tuna sets the affinity mask of threads starting with **ssh** to the appropriate CPUs. The CPUs can be expressed numerically as 0 and 1, in hex mask as 0x3, or in binary as 11.
4. Resets the CPU list to 2 and 3.
5. Selects all threads that begin with **http**.
6. Moves the selected threads to the specified CPUs. Tuna sets the affinity mask of threads starting with **http** to the specified CPUs. The CPUs can be expressed numerically as 2 and 3, in hex mask as 0xC, or in binary as 1100.

Verification steps

- Display the current configuration and verify that the changes were performed as expected:

```
# tuna --threads=gnome-sc\* --show_threads \
--cpus=0 --move --show_threads --cpus=1 \
--move --show_threads --cpus=+0 --move --show_threads
```

	thread	ctxt_switches					
	pid	SCHED_	rtpri	affinity	voluntary	nonvoluntary	cmd
	3861	OTHER	0	0,1	33997	58	gnome-screensav

	thread	ctxt_switches					
	pid	SCHED_	rtpri	affinity	voluntary	nonvoluntary	cmd
	3861	OTHER	0	0	33997	58	gnome-screensav

```

pid SCHED_ rtpri affinity voluntary nonvoluntary      cmd
3861  OTHER    0      1  33997      58 gnome-screensav
      thread    ctxt_switches
pid SCHED_ rtpri affinity voluntary nonvoluntary      cmd
3861  OTHER    0    0,1  33997      58 gnome-screensav

```

This command performs the following operations sequentially:

1. Selects all threads that begin with the **gnome-sc** threads.
2. Displays the selected threads to enable the user to verify their affinity mask and RT priority.
3. Selects CPU 0.
4. Moves the **gnome-sc** threads to the specified CPU, CPU 0.
5. Shows the result of the move.
6. Resets the CPU list to CPU 1.
7. Moves the **gnome-sc** threads to the specified CPU, CPU 1.
8. Displays the result of the move.
9. Adds CPU 0 to the CPU list.
10. Moves the **gnome-sc** threads to the specified CPUs, CPUs 0 and 1.
11. Displays the result of the move.

Additional resources

- `/proc/cpuinfo` file
- **tuna(8)** man page

3.4. TUNING IRQS USING TUNA TOOL

The `/proc/interrupts` file records the number of interrupts per IRQ, the type of interrupt, and the name of the device that is located at that IRQ.

This procedure describes how to tune the IRQs using the **tuna** tool.

Prerequisites

- The tuna tool is installed. For more information, see [Installing tuna tool](#).

Procedure

- To view the current IRQs and their affinity:

```

# tuna --show_irqs
# users      affinity
0 timer      0
1 i8042      0
7 parport0   0

```

- To specify the list of IRQs to be affected by a command:

```
# tuna --irqs=irq_list [command]
```

The *irq_list* argument is a list of comma-separated IRQ numbers or user-name patterns.

Replace [*command*] with, for example, **--spread**.

- To move an interrupt to a specified CPU:

```
# tuna --irqs=128 --show_irqs
# users      affinity
128 iwlwifi   0,1,2,3

# tuna --irqs=128 --cpus=3 --move
```

Replace *128* with the *irq_list* argument and *3* with the *cpu_list* argument.

The *cpu_list* argument is a list of comma-separated CPU numbers, for example, **--cpus=0,2**. For more information, see [Tuning CPUs using tuna tool](#).

Verification steps

- Compare the state of the selected IRQs before and after moving any interrupt to a specified CPU:

```
# tuna --irqs=128 --show_irqs
# users      affinity
128 iwlwifi   3
```

Additional resources

- **/procs/interrupts** file
- **tuna(8)** man page

CHAPTER 4. MONITORING PERFORMANCE USING RHEL SYSTEM ROLES

As a system administrator, you can use the **metrics** RHEL System Role to monitor the performance of a system.

4.1. PREPARING A CONTROL NODE AND MANAGED NODES TO USE RHEL SYSTEM ROLES

Before you can use individual RHEL System Roles to manage services and settings, you must prepare the control node and managed nodes.

4.1.1. Preparing a control node on RHEL 9

Before using RHEL System Roles, you must configure a control node. This system then configures the managed hosts from the inventory according to the playbooks.

Prerequisites

- RHEL 8.6 or later is installed. For more information about installing RHEL, see [Performing a standard RHEL 9 installation](#).
- The system is registered to the Customer Portal.
- A **Red Hat Enterprise Linux Server** subscription is attached to the system.
- If available in your Customer Portal account, an **Ansible Automation Platform** subscription is attached to the system.

Procedure

1. Install the **rhel-system-roles** package:

```
[root@control-node]# dnf install rhel-system-roles
```

This command installs the **ansible-core** package as a dependency.



NOTE

In RHEL 8.5 and earlier versions, Ansible packages were provided through Ansible Engine instead of Ansible Core, and with a different level of support. Do not use Ansible Engine because the packages might not be compatible with Ansible automation content in RHEL 8.6 and later. For more information, see [Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories](#).

2. Create a user named **ansible** to manage and run playbooks:

```
[root@control-node]# useradd ansible
```

3. Switch to the newly created **ansible** user:


```
[root@control-node]# su - ansible
```

Perform the rest of the procedure as this user.

4. Create an SSH public and private key:

```
[ansible@control-node]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ansible/.ssh/id_rsa): <password>
...
```

Use the suggested default location for the key file.

5. Optional: To prevent Ansible from prompting you for the SSH key password each time you establish a connection, configure an SSH agent.
6. Create the `~/.ansible.cfg` file with the following content:

```
[defaults]
inventory = /home/ansible/inventory
remote_user = ansible

[privilege_escalation]
become = True
become_method = sudo
become_user = root
become_ask_pass = True
```



NOTE

Settings in the `~/.ansible.cfg` file have a higher priority and override settings from the global `/etc/ansible/ansible.cfg` file.

With these settings, Ansible performs the following actions:

- Manages hosts in the specified inventory file.
 - Uses the account set in the **remote_user** parameter when it establishes SSH connections to managed nodes.
 - Uses the **sudo** utility to execute tasks on managed nodes as the **root** user.
 - Prompts for the root password of the remote user every time you apply a playbook. This is recommended for security reasons.
7. Create an `~/.inventory` file in INI or YAML format that lists the hostnames of managed hosts. You can also define groups of hosts in the inventory file. For example, the following is an inventory file in the INI format with three hosts and one host group named **US**:

```
managed-node-01.example.com

[US]
managed-node-02.example.com ansible_host=192.0.2.100
managed-node-03.example.com
```

Note that the control node must be able to resolve the hostnames. If the DNS server cannot resolve certain hostnames, add the **ansible_host** parameter next to the host entry to specify its IP address.

Next steps

- Prepare the managed nodes. For more information, see [Preparing a managed node](#).

Additional resources

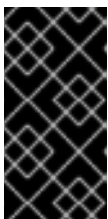
- [Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories](#)
- [How to register and subscribe a system to the Red Hat Customer Portal using subscription-manager](#)
- The **ssh-keygen(1)** man page
- [Connecting to remote machines with SSH keys using ssh-agent](#)
- [Ansible configuration settings](#)
- [How to build your inventory](#)

4.1.2. Preparing a managed node

Managed nodes are the systems listed in the inventory and which will be configured by the control node according to the playbook. You do not have to install Ansible on managed hosts.

Prerequisites

- You prepared the control node. For more information, see [Preparing a control node on RHEL 9](#).
- You have SSH access from the control node.



IMPORTANT

Direct SSH access as the **root** user is a security risk. To reduce this risk, you will create a local user on this node and configure a **sudo** policy when preparing a managed node. Ansible on the control node can then use the local user account to log in to the managed node and run playbooks as different users, such as **root**.

Procedure

1. Create a user named **ansible**:

```
[root@managed-node-01]# useradd ansible
```

The control node later uses this user to establish an SSH connection to this host.

2. Set a password for the **ansible** user:

```
[root@managed-node-01]# passwd ansible
Changing password for user ansible.
New password: <password>
```

Retype new password: *<password>*
 passwd: all authentication tokens updated successfully.

You must enter this password when Ansible uses **sudo** to perform tasks as the **root** user.

3. Install the **ansible** user's SSH public key on the managed node:

- a. Log in to the control node as the **ansible** user, and copy the SSH public key to the managed node:

```
[ansible@control-node]$ ssh-copy-id managed-node-01.example.com
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed:
"/home/ansible/.ssh/id_rsa.pub"
The authenticity of host 'managed-node-01.example.com (192.0.2.100)' can't be
established.
ECDSA key fingerprint is
SHA256:9bZ33GJNODK3zbNhybokN/6Mq7hu3vpBXDrCxe7NAvo.
```

- b. When prompted, connect by entering **yes**:

```
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that
are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is
to install the new keys
```

- c. When prompted, enter the password:

```
ansible@managed-node-01.example.com's password: <password>

Number of key(s) added: 1

Now try logging into the machine, with: "ssh '<managed-node-01.example.com>'"
and check to make sure that only the key(s) you wanted were added.
```

- d. Verify the SSH connection by remotely executing a command on the control node:

```
[ansible@control-node]$ ssh <managed-node-01.example.com> whoami
ansible
```

4. Create a **sudo** configuration for the **ansible** user:

- a. Create and edit the **/etc/sudoers.d/ansible** file by using the **visudo** command:

```
[root@managed-node-01]# visudo /etc/sudoers.d/ansible
```

The benefit of using **visudo** over a normal editor is that this utility provides basic sanity checks and checks for parse errors before installing the file.

- b. Configure a **sudoers** policy in the **/etc/sudoers.d/ansible** file that meets your requirements, for example:
- To grant permissions to the **ansible** user to run all commands as any user and group on this host after entering the **ansible** user's password, use:

ansible ALL=(ALL) ALL

- To grant permissions to the **ansible** user to run all commands as any user and group on this host without entering the **ansible** user's password, use:

ansible ALL=(ALL) NOPASSWD: ALL

Alternatively, configure a more fine-granular policy that matches your security requirements. For further details on **sudoers** policies, see the **sudoers(5)** man page.

Verification

1. Verify that you can execute commands from the control node on an all managed nodes:

```
[ansible@control-node]$ ansible all -m ping
BECOME password: <password>
managed-node-01.example.com | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
...
```

The hard-coded all group dynamically contains all hosts listed in the inventory file.

2. Verify that privilege escalation works correctly by running the **whoami** utility on a managed host by using the Ansible **command** module:

```
[ansible@control-node]$ ansible managed-node-01.example.com -m command -a
whoami
BECOME password: <password>
managed-node-01.example.com | CHANGED | rc=0 >>
root
```

If the command returns root, you configured **sudo** on the managed nodes correctly.

Additional resources

- [Preparing a control node on RHEL 9](#) .
- The **sudoers(5)** man page

4.2. INTRODUCTION TO THE METRICS SYSTEM ROLE

RHEL System Roles is a collection of Ansible roles and modules that provide a consistent configuration interface to remotely manage multiple RHEL systems. The **metrics** System Role configures performance analysis services for the local system and, optionally, includes a list of remote systems to be monitored by the local system. The **metrics** System Role enables you to use **pcp** to monitor your systems performance without having to configure **pcp** separately, as the set-up and deployment of **pcp** is handled by the playbook.

Table 4.1. metrics system role variables

Role variable	Description	Example usage
<code>metrics_monitored_hosts</code>	List of remote hosts to be analyzed by the target host. These hosts will have metrics recorded on the target host, so ensure enough disk space exists below <code>/var/log</code> for each host.	metrics_monitored_hosts: <code>["webserver.example.com", "database.example.com"]</code>
<code>metrics_retention_days</code>	Configures the number of days for performance data retention before deletion.	metrics_retention_days: 14
<code>metrics_graph_service</code>	A boolean flag that enables the host to be set up with services for performance data visualization via pcp and grafana . Set to false by default.	metrics_graph_service: no
<code>metrics_query_service</code>	A boolean flag that enables the host to be set up with time series query services for querying recorded pcp metrics via redis . Set to false by default.	metrics_query_service: no
<code>metrics_provider</code>	Specifies which metrics collector to use to provide metrics. Currently, pcp is the only supported metrics provider.	metrics_provider: "pcp"
<code>metrics_manage_firewall</code>	Uses the firewall role to manage port access directly from the metrics role. Set to false by default.	metrics_manage_firewall: true
<code>metrics_manage_selinux</code>	Uses the selinux role to manage port access directly from the metrics role. Set to false by default.	metrics_manage_selinux: true

**NOTE**

For details about the parameters used in **metrics_connections** and additional information about the **metrics** System Role, see the `/usr/share/ansible/roles/rhel-system-roles.metrics/README.md` file.

4.3. USING THE METRICS SYSTEM ROLE TO MONITOR YOUR LOCAL SYSTEM WITH VISUALIZATION

This procedure describes how to use the **metrics** RHEL System Role to monitor your local system while simultaneously provisioning data visualization via **Grafana**.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the machine you want to monitor.

Procedure

1. Configure **localhost** in the **/etc/ansible/hosts** Ansible inventory by adding the following content to the inventory:

```
localhost ansible_connection=local
```

2. Create an Ansible playbook with the following content:

```
---
- name: Manage metrics
  hosts: localhost
  vars:
    metrics_graph_service: yes
    metrics_manage_firewall: true
    metrics_manage_selinux: true
  roles:
    - rhel-system-roles.metrics
```

3. Run the Ansible playbook:

```
# ansible-playbook name_of_your_playbook.yml
```



NOTE

Because the **metrics_graph_service** boolean is set to **value="yes"**, **Grafana** is automatically installed and provisioned with **pcp** added as a data source. Because **metrics_manage_firewall** and **metrics_manage_selinux** are both set to **true**, the metrics role uses the **firewall** and **selinux** system roles to manage the ports used by the metrics role.

4. To view visualization of the metrics being collected on your machine, access the **grafana** web interface as described in [Accessing the Grafana web UI](#).

4.4. USING THE METRICS SYSTEM ROLE TO SET UP A FLEET OF INDIVIDUAL SYSTEMS TO MONITOR THEMSELVES

This procedure describes how to use the **metrics** System Role to set up a fleet of machines to monitor themselves.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the machine you want to use to run the playbook.

- You have the SSH connection established.

Procedure

1. Add the name or IP address of the machines you want to monitor via the playbook to the **/etc/ansible/hosts** Ansible inventory file under an identifying group name enclosed in brackets:

```
[remotes]
webserver.example.com
database.example.com
```

2. Create an Ansible playbook with the following content:

```
---
- hosts: remotes
  vars:
    metrics_retention_days: 0
    metrics_manage_firewall: true
    metrics_manage_selinux: true
  roles:
    - rhel-system-roles.metrics
```



NOTE

Because **metrics_manage_firewall** and **metrics_manage_selinux** are both set to **true**, the metrics role uses the **firewall** and **selinux** roles to manage the ports used by the **metrics** role.

3. Run the Ansible playbook:

```
# ansible-playbook name_of_your_playbook.yml -k
```

Where the **-k** prompt for password to connect to remote system.

4.5. USING THE METRICS SYSTEM ROLE TO MONITOR A FLEET OF MACHINES CENTRALLY VIA YOUR LOCAL MACHINE

This procedure describes how to use the **metrics** System Role to set up your local machine to centrally monitor a fleet of machines while also provisioning visualization of the data via **grafana** and querying of the data via **redis**.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the machine you want to use to run the playbook.

Procedure

1. Create an Ansible playbook with the following content:

```

---
- hosts: localhost
  vars:
    metrics_graph_service: yes
    metrics_query_service: yes
    metrics_retention_days: 10
    metrics_monitored_hosts: ["database.example.com", "webserver.example.com"]
    metrics_manage_firewall: yes
    metrics_manage_selinux: yes
  roles:
    - rhel-system-roles.metrics

```

2. Run the Ansible playbook:

```
# ansible-playbook name_of_your_playbook.yml
```



NOTE

Because the **metrics_graph_service** and **metrics_query_service** booleans are set to **value="yes"**, **grafana** is automatically installed and provisioned with **pcp** added as a data source with the **pcp** data recording indexed into **redis**, allowing the **pcp** querying language to be used for complex querying of the data. Because **metrics_manage_firewall** and **metrics_manage_selinux** are both set to **true**, the **metrics** role uses the **firewall** and **selinux** roles to manage the ports used by the **metrics** role.

3. To view a graphical representation of the metrics being collected centrally by your machine and to query the data, access the **grafana** web interface as described in [Accessing the Grafana web UI](#).

4.6. SETTING UP AUTHENTICATION WHILE MONITORING A SYSTEM USING THE METRICS SYSTEM ROLE

PCP supports the **scram-sha-256** authentication mechanism through the Simple Authentication Security Layer (SASL) framework. The **metrics** RHEL System Role automates the steps to setup authentication using the **scram-sha-256** authentication mechanism. This procedure describes how to setup authentication using the **metrics** RHEL System Role.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the machine you want to use to run the playbook.

Procedure

1. Include the following variables in the Ansible playbook you want to setup authentication for:

```

---
vars:
  metrics_username: your_username

```



```
metrics_password: your_password
metrics_manage_firewall: true
metrics_manage_selinux: true
```



NOTE

Because **metrics_manage_firewall** and **metrics_manage_selinux** are both set to **true**, the **metrics** role uses the **firewall** and **selinux** roles to manage the ports used by the **metrics** role.

2. Run the Ansible playbook:

```
# ansible-playbook name_of_your_playbook.yml
```

Verification steps

- Verify the **sasl** configuration:

```
# pminfo -f -h "pcp://ip_adress?username=your_username" disk.dev.read
Password:
disk.dev.read
inst [0 or "sda"] value 19540
```

ip_adress should be replaced by the IP address of the host.

4.7. USING THE METRICS SYSTEM ROLE TO CONFIGURE AND ENABLE METRICS COLLECTION FOR SQL SERVER

This procedure describes how to use the **metrics** RHEL System Role to automate the configuration and enabling of metrics collection for Microsoft SQL Server via **pcp** on your local system.

Prerequisites

- The Ansible Core package is installed on the control machine.
- You have the **rhel-system-roles** package installed on the machine you want to monitor.
- You have installed Microsoft SQL Server for Red Hat Enterprise Linux and established a 'trusted' connection to an SQL server. See [Install SQL Server and create a database on Red Hat](#) .
- You have installed the Microsoft ODBC driver for SQL Server for Red Hat Enterprise Linux. See [Red Hat Enterprise Server and Oracle Linux](#) .

Procedure

1. Configure **localhost** in the **/etc/ansible/hosts** Ansible inventory by adding the following content to the inventory:

```
localhost ansible_connection=local
```

2. Create an Ansible playbook that contains the following content:

```

---
- hosts: localhost
  vars:
    metrics_from_mssql: true
    metrics_manage_firewall: true
    metrics_manage_selinux: true
  roles:
    - role: rhel-system-roles.metrics

```



NOTE

Because **metrics_manage_firewall** and **metrics_manage_selinux** are both set to **true**, the **metrics** role uses the **firewall** and **selinux** roles to manage the ports used by the **metrics** role.

3. Run the Ansible playbook:

```
# ansible-playbook name_of_your_playbook.yml
```

Verification steps

- Use the **pcp** command to verify that SQL Server PMDA agent (mssql) is loaded and running:

```

# pcp
platform: Linux rhel82-2.local 4.18.0-167.el8.x86_64 #1 SMP Sun Dec 15 01:24:23 UTC
2019 x86_64
hardware: 2 cpus, 1 disk, 1 node, 2770MB RAM
timezone: PDT+7
services: pmcd pmproxy
  pmcd: Version 5.0.2-1, 12 agents, 4 clients
  pmda: root pmcd proc pmproxy xfs linux nfsclient mmv kvm mssql
        jbd2 dm
pmlogger: primary logger: /var/log/pcp/pmllogger/rhel82-2.local/20200326.16.31
pmie: primary engine: /var/log/pcp/pmie/rhel82-2.local/pmie.log

```

Additional resources

- [For more information about using Performance Co-Pilot for Microsoft SQL Server, see this Red Hat Developers Blog post.](#)

CHAPTER 5. SETTING UP PCP

Performance Co-Pilot (PCP) is a suite of tools, services, and libraries for monitoring, visualizing, storing, and analyzing system-level performance measurements.

5.1. OVERVIEW OF PCP

You can add performance metrics using Python, Perl, C++, and C interfaces. Analysis tools can use the Python, C++, C client APIs directly, and rich web applications can explore all available performance data using a JSON interface.

You can analyze data patterns by comparing live results with archived data.

Features of PCP:

- Light-weight distributed architecture, which is useful during the centralized analysis of complex systems.
- It allows the monitoring and management of real-time data.
- It allows logging and retrieval of historical data.

PCP has the following components:

- The Performance Metric Collector Daemon (**pmcd**) collects performance data from the installed Performance Metric Domain Agents (**pmda**). **PMDAs** can be individually loaded or unloaded on the system and are controlled by the **PMCD** on the same host.
- Various client tools, such as **pminfo** or **pmstat**, can retrieve, display, archive, and process this data on the same host or over the network.
- The **pcp** package provides the command-line tools and underlying functionality.
- The **pcp-gui** package provides the graphical application. Install the **pcp-gui** package by executing the **dnf install pcp-gui** command. For more information, see [Visually tracing PCP log archives with the PCP Charts application](#).

Additional resources

- **pcp(1)** man page
- **/usr/share/doc/pcp-doc/** directory
- [Tools distributed with PCP](#)
- [Index of Performance Co-Pilot \(PCP\) articles, solutions, tutorials, and white papers fromon Red Hat Customer Portal](#)
- [Side-by-side comparison of PCP tools with legacy tools Red Hat Knowledgebase article](#)
- [PCP upstream documentation](#)

5.2. INSTALLING AND ENABLING PCP

To begin using PCP, install all the required packages and enable the PCP monitoring services.

This procedure describes how to install PCP using the **pcp** package. If you want to automate the PCP installation, install it using the **pcp-zeroconf** package. For more information about installing PCP by using **pcp-zeroconf**, see [Setting up PCP with pcp-zeroconf](#).

Procedure

1. Install the **pcp** package:

```
# dnf install pcp
```

2. Enable and start the **pmcd** service on the host machine:

```
# systemctl enable pmcd
```

```
# systemctl start pmcd
```

Verification steps

- Verify if the **pmcd** process is running on the host:

```
# pcp
```

Performance Co-Pilot configuration on workstation:

```
platform: Linux workstation 4.18.0-80.el8.x86_64 #1 SMP Wed Mar 13 12:02:46 UTC 2019
x86_64
hardware: 12 cpus, 2 disks, 1 node, 36023MB RAM
timezone: CEST-2
services: pmcd
pmcd: Version 4.3.0-1, 8 agents
pmda: root pmcd proc xfs linux mmv kvm jbd2
```

Additional resources

- **pmcd(1)** man page
- [Tools distributed with PCP](#)

5.3. DEPLOYING A MINIMAL PCP SETUP

The minimal PCP setup collects performance statistics on Red Hat Enterprise Linux. The setup involves adding the minimum number of packages on a production system needed to gather data for further analysis.

You can analyze the resulting **tar.gz** file and the archive of the **pmlogger** output using various PCP tools and compare them with other sources of performance information.

Prerequisites

- PCP is installed. For more information, see [Installing and enabling PCP](#).

Procedure

1. Update the **pmlogger** configuration:

```
# pmlogconf -r /var/lib/pcp/config/pmlogger/config.default
```

2. Start the **pmcd** and **pmlogger** services:

```
# systemctl start pmcd.service
# systemctl start pmlogger.service
```

3. Execute the required operations to record the performance data.

4. Stop the **pmcd** and **pmlogger** services:

```
# systemctl stop pmcd.service
# systemctl stop pmlogger.service
```

5. Save the output and save it to a **tar.gz** file named based on the host name and the current date and time:

```
# cd /var/log/pcp/pmlogger/
# tar -czf $(hostname).$(date +%F-%H%M).pcp.tar.gz $(hostname)
```

Extract this file and analyze the data using PCP tools.

Additional resources

- **pmlogconf(1)**, **pmlogger(1)**, and **pmcd(1)** man pages
- [Tools distributed with PCP](#)
- [System services distributed with PCP](#)

5.4. SYSTEM SERVICES DISTRIBUTED WITH PCP

The following table describes roles of various system services, which are distributed with PCP.

Table 5.1. Roles of system services distributed with PCP

Name	Description
pmcd	The Performance Metric Collector Daemon (PMCD).
pmie	The Performance Metrics Inference Engine.
pmlogger	The performance metrics logger.
pmproxy	The realtime and historical performance metrics proxy, time series query and REST API service.

5.5. TOOLS DISTRIBUTED WITH PCP

The following table describes usage of various tools, which are distributed with PCP.

Table 5.2. Usage of tools distributed with PCP

Name	Description
pcp	Displays the current status of a Performance Co-Pilot installation.
pcp-atop	Shows the system-level occupation of the most critical hardware resources from the performance point of view: CPU, memory, disk, and network.
pcp-atopsar	Generates a system-level activity report over a variety of system resource utilization. The report is generated from a raw logfile previously recorded using pmlogger or the -w option of pcp-atop.
pcp-dmcache	Displays information about configured Device Mapper Cache targets, such as: device IOPs, cache and metadata device utilization, as well as hit and miss rates and ratios for both reads and writes for each cache device.
pcp-dstat	Displays metrics of one system at a time. To display metrics of multiple systems, use --host option.
pcp-free	Reports on free and used memory in a system.
pcp-htop	Displays all processes running on a system along with their command line arguments in a manner similar to the top command, but allows you to scroll vertically and horizontally as well as interact using a mouse. You can also view processes in a tree format and select and act on multiple processes at once.
pcp-ipcs	Displays information about the inter-process communication (IPC) facilities that the calling process has read access for.
pcp-numastat	Displays NUMA allocation statistics from the kernel memory allocator.
pcp-pidstat	Displays information about individual tasks or processes running on the system such as: CPU percentage, memory and stack usage, scheduling, and priority. Reports live data for the local host by default.

pcp-ss	Displays socket statistics collected by the pmdasockets Performance Metrics Domain Agent (PMDA).
pcp-uptime	Displays how long the system has been running, how many users are currently logged on, and the system load averages for the past 1, 5, and 15 minutes.
pcp-vmstat	Provides a high-level system performance overview every 5 seconds. Displays information about processes, memory, paging, block IO, traps, and CPU activity.
pmchart	Plots performance metrics values available through the facilities of the Performance Co-Pilot.
pmclient	Displays high-level system performance metrics by using the Performance Metrics Application Programming Interface (PMAPI).
pmconfig	Displays the values of configuration parameters.
pmdbg	Displays available Performance Co-Pilot debug control flags and their values.
pmdiff	Compares the average values for every metric in either one or two archives, in a given time window, for changes that are likely to be of interest when searching for performance regressions.
pmdumplog	Displays control, metadata, index, and state information from a Performance Co-Pilot archive file.
pmdumptext	Outputs the values of performance metrics collected live or from a Performance Co-Pilot archive.
pmerr	Displays available Performance Co-Pilot error codes and their corresponding error messages.
pmfind	Finds PCP services on the network.
pmie	An inference engine that periodically evaluates a set of arithmetic, logical, and rule expressions. The metrics are collected either from a live system, or from a Performance Co-Pilot archive file.
pmieconf	Displays or sets configurable pmie variables.

pmiectl	Manages non-primary instances of pmie.
pminfo	Displays information about performance metrics. The metrics are collected either from a live system, or from a Performance Co-Pilot archive file.
pmiostat	Reports I/O statistics for SCSI devices (by default) or device-mapper devices (with the <code>-x dm</code> option).
pmic	Interactively configures active pmlogger instances.
pmlogcheck	Identifies invalid data in a Performance Co-Pilot archive file.
pmlogconf	Creates and modifies a pmlogger configuration file.
pmlogctl	Manages non-primary instances of pmlogger.
pmloglabel	Verifies, modifies, or repairs the label of a Performance Co-Pilot archive file.
pmlogsummary	Calculates statistical information about performance metrics stored in a Performance Co-Pilot archive file.
pmprobe	Determines the availability of performance metrics.
pmrep	Reports on selected, easily customizable, performance metrics values.
pmsocks	Allows access to a Performance Co-Pilot hosts through a firewall.
pmstat	Periodically displays a brief summary of system performance.
pmstore	Modifies the values of performance metrics.
pmtrace	Provides a command line interface to the trace PMDA.
pmval	Displays the current value of a performance metric.

5.6. PCP DEPLOYMENT ARCHITECTURES

Performance Co-Pilot (PCP) supports multiple deployment architectures, based on the scale of the PCP deployment, and offers many options to accomplish advanced setups.

Available scaling deployment setup variants based on the recommended deployment set up by Red Hat, sizing factors, and configuration options include:

Localhost

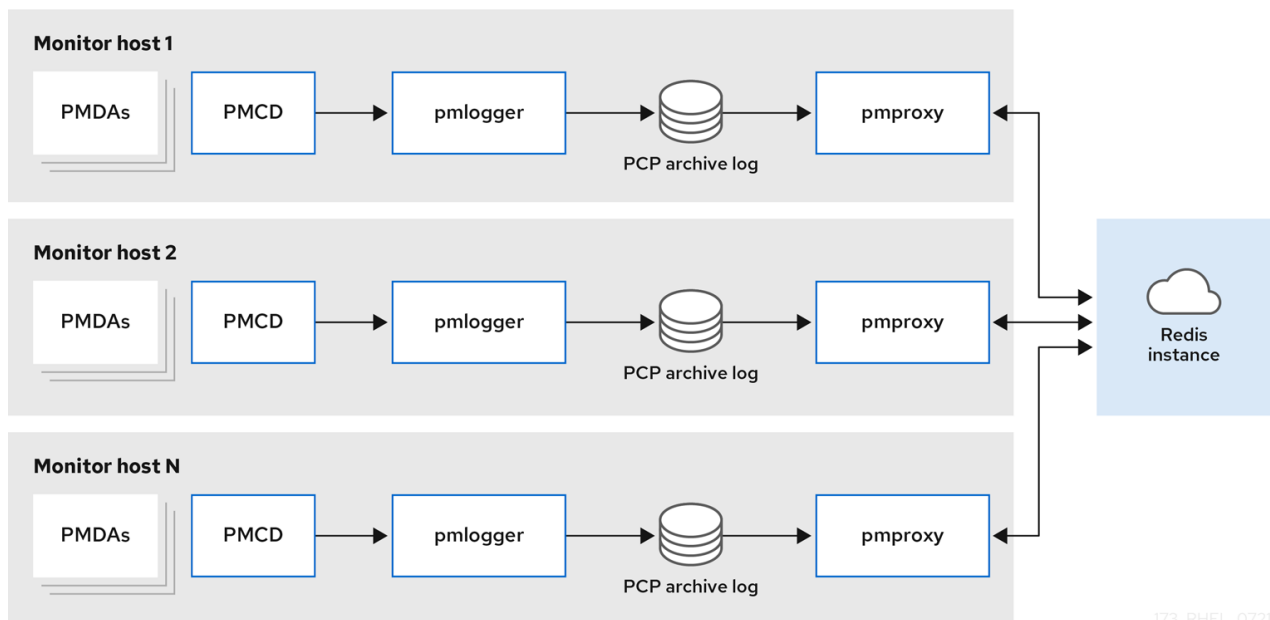
Each service runs locally on the monitored machine. When you start a service without any configuration changes, this is the default deployment. Scaling beyond the individual node is not possible in this case.

By default, the deployment setup for Redis is standalone, localhost. However, Redis can optionally perform in a highly-available and highly scalable clustered fashion, where data is shared across multiple hosts. Another viable option is to deploy a Redis cluster in the cloud, or to utilize a managed Redis cluster from a cloud vendor.

Decentralized

The only difference between localhost and decentralized setup is the centralized Redis service. In this model, the host executes **pmlogger** service on each monitored host and retrieves metrics from a local **pmcd** instance. A local **pmproxy** service then exports the performance metrics to a central Redis instance.

Figure 5.1. Decentralized logging

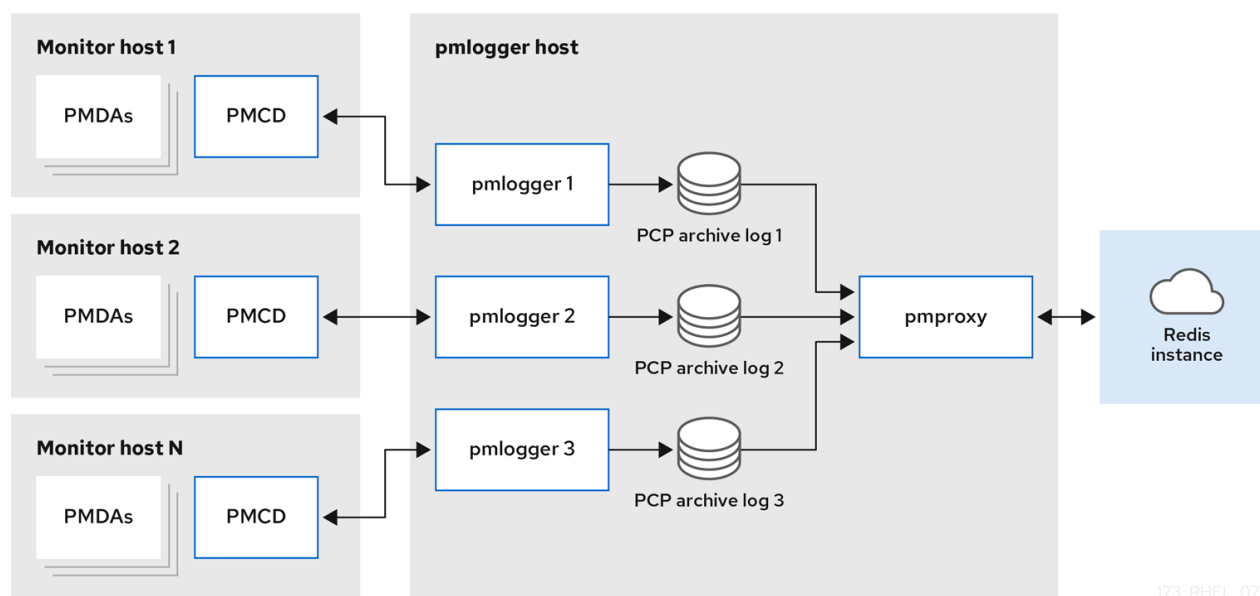


173_RHEL_0721

Centralized logging - pmlogger farm

When the resource usage on the monitored hosts is constrained, another deployment option is a **pmlogger** farm, which is also known as centralized logging. In this setup, a single logger host executes multiple **pmlogger** processes, and each is configured to retrieve performance metrics from a different remote **pmcd** host. The centralized logger host is also configured to execute the **pmproxy** service, which discovers the resulting PCP archives logs and loads the metric data into a Redis instance.

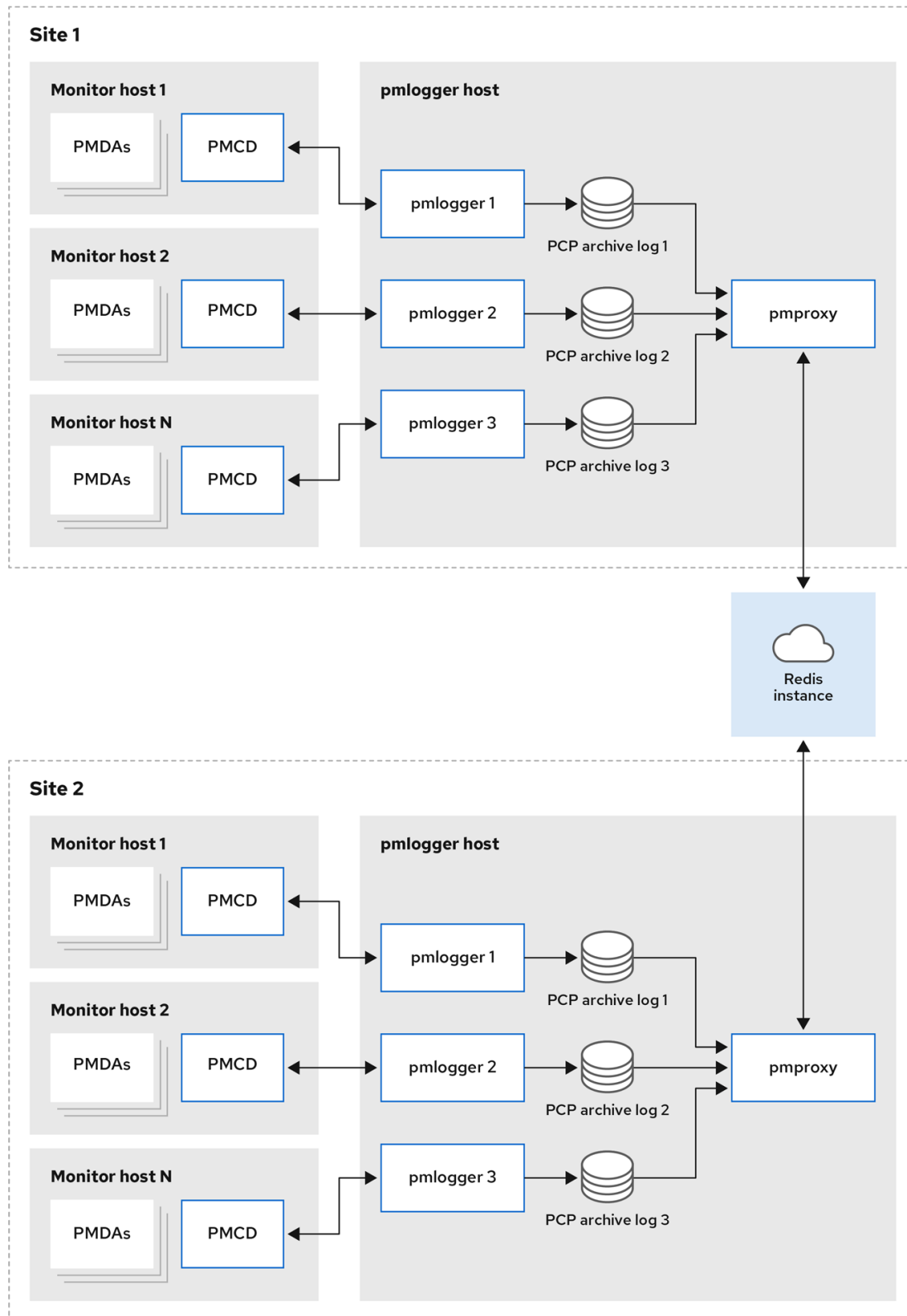
Figure 5.2. Centralized logging - pmlogger farm



Federated - multiple pmlogger farms

For large scale deployments, Red Hat recommends to deploy multiple **pmlogger** farms in a federated fashion. For example, one **pmlogger** farm per rack or data center. Each **pmlogger** farm loads the metrics into a central Redis instance.

Figure 5.3. Federated - multiple pmlogger farms



173_RHEL_0721

**NOTE**

By default, the deployment setup for Redis is standalone, localhost. However, Redis can optionally perform in a highly-available and highly scalable clustered fashion, where data is shared across multiple hosts. Another viable option is to deploy a Redis cluster in the cloud, or to utilize a managed Redis cluster from a cloud vendor.

Additional resources

- **pcp(1)**, **pmlogger(1)**, **pmproxy(1)**, and **pmcd(1)** man pages
- [Recommended deployment architecture](#)

5.7. RECOMMENDED DEPLOYMENT ARCHITECTURE

The following table describes the recommended deployment architectures based on the number of monitored hosts.

Table 5.3. Recommended deployment architecture

Number of hosts (N)	1-10	10-100	100-1000
pmcd servers	N	N	N
pmlogger servers	1 to N	N/10 to N	N/100 to N
pmproxy servers	1 to N	1 to N	N/100 to N
Redis servers	1 to N	1 to N/10	N/100 to N/10
Redis cluster	No	Maybe	Yes
Recommended deployment setup	Localhost, Decentralized, or Centralized logging	Decentralized, Centralized logging, or Federated	Decentralized or Federated

5.8. SIZING FACTORS

The following are the sizing factors required for scaling:

Remote system size

The number of CPUs, disks, network interfaces, and other hardware resources affects the amount of data collected by each **pmlogger** on the centralized logging host.

Logged Metrics

The number and types of logged metrics play an important role. In particular, the **per-process proc.*** metrics require a large amount of disk space, for example, with the standard **pcp-zeroconf** setup, 10s logging interval, 11 MB without proc metrics versus 155 MB with proc metrics – a factor of 10 times more. Additionally, the number of instances for each metric, for example the number of CPUs, block devices, and network interfaces also impacts the required storage capacity.

Logging Interval

The interval how often metrics are logged, affects the storage requirements. The expected daily PCP archive file sizes are written to the **pmlogger.log** file for each **pmlogger** instance. These values are uncompressed estimates. Since PCP archives compress very well, approximately 10:1, the actual long term disk space requirements can be determined for a particular site.

pmlogrewrite

After every PCP upgrade, the **pmlogrewrite** tool is executed and rewrites old archives if there were changes in the metric metadata from the previous version and the new version of PCP. This process duration scales linear with the number of archives stored.

Additional resources

- **pmlogrewrite(1)** and **pmlogger(1)** man pages

5.9. CONFIGURATION OPTIONS FOR PCP SCALING

The following are the configuration options, which are required for scaling:

sysctl and rlimit settings

When archive discovery is enabled, **pmproxy** requires four descriptors for every **pmlogger** that it is monitoring or log-tailing, along with the additional file descriptors for the service logs and **pmproxy** client sockets, if any. Each **pmlogger** process uses about 20 file descriptors for the remote **pmcd** socket, archive files, service logs, and others. In total, this can exceed the default 1024 soft limit on a system running around 200 **pmlogger** processes. The **pmproxy** service in **pcp-5.3.0** and later automatically increases the soft limit to the hard limit. On earlier versions of PCP, tuning is required if a high number of **pmlogger** processes are to be deployed, and this can be accomplished by increasing the soft or hard limits for **pmlogger**. For more information, see [How to set limits \(ulimit\) for services run by systemd](#).

Local Archives

The **pmlogger** service stores metrics of local and remote **pmcds** in the **/var/log/pcp/pmlogger/** directory. To control the logging interval of the local system, update the **/etc/pcp/pmlogger/control.d/configfile** file and add **-t X** in the arguments, where **X** is the logging interval in seconds. To configure which metrics should be logged, execute **pmlogconf /var/lib/pcp/config/pmlogger/config.clienthostname**. This command deploys a configuration file with a default set of metrics, which can optionally be further customized. To specify retention settings, that is when to purge old PCP archives, update the **/etc/sysconfig/pmlogger_timers** file and specify **PMLOGGER_DAILY_PARAMS="-E -k X"**, where **X** is the amount of days to keep PCP archives.

Redis

The **pmproxy** service sends logged metrics from **pmlogger** to a Redis instance. The following are the available two options to specify the retention settings in the **/etc/pcp/pmproxy/pmproxy.conf** configuration file:

- **stream.expire** specifies the duration when stale metrics should be removed, that is metrics which were not updated in a specified amount of time in seconds.
- **stream.maxlen** specifies the maximum number of metric values for one metric per host. This setting should be the retention time divided by the logging interval, for example 20160 for 14 days of retention and 60s logging interval ($60 \times 60 \times 24 \times 14 / 60$)

Additional resources

- **pmproxy(1)**, **pmlogger(1)**, and **sysctl(8)** man pages

5.10. EXAMPLE: ANALYZING THE CENTRALIZED LOGGING DEPLOYMENT

The following results were gathered on a centralized logging setup, also known as **pmlogger** farm deployment, with a default **pcp-zeroconf 5.3.0** installation, where each remote host is an identical container instance running **pmcd** on a server with 64 CPU cores, 376 GB RAM, and one disk attached.

The logging interval is 10s, proc metrics of remote nodes are not included, and the memory values refer to the Resident Set Size (RSS) value.

Table 5.4. Detailed utilization statistics for 10s logging interval

Number of Hosts	10	50
PCP Archives Storage per Day	91 MB	522 MB
pmlogger Memory	160 MB	580 MB
pmlogger Network per Day (In)	2 MB	9 MB
pmproxy Memory	1.4 GB	6.3 GB
Redis Memory per Day	2.6 GB	12 GB

Table 5.5. Used resources depending on monitored hosts for 60s logging interval

Number of Hosts	10	50	100
PCP Archives Storage per Day	20 MB	120 MB	271 MB
pmlogger Memory	104 MB	524 MB	1049 MB
pmlogger Network per Day (In)	0.38 MB	1.75 MB	3.48 MB
pmproxy Memory	2.67 GB	5.5GB	9 GB
Redis Memory per Day	0.54 GB	2.65 GB	5.3 GB



NOTE

The **pmproxy** queues Redis requests and employs Redis pipelining to speed up Redis queries. This can result in high memory usage. For troubleshooting this issue, see [Troubleshooting high memory usage](#).

5.11. EXAMPLE: ANALYZING THE FEDERATED SETUP DEPLOYMENT

The following results were observed on a federated setup, also known as multiple **pmlogger** farms, consisting of three centralized logging (**pmlogger** farm) setups, where each **pmlogger** farm was monitoring 100 remote hosts, that is 300 hosts in total.

This setup of the **pmlogger** farms is identical to the configuration mentioned in the

[Example: Analyzing the centralized logging deployment](#) for 60s logging interval, except that the Redis servers were operating in cluster mode.

Table 5.6. Used resources depending on federated hosts for 60s logging interval

PCP Archives Storage per Day	pmlogger Memory	Network per Day (In/Out)	pmproxy Memory	Redis Memory per Day
277 MB	1058 MB	15.6 MB / 12.3 MB	6-8 GB	5.5 GB

Here, all values are per host. The network bandwidth is higher due to the inter-node communication of the Redis cluster.

5.12. ESTABLISHING SECURE PCP CONNECTIONS

You can configure PCP collector and monitoring components to participate in secure PCP protocol exchanges.

5.12.1. Secure PCP connections

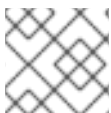
You can establish secure connections between Performance Co-Pilot (PCP) collector and monitoring components. PCP collector components are the parts of PCP that collect and extract performance data from different sources. PCP monitor components are the parts of PCP that display data collected from hosts or archives that have the PCP collector components installed. Establishing secure connections between these components helps prevent unauthorized parties from accessing or modifying the data being collected and monitored.

All connections with the Performance Metrics Collector Daemon (**pmcd**) are made using the TCP/IP based PCP protocol. Protocol proxying and the PCP REST APIs are served by the **pmproxy** daemon - the REST API can be accessed over HTTPS, ensuring a secure connection.

Both the **pmcd** and **pmproxy** daemons are capable of simultaneous TLS and non-TLS communications on a single port. The default port for **pmcd** is 44321 and 44322 for **pmproxy**. This means that you do not have to choose between TLS or non-TLS communications for your PCP collector systems and can use both at the same time.

5.12.2. Configuring secure connections for PCP collector components

All PCP collector systems must have valid certificates in order to participate in secure PCP protocol exchanges.



NOTE

the **pmproxy** daemon operates as both a client and a server from the perspective of TLS.

Prerequisites

- PCP is installed. For more information, see [Installing and enabling PCP](#).
- The private client key is stored in the `/etc/pcp/tls/client.key` file. If you use a different path, adapt the corresponding steps of the procedure.
For details about creating a private key and certificate signing request (CSR), as well as how to request a certificate from a certificate authority (CA), see your CA's documentation.

- The TLS client certificate is stored in the **/etc/pcp/tls/client.crt** file. If you use a different path, adapt the corresponding steps of the procedure.
- The CA certificate is stored in the **/etc/pcp/tls/ca.crt** file. If you use a different path, adapt the corresponding steps of the procedure. Additionally, for the **pmproxy** daemon:
- The private server key is stored in the **/etc/pcp/tls/server.key** file. If you use a different path, adapt the corresponding steps of the procedure
- The TLS server certificate is stored in the **/etc/pcp/tls/server.crt** file. If you use a different path, adapt the corresponding steps of the procedure.

Procedure

1. Update the PCP TLS configuration file on the collector systems to use the CA issued certificates to establish a secure connection:

```
# cat > /etc/pcp/tls.conf << END
tls-ca-cert-file = /etc/pcp/tls/ca.crt
tls-key-file = /etc/pcp/tls/server.key
tls-cert-file = /etc/pcp/tls/server.crt
tls-client-key-file = /etc/pcp/tls/client.key
tls-client-cert-file = /etc/pcp/tls/client.crt
END
```

2. Restart the PCP collector infrastructure:

```
# systemctl restart pmcd.service
# systemctl restart pmproxy.service
```

Verification

- Verify the TLS configuration:

- On the **pmcd** service:

```
# grep 'Info:' /var/log/pcp/pmcd/pmcd.log
[Tue Feb 07 11:47:33] pmcd(6558) Info: OpenSSL 3.0.7 setup
```

- On the **pmproxy** service:

```
# grep 'Info:' /var/log/pcp/pmproxy/pmproxy.log
[Tue Feb 07 11:44:13] pmproxy(6014) Info: OpenSSL 3.0.7 setup
```

5.12.3. Configuring secure connections for PCP monitoring components

Configure your PCP monitoring components to participate in secure PCP protocol exchanges.

Prerequisites

- PCP is installed. For more information, see [Installing and enabling PCP](#).
- The private client key is stored in the **~/.pcp/tls/client.key** file. If you use a different path, adapt the corresponding steps of the procedure.

For details about creating a private key and certificate signing request (CSR), as well as how to request a certificate from a certificate authority (CA), see your CA's documentation.

- The TLS client certificate is stored in the `~/pcp/tls/client.crt` file. If you use a different path, adapt the corresponding steps of the procedure.
- The CA certificate is stored in the `/etc/pcp/tls/ca.crt` file. If you use a different path, adapt the corresponding steps of the procedure.

Procedure

1. Create a TLS configuration file with the following information:

```
$ home=echo ~
$ cat > ~/.pcp/tls.conf << END
tls-ca-cert-file = /etc/pcp/tls/ca.crt
tls-key-file = $home/.pcp/tls/client.key
tls-cert-file = $home/.pcp/tls/client.crt
END
```

2. Establish the secure connection:

```
$ export PCP_SECURE_SOCKETS=enforce
$ export PCP_TLSCONF_PATH=~/.pcp/tls.conf
```

Verification

- Verify the secure connection is configured:

```
$ pminfo --fetch --host pcps://localhost kernel.all.load

kernel.all.load
  inst [1 or "1 minute"] value 1.26
  inst [5 or "5 minute"] value 1.29
  inst [15 or "15 minute"] value 1.28
```

5.13. TROUBLESHOOTING HIGH MEMORY USAGE

The following scenarios can result in high memory usage:

- The **pmproxy** process is busy processing new PCP archives and does not have spare CPU cycles to process Redis requests and responses.
- The Redis node or cluster is overloaded and cannot process incoming requests on time.

The **pmproxy** service daemon uses Redis streams and supports the configuration parameters, which are PCP tuning parameters and affects Redis memory usage and key retention. The `/etc/pcp/pmproxy/pmproxy.conf` file lists the available configuration options for **pmproxy** and the associated APIs.

The following procedure describes how to troubleshoot high memory usage issue.

Prerequisites

1. Install the **pcp-pmda-redis** package:

```
# dnf install pcp-pmda-redis
```

2. Install the redis PMDA:

```
# cd /var/lib/pcp/pmdas/redis && ./Install
```

Procedure

- To troubleshoot high memory usage, execute the following command and observe the **inflight** column:

```
$ pmrep :pmproxy
      backlog inflight reqs/s resp/s wait req err resp err changed throttled
      byte   count  count/s count/s s/s count/s count/s count/s count/s
14:59:08 0      0    N/A    N/A  N/A  N/A  N/A  N/A  N/A  N/A
14:59:09 0      0  2268.9  2268.9  28  0    0    2.0  4.0
14:59:10 0      0    0.0    0.0  0  0    0    0.0  0.0
14:59:11 0      0    0.0    0.0  0  0    0    0.0  0.0
```

This column shows how many Redis requests are in-flight, which means they are queued or sent, and no reply was received so far.

A high number indicates one of the following conditions:

- The **pmproxy** process is busy processing new PCP archives and does not have spare CPU cycles to process Redis requests and responses.
- The Redis node or cluster is overloaded and cannot process incoming requests on time.
- To troubleshoot the high memory usage issue, reduce the number of **pmlogger** processes for this farm, and add another pmlogger farm. Use the federated - multiple pmlogger farms setup. If the Redis node is using 100% CPU for an extended amount of time, move it to a host with better performance or use a clustered Redis setup instead.
- To view the **pmproxy.redis.*** metrics, use the following command:

```
$ pminfo -ftd pmproxy.redis
pmproxy.redis.responses.wait [wait time for responses]
  Data Type: 64-bit unsigned int InDom: PM_INDOM_NULL 0xffffffff
  Semantics: counter Units: microsec
  value 546028367374
pmproxy.redis.responses.error [number of error responses]
  Data Type: 64-bit unsigned int InDom: PM_INDOM_NULL 0xffffffff
  Semantics: counter Units: count
  value 1164
[...]
pmproxy.redis.requests.inflight.bytes [bytes allocated for inflight requests]
  Data Type: 64-bit int InDom: PM_INDOM_NULL 0xffffffff
  Semantics: discrete Units: byte
  value 0
pmproxy.redis.requests.inflight.total [inflight requests]
  Data Type: 64-bit unsigned int InDom: PM_INDOM_NULL 0xffffffff
```

```
Semantics: discrete Units: count
value 0
[...]
```

To view how many Redis requests are inflight, see the **pmproxy.redis.requests.inflight.total** metric and **pmproxy.redis.requests.inflight.bytes** metric to view how many bytes are occupied by all current inflight Redis requests.

In general, the redis request queue would be zero but can build up based on the usage of large pmlogger farms, which limits scalability and can cause high latency for **pmproxy** clients.

- Use the **pminfo** command to view information about performance metrics. For example, to view the **redis.*** metrics, use the following command:

```
$ pminfo -ftd redis
redis.redis_build_id [Build ID]
  Data Type: string InDom: 24.0 0x60000000
  Semantics: discrete Units: count
  inst [0 or "localhost:6379"] value "87e335e57cfa755"
redis.total_commands_processed [Total number of commands processed by the server]
  Data Type: 64-bit unsigned int InDom: 24.0 0x60000000
  Semantics: counter Units: count
  inst [0 or "localhost:6379"] value 595627069
[...]

redis.used_memory_peak [Peak memory consumed by Redis (in bytes)]
  Data Type: 32-bit unsigned int InDom: 24.0 0x60000000
  Semantics: instant Units: count
  inst [0 or "localhost:6379"] value 572234920
[...]
```

To view the peak memory usage, see the **redis.used_memory_peak** metric.

Additional resources

- **pmdaredis(1)**, **pmproxy(1)**, and **pminfo(1)** man pages
- [PCP deployment architectures](#)

CHAPTER 6. LOGGING PERFORMANCE DATA WITH PMLOGGER

With the PCP tool you can log the performance metric values and replay them later. This allows you to perform a retrospective performance analysis.

Using the **pmlogger** tool, you can:

- Create the archived logs of selected metrics on the system
- Specify which metrics are recorded on the system and how often

6.1. MODIFYING THE PMLOGGER CONFIGURATION FILE WITH PMLOGCONF

When the **pmlogger** service is running, PCP logs a default set of metrics on the host.

Use the **pmlogconf** utility to check the default configuration. If the **pmlogger** configuration file does not exist, **pmlogconf** creates it with a default metric values.

Prerequisites

- PCP is installed. For more information, see [Installing and enabling PCP](#).

Procedure

1. Create or modify the **pmlogger** configuration file:

```
# pmlogconf -r /var/lib/pcp/config/pmlogger/config.default
```

2. Follow **pmlogconf** prompts to enable or disable groups of related performance metrics and to control the logging interval for each enabled group.

Additional resources

- **pmlogconf(1)** and **pmlogger(1)** man pages
- [Tools distributed with PCP](#)
- [System services distributed with PCP](#)

6.2. EDITING THE PMLOGGER CONFIGURATION FILE MANUALLY

To create a tailored logging configuration with specific metrics and given intervals, edit the **pmlogger** configuration file manually. The default **pmlogger** configuration file is **/var/lib/pcp/config/pmlogger/config.default**. The configuration file specifies which metrics are logged by the primary logging instance.

In manual configuration, you can:

- Record metrics which are not listed in the automatic configuration.
- Choose custom logging frequencies.

- Add **PMDA** with the application metrics.

Prerequisites

- PCP is installed. For more information, see [Installing and enabling PCP](#).

Procedure

- Open and edit the **/var/lib/pcp/config/pmlogger/config.default** file to add specific metrics:

```
# It is safe to make additions from here on ...
#

log mandatory on every 5 seconds {
    xfs.write
    xfs.write_bytes
    xfs.read
    xfs.read_bytes
}

log mandatory on every 10 seconds {
    xfs.allocs
    xfs.block_map
    xfs.transactions
    xfs.log
}

[access]
disallow * : all;
allow localhost : enquire;
```

Additional resources

- **pmlogger(1)** man page
- [Tools distributed with PCP](#)
- [System services distributed with PCP](#)

6.3. ENABLING THE PMLOGGER SERVICE

The **pmlogger** service must be started and enabled to log the metric values on the local machine.

This procedure describes how to enable the **pmlogger** service.

Prerequisites

- PCP is installed. For more information, see [Installing and enabling PCP](#).

Procedure

- Start and enable the **pmlogger** service:

```
# systemctl start pmlogger  
# systemctl enable pmlogger
```

Verification steps

- Verify if the **pmlogger** service is enabled:

```
# pcp  
  
Performance Co-Pilot configuration on workstation:  
  
platform: Linux workstation 4.18.0-80.el8.x86_64 #1 SMP Wed Mar 13 12:02:46 UTC 2019  
x86_64  
hardware: 12 cpus, 2 disks, 1 node, 36023MB RAM  
timezone: CEST-2  
services: pmcd  
pmcd: Version 4.3.0-1, 8 agents, 1 client  
pmda: root pmcd proc xfs linux mmv kvm jbd2  
pmlogger: primary logger: /var/log/pcp/pmlogger/workstation/20190827.15.54
```

Additional resources

- **pmlogger(1)** man page
- [Tools distributed with PCP](#)
- [System services distributed with PCP](#)
- `/var/lib/pcp/config/pmlogger/config.default` file

6.4. SETTING UP A CLIENT SYSTEM FOR METRICS COLLECTION

This procedure describes how to set up a client system so that a central server can collect metrics from clients running PCP.

Prerequisites

- PCP is installed. For more information, see [Installing and enabling PCP](#).

Procedure

1. Install the **pcp-system-tools** package:

```
# dnf install pcp-system-tools
```

2. Configure an IP address for **pmcd**:

```
# echo "-i 192.168.4.62" >>/etc/pcp/pmcd/pmcd.options
```

Replace `192.168.4.62` with the IP address, the client should listen on.

By default, **pmcd** is listening on the localhost.

3. Configure the firewall to add the public **zone** permanently:

```
# firewall-cmd --permanent --zone=public --add-port=44321/tcp
success

# firewall-cmd --reload
success
```

4. Set an SELinux boolean:

```
# setsebool -P pcp_bind_all_unreserved_ports on
```

5. Enable the **pmcd** and **pmlogger** services:

```
# systemctl enable pmcd pmlogger
# systemctl restart pmcd pmlogger
```

Verification steps

- Verify if the **pmcd** is correctly listening on the configured IP address:

```
# ss -tln | grep 44321
LISTEN 0 5 127.0.0.1:44321 0.0.0.0:* users:(("pmcd",pid=151595,fd=6))
LISTEN 0 5 192.168.4.62:44321 0.0.0.0:* users:(("pmcd",pid=151595,fd=0))
LISTEN 0 5 [::]:44321 [::]:* users:(("pmcd",pid=151595,fd=7))
```

Additional resources

- **pmlogger(1)**, **firewall-cmd(1)**, **ss(8)**, and **setsebool(8)** man pages
- [Tools distributed with PCP](#)
- [System services distributed with PCP](#)
- `/var/lib/pcp/config/pmlogger/config.default` file

6.5. SETTING UP A CENTRAL SERVER TO COLLECT DATA

This procedure describes how to create a central server to collect metrics from clients running PCP.

Prerequisites

- PCP is installed. For more information, see [Installing and enabling PCP](#).
- Client is configured for metrics collection. For more information, see [Setting up a client system for metrics collection](#).

Procedure

1. Install the **pcp-system-tools** package:

```
# dnf install pcp-system-tools
```

2. Create the `/etc/pcp/pmlogger/control.d/remote` file with the following content:

```
# DO NOT REMOVE OR EDIT THE FOLLOWING LINE
$version=1.1

192.168.4.13 n n PCP_ARCHIVE_DIR/rhel7u4a -r -T24h10m -c config.rhel7u4a
192.168.4.14 n n PCP_ARCHIVE_DIR/rhel6u10a -r -T24h10m -c config.rhel6u10a
192.168.4.62 n n PCP_ARCHIVE_DIR/rhel8u1a -r -T24h10m -c config.rhel8u1a
192.168.4.69 n n PCP_ARCHIVE_DIR/rhel9u3a -r -T24h10m -c config.rhel9u3a
```

Replace `192.168.4.13`, `192.168.4.14`, `192.168.4.62` and `192.168.4.69` with the client IP addresses.

3. Enable the **pmcd** and **pmlogger** services:

```
# systemctl enable pmcd pmlogger
# systemctl restart pmcd pmlogger
```

Verification steps

- Ensure that you can access the latest archive file from each directory:

```
# for i in /var/log/pcp/pmlogger/rhel*/*.0; do pmdumplog -L $i; done
Log Label (Log Format Version 2)
Performance metrics from host rhel6u10a.local
  commencing Mon Nov 25 21:55:04.851 2019
  ending    Mon Nov 25 22:06:04.874 2019
Archive timezone: JST-9
PID for pmlogger: 24002
Log Label (Log Format Version 2)
Performance metrics from host rhel7u4a
  commencing Tue Nov 26 06:49:24.954 2019
  ending    Tue Nov 26 07:06:24.979 2019
Archive timezone: CET-1
PID for pmlogger: 10941
[..]
```

The archive files from the `/var/log/pcp/pmlogger/` directory can be used for further analysis and graphing.

Additional resources

- **pmlogger(1)** man page
- [Tools distributed with PCP](#)
- [System services distributed with PCP](#)
- `/var/lib/pcp/config/pmlogger/config.default` file

6.6. REPLAYING THE PCP LOG ARCHIVES WITH PMREP

After recording the metric data, you can replay the PCP log archives. To export the logs to text files and import them into spreadsheets, use PCP utilities such as **pcp2csv**, **pcp2xml**, **pmrep** or **pmlogsummary**.

Using the **pmrep** tool, you can:

- View the log files
- Parse the selected PCP log archive and export the values into an ASCII table
- Extract the entire archive log or only select metric values from the log by specifying individual metrics on the command line

Prerequisites

- PCP is installed. For more information, see [Installing and enabling PCP](#).
- The **pmlogger** service is enabled. For more information, see [Enabling the pmlogger service](#).
- Install the **pcp-system-tools** package:

```
# dnf install pcp-gui
```

Procedure

- Display the data on the metric:

```
$ pmrep --start @3:00am --archive 20211128 --interval 5seconds --samples 10 --output csv
disk.dev.write
Time,"disk.dev.write-sda","disk.dev.write-sdb"
2021-11-28 03:00:00,,
2021-11-28 03:00:05,4.000,5.200
2021-11-28 03:00:10,1.600,7.600
2021-11-28 03:00:15,0.800,7.100
2021-11-28 03:00:20,16.600,8.400
2021-11-28 03:00:25,21.400,7.200
2021-11-28 03:00:30,21.200,6.800
2021-11-28 03:00:35,21.000,27.600
2021-11-28 03:00:40,12.400,33.800
2021-11-28 03:00:45,9.800,20.600
```

The mentioned example displays the data on the **disk.dev.write** metric collected in an archive at a 5 second interval in comma-separated-value format.



NOTE

Replace **20211128** in this example with a filename containing the **pmlogger** archive you want to display data for.

Additional resources

- **pmlogger(1)**, **pmrep(1)**, and **pmlogsummary(1)** man pages
- [Tools distributed with PCP](#)
- [System services distributed with PCP](#)

6.7. ENABLING PCP VERSION 3 ARCHIVES

Performance Co-Pilot (PCP) archives store historical values of PCP metrics recorded from a single host and support retrospective performance analysis. PCP archives contain all the important metric data and metadata needed for offline or offsite analysis. These archives can be read by most PCP client tools or dumped raw by the **pmdumplog** tool.

From PCP 6.0, version 3 archives are supported in addition to version 2 archives. Version 2 archives remain the default and will continue to receive long-term support for backwards compatibility purposes in addition to version 3 archives receiving long-term support from RHEL 9.2 and on.

Using PCP version 3 archives offers the following benefits over version 2:

- Support for instance domain change-deltas
- Y2038-safe timestamps
- Nanosecond-precision timestamps
- Arbitrary timezones support
- 64-bit file offsets used for individual volumes larger than 2GB

Prerequisites

- PCP is installed. For more information, see [Installing and enabling PCP](#).

Procedure

1. Open the **/etc/pcp.conf** file in a text editor of your choice and set the PCP archive version:

```
PCP_ARCHIVE_VERSION=3
```

2. Restart the **pmlogger** service to apply your configuration changes:

```
# systemctl restart pmlogger.service
```

3. Create a new PCP archive log using your new configuration. For more information, see [Logging performance data with pmlogger](#).

Verification

- Verify the version of the archive created with your new configuration:

```
# pmloglabel -l /var/log/pcp/pmlogger/20230208
Log Label (Log Format Version 3)
Performance metrics from host host1
    commencing Wed Feb 08 00:11:09.396 2023
    ending      Thu Feb 07 00:13:54.347 2023
```

Additional resources

- **logarchive(5)** man page
- **pmlogger(1)** man page
- [Logging performance data with pmlogger](#)

CHAPTER 7. MONITORING PERFORMANCE WITH PERFORMANCE CO-PILOT

Performance Co-Pilot (PCP) is a suite of tools, services, and libraries for monitoring, visualizing, storing, and analyzing system-level performance measurements.

As a system administrator, you can monitor the system's performance using the PCP application in Red Hat Enterprise Linux 9.

7.1. MONITORING POSTFIX WITH PMDA-POSTFIX

This procedure describes how to monitor performance metrics of the **postfix** mail server with **pmda-postfix**. It helps to check how many emails are received per second.

Prerequisites

- PCP is installed. For more information, see [Installing and enabling PCP](#).
- The **pmlogger** service is enabled. For more information, see [Enabling the pmlogger service](#).

Procedure

1. Install the following packages:

- a. Install the **pcp-system-tools**:

```
# dnf install pcp-system-tools
```

- b. Install the **pmda-postfix** package to monitor **postfix**:

```
# dnf install pcp-pmda-postfix postfix
```

- c. Install the logging daemon:

```
# dnf install rsyslog
```

- d. Install the mail client for testing:

```
# dnf install mutt
```

2. Enable the **postfix** and **rsyslog** services:

```
# systemctl enable postfix rsyslog  
# systemctl restart postfix rsyslog
```

3. Enable the SELinux boolean, so that **pmda-postfix** can access the required log files:

```
# setsebool -P pcp_read_generic_logs=on
```

4. Install the **PMDA**:

```
# cd /var/lib/pcp/pmdas/postfix/
```

```
# ./Install
```

```
Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Waiting for pmcd to terminate ...
Starting pmcd ...
Check postfix metrics have appeared ... 7 metrics and 58 values
```

Verification steps

- Verify the **pmda-postfix** operation:

```
echo testmail | mutt root
```

- Verify the available metrics:

```
# pminfo postfix

postfix.received
postfix.sent
postfix.queues.incoming
postfix.queues.maildrop
postfix.queues.hold
postfix.queues.deferred
postfix.queues.active
```

Additional resources

- **rsyslogd(8)**, **postfix(1)**, and **setsebool(8)** man pages
- [Tools distributed with PCP](#)
- [System services distributed with PCP](#)
- **/var/lib/pcp/config/pmlogger/config.default** file

7.2. VISUALLY TRACING PCP LOG ARCHIVES WITH THE PCP CHARTS APPLICATION

After recording metric data, you can replay the PCP log archives as graphs. The metrics are sourced from one or more live hosts with alternative options to use metric data from PCP log archives as a source of historical data. To customize the **PCP Charts** application interface to display the data from the performance metrics, you can use line plot, bar graphs, or utilization graphs.

Using the **PCP Charts** application, you can:

- Replay the data in the **PCP Charts** application and use graphs to visualize the retrospective data alongside live data of the system.
- Plot performance metric values into graphs.
- Display multiple charts simultaneously.

Prerequisites

- PCP is installed. For more information, see [Installing and enabling PCP](#).
- Logged performance data with the **pmlogger**. For more information, see [Logging performance data with pmlogger](#).
- Install the **pcp-gui** package:

```
# dnf install pcp-gui
```

Procedure

1. Launch the **PCP Charts** application from the command line:

```
# pmchart
```

Figure 7.1. PCP Charts application



The **pmtime** server settings are located at the bottom. The **start** and **pause** button allows you to control:

- The interval in which PCP polls the metric data
 - The date and time for the metrics of historical data
2. Click **File** and then **New Chart** to select metric from both the local machine and remote machines by specifying their host name or address. Advanced configuration options include the ability to manually set the axis values for the chart, and to manually choose the color of the plots.
 3. Record the views created in the **PCP Charts** application:
Following are the options to take images or record the views created in the **PCP Charts** application:
 - Click **File** and then **Export** to save an image of the current view.

- Click **Record** and then **Start** to start a recording. Click **Record** and then **Stop** to stop the recording. After stopping the recording, the recorded metrics are archived to be viewed later.
4. Optional: In the **PCP Charts** application, the main configuration file, known as the **view**, allows the metadata associated with one or more charts to be saved. This metadata describes all chart aspects, including the metrics used and the chart columns. Save the custom **view** configuration by clicking **File** and then **Save View**, and load the **view** configuration later.
- The following example of the **PCP Charts** application view configuration file describes a stacking chart graph showing the total number of bytes read and written to the given XFS file system **loop1**:

```
#kmchart
version 1

chart title "Filesystem Throughput /loop1" style stacking antialiasing off
plot legend "Read rate" metric xfs.read_bytes instance "loop1"
plot legend "Write rate" metric xfs.write_bytes instance "loop1"
```

Additional resources

- **pmchart(1)** and **pmtime(1)** man pages
- [Tools distributed with PCP](#)

7.3. COLLECTING DATA FROM SQL SERVER USING PCP

The SQL Server agent is available in Performance Co-Pilot (PCP), which helps you to monitor and analyze database performance issues.

This procedure describes how to collect data for Microsoft SQL Server via **pcp** on your system.

Prerequisites

- You have installed Microsoft SQL Server for Red Hat Enterprise Linux and established a 'trusted' connection to an SQL server.
- You have installed the Microsoft ODBC driver for SQL Server for Red Hat Enterprise Linux.

Procedure

1. Install PCP:

```
# dnf install pcp-zeroconf
```

2. Install packages required for the **pyodbc** driver:

```
# dnf install python3-pyodbc
```

3. Install the **mssql** agent:

- a. Install the Microsoft SQL Server domain agent for PCP:

```
# dnf install pcp-pmda-mssql
```

- b. Edit the `/etc/pcp/mssql/mssql.conf` file to configure the SQL server account's username and password for the **mssql** agent. Ensure that the account you configure has access rights to performance data.

```
username: user_name
password: user_password
```

Replace *user_name* with the SQL Server account and *user_password* with the SQL Server user password for this account.

4. Install the agent:

```
# cd /var/lib/pcp/pmdas/mssql
# ./Install
Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Check mssql metrics have appeared ... 168 metrics and 598 values
[...]
```

Verification steps

- Using the **pcp** command, verify if the SQL Server PMDA (**mssql**) is loaded and running:

```
$ pcp
Performance Co-Pilot configuration on rhel.local:

platform: Linux rhel.local 4.18.0-167.el8.x86_64 #1 SMP Sun Dec 15 01:24:23 UTC 2019
x86_64
hardware: 2 cpus, 1 disk, 1 node, 2770MB RAM
timezone: PDT+7
services: pmcd pmproxy
          pmcd: Version 5.0.2-1, 12 agents, 4 clients
          pmda: root pmcd proc pmproxy xfs linux nfsclient mmv kvm mssql
                jbd2 dm
pmlogger: primary logger: /var/log/pcp/pmlogger/rhel.local/20200326.16.31
pmie: primary engine: /var/log/pcp/pmie/rhel.local/pmie.log
```

- View the complete list of metrics that PCP can collect from the SQL Server:

```
# pminfo mssql
```

- After viewing the list of metrics, you can report the rate of transactions. For example, to report on the overall transaction count per second, over a five second time window:

```
# pmval -t 1 -T 5 mssql.databases.transactions
```

- View the graphical chart of these metrics on your system by using the **pmchart** command. For more information, see [Visually tracing PCP log archives with the PCP Charts application](#).

Additional resources

- **pcp(1)**, **pminfo(1)**, **pmval(1)**, **pmchart(1)**, and **pmdamssql(1)** man pages

- [Performance Co-Pilot for Microsoft SQL Server with RHEL 8.2 Red Hat Developers Blog post](#)

7.4. GENERATING PCP ARCHIVES FROM SADC ARCHIVES

You can use the **sadf** tool provided by the **sysstat** package to generate PCP archives from native **sadc** archives.

Prerequisites

- A **sadc** archive has been created:

```
# /usr/lib64/sa/sadc 1 5 -
```

In this example, **sadc** is sampling system data 1 time in a 5 second interval. The outfile is specified as **-** which results in **sadc** writing the data to the standard system activity daily data file. This file is named saDD and is located in the `/var/log/sa` directory by default.

Procedure

- Generate a PCP archive from a **sadc** archive:

```
# sadf -l -O pcparchive=/tmp/recording -2
```

In this example, using the **-2** option results in **sadf** generating a PCP archive from a **sadc** archive recorded 2 days ago.

Verification steps

You can use PCP commands to inspect and analyze the PCP archive generated from a **sadc** archive as you would a native PCP archive. For example:

- To show a list of metrics in the PCP archive generated from an **sadc** archive, run:

```
$ pminfo --archive /tmp/recording
Disk.dev.avactive
Disk.dev.read
Disk.dev.write
Disk.dev.blkread
[...]
```

- To show the timespace of the archive and hostname of the PCP archive, run:

```
$ pmdumplog --label /tmp/recording
Log Label (Log Format Version 2)
Performance metrics from host shard
    commencing Tue Jul 20 00:10:30.642477 2021
    ending    Wed Jul 21 00:10:30.222176 2021
```

- To plot performance metrics values into graphs, run:

```
$ pmchart --archive /tmp/recording
```


CHAPTER 8. PERFORMANCE ANALYSIS OF XFS WITH PCP

The XFS PMDA ships as part of the **pcp** package and is enabled by default during the installation. It is used to gather performance metric data of XFS file systems in Performance Co-Pilot (PCP).

You can use PCP to analyze XFS file system's performance.

8.1. INSTALLING XFS PMDA MANUALLY

If the XFS PMDA is not listed in the **pcp** configuration output, install the PMDA agent manually.

This procedure describes how to manually install the PMDA agent.

Prerequisites

- PCP is installed. For more information, see [Installing and enabling PCP](#).

Procedure

1. Navigate to the xfs directory:

```
# cd /var/lib/pcp/pmdas/xfs/
```

2. Install the XFS PMDA manually:

```
xfs]# ./Install
Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Check xfs metrics have appeared ... 387 metrics and 387 values
```

Verification steps

- Verify that the **pmcd** process is running on the host and the XFS PMDA is listed as enabled in the configuration:

```
# pcp

Performance Co-Pilot configuration on workstation:

platform: Linux workstation 4.18.0-80.el8.x86_64 #1 SMP Wed Mar 13 12:02:46 UTC 2019
x86_64
hardware: 12 cpus, 2 disks, 1 node, 36023MB RAM
timezone: CEST-2
services: pmcd
pmcd: Version 4.3.0-1, 8 agents
pmda: root pmcd proc xfs linux mmv kvm jbd2
```

Additional resources

- **pmcd(1)** man page
- [Tools distributed with PCP](#)

8.2. EXAMINING XFS PERFORMANCE METRICS WITH PMINFO

PCP enables XFS PMDA to allow the reporting of certain XFS metrics per each of the mounted XFS file systems. This makes it easier to pinpoint specific mounted file system issues and evaluate performance.

The **pminfo** command provides per-device XFS metrics for each mounted XFS file system.

This procedure displays a list of all available metrics provided by the XFS PMDA.

Prerequisites

- PCP is installed. For more information, see [Installing and enabling PCP](#).

Procedure

- Display the list of all available metrics provided by the XFS PMDA:

```
# pminfo xfs
```

- Display information for the individual metrics. The following examples examine specific XFS **read** and **write** metrics using the **pminfo** tool:

- Display a short description of the **xfs.write_bytes** metric:

```
# pminfo --oneline xfs.write_bytes
```

```
xfs.write_bytes [number of bytes written in XFS file system write operations]
```

- Display a long description of the **xfs.read_bytes** metric:

```
# pminfo --helptext xfs.read_bytes
```

```
xfs.read_bytes
```

```
Help:
```

```
This is the number of bytes read via read(2) system calls to files in
XFS file systems. It can be used in conjunction with the read_calls
count to calculate the average size of the read operations to file in
XFS file systems.
```

- Obtain the current performance value of the **xfs.read_bytes** metric:

```
# pminfo --fetch xfs.read_bytes
```

```
xfs.read_bytes
```

```
value 4891346238
```

- Obtain per-device XFS metrics with **pminfo**:

```
# pminfo --fetch --oneline xfs.perdev.read xfs.perdev.write
```

```
xfs.perdev.read [number of XFS file system read operations]
```

```
inst [0 or "loop1"] value 0
```

```
inst [0 or "loop2"] value 0
```

```
xfs.perdev.write [number of XFS file system write operations]
inst [0 or "loop1"] value 86
inst [0 or "loop2"] value 0
```

Additional resources

- **pminfo(1)** man page
- [PCP metric groups for XFS](#)
- [Per-device PCP metric groups for XFS](#)

8.3. RESETTING XFS PERFORMANCE METRICS WITH PMSTORE

With PCP, you can modify the values of certain metrics, especially if the metric acts as a control variable, such as the **xfs.control.reset** metric. To modify a metric value, use the **pmstore** tool.

This procedure describes how to reset XFS metrics using the **pmstore** tool.

Prerequisites

- PCP is installed. For more information, see [Installing and enabling PCP](#).

Procedure

1. Display the value of a metric:

```
$ pminfo -f xfs.write

xfs.write
value 325262
```

2. Reset all the XFS metrics:

```
# pmstore xfs.control.reset 1

xfs.control.reset old value=0 new value=1
```

Verification steps

- View the information after resetting the metric:

```
$ pminfo --fetch xfs.write

xfs.write
value 0
```

Additional resources

- **pmstore(1)** and **pminfo(1)** man pages
- [Tools distributed with PCP](#)

- [PCP metric groups for XFS](#)

8.4. PCP METRIC GROUPS FOR XFS

The following table describes the available PCP metric groups for XFS.

Table 8.1. Metric groups for XFS

Metric Group	Metrics provided
xfs.*	General XFS metrics including the read and write operation counts, read and write byte counts. Along with counters for the number of times inodes are flushed, clustered and number of failure to cluster.
xfs.allocs.* xfs.alloc_btree.*	Range of metrics regarding the allocation of objects in the file system, these include number of extent and block creations/frees. Allocation tree lookup and compares along with extend record creation and deletion from the btree.
xfs.block_map.* xfs.bmap_btree.*	Metrics include the number of block map read/write and block deletions, extent list operations for insertion, deletions and lookups. Also operations counters for compares, lookups, insertions and deletion operations from the blockmap.
xfs.dir_ops.*	Counters for directory operations on XFS file systems for creation, entry deletions, count of "getdent" operations.
xfs.transactions.*	Counters for the number of meta-data transactions, these include the count for the number of synchronous and asynchronous transactions along with the number of empty transactions.
xfs.inode_ops.*	Counters for the number of times that the operating system looked for an XFS inode in the inode cache with different outcomes. These count cache hits, cache misses, and so on.
xfs.log.* xfs.log_tail.*	Counters for the number of log buffer writes over XFS file systems includes the number of blocks written to disk. Metrics also for the number of log flushes and pinning.
xfs.xstrat.*	Counts for the number of bytes of file data flushed out by the XFS flush daemon along with counters for number of buffers flushed to contiguous and non-contiguous space on disk.

xfs.attr.*	Counts for the number of attribute get, set, remove and list operations over all XFS file systems.
xfs.quota.*	Metrics for quota operation over XFS file systems, these include counters for number of quota reclaims, quota cache misses, cache hits and quota data reclaims.
xfs.buffer.*	Range of metrics regarding XFS buffer objects. Counters include the number of requested buffer calls, successful buffer locks, waited buffer locks, miss_locks, miss_retries and buffer hits when looking up pages.
xfs.btree.*	Metrics regarding the operations of the XFS btree.
xfs.control.reset	Configuration metrics which are used to reset the metric counters for the XFS stats. Control metrics are toggled by means of the pmstore tool.

8.5. PER-DEVICE PCP METRIC GROUPS FOR XFS

The following table describes the available per-device PCP metric group for XFS.

Table 8.2. Per-device PCP metric groups for XFS

Metric Group	Metrics provided
xfs.perdev.*	General XFS metrics including the read and write operation counts, read and write byte counts. Along with counters for the number of times inodes are flushed, clustered and number of failure to cluster.
xfs.perdev.allocs.* xfs.perdev.alloc_btree.*	Range of metrics regarding the allocation of objects in the file system, these include number of extent and block creations/frees. Allocation tree lookup and compares along with extend record creation and deletion from the btree.
xfs.perdev.block_map.* xfs.perdev.bmap_btree.*	Metrics include the number of block map read/write and block deletions, extent list operations for insertion, deletions and lookups. Also operations counters for compares, lookups, insertions and deletion operations from the blockmap.
xfs.perdev.dir_ops.*	Counters for directory operations of XFS file systems for creation, entry deletions, count of "getdent" operations.

xfs.perdev.transactions.*	Counters for the number of meta-data transactions, these include the count for the number of synchronous and asynchronous transactions along with the number of empty transactions.
xfs.perdev.inode_ops.*	Counters for the number of times that the operating system looked for an XFS inode in the inode cache with different outcomes. These count cache hits, cache misses, and so on.
xfs.perdev.log.* xfs.perdev.log_tail.*	Counters for the number of log buffer writes over XFS filesystems includes the number of blocks written to disk. Metrics also for the number of log flushes and pinning.
xfs.perdev.xstrat.*	Counts for the number of bytes of file data flushed out by the XFS flush daemon along with counters for number of buffers flushed to contiguous and non-contiguous space on disk.
xfs.perdev.attr.*	Counts for the number of attribute get, set, remove and list operations over all XFS file systems.
xfs.perdev.quota.*	Metrics for quota operation over XFS file systems, these include counters for number of quota reclaims, quota cache misses, cache hits and quota data reclaims.
xfs.perdev.buffer.*	Range of metrics regarding XFS buffer objects. Counters include the number of requested buffer calls, successful buffer locks, waited buffer locks, miss_locks, miss_retries and buffer hits when looking up pages.
xfs.perdev.btree.*	Metrics regarding the operations of the XFS btree.

CHAPTER 9. SETTING UP GRAPHICAL REPRESENTATION OF PCP METRICS

Using a combination of **pcp**, **grafana**, **pcp redis**, **pcp bpfftrace**, and **pcp vector** provides graphical representation of the live data or data collected by Performance Co-Pilot (PCP).

9.1. SETTING UP PCP WITH PCP-ZEROCONF

This procedure describes how to set up PCP on a system with the **pcp-zeroconf** package. Once the **pcp-zeroconf** package is installed, the system records the default set of metrics into archived files.

Procedure

- Install the **pcp-zeroconf** package:

```
# dnf install pcp-zeroconf
```

Verification steps

- Ensure that the **pmlogger** service is active, and starts archiving the metrics:

```
# pcp | grep pmlogger
pmlogger: primary logger: /var/log/pcp/pmlogger/localhost.localdomain/20200401.00.12
```

Additional resources

- **pmlogger** man page
- [Monitoring performance with Performance Co-Pilot](#)

9.2. SETTING UP A GRAFANA-SERVER

Grafana generates graphs that are accessible from a browser. The **grafana-server** is a back-end server for the Grafana dashboard. It listens, by default, on all interfaces, and provides web services accessed through the web browser. The **grafana-pcp** plugin interacts with the **pmproxy** protocol in the backend.

This procedure describes how to set up a **grafana-server**.

Prerequisites

- PCP is configured. For more information, see [Setting up PCP with pcp-zeroconf](#).

Procedure

1. Install the following packages:

```
# dnf install grafana grafana-pcp
```

2. Restart and enable the following service:

```
# systemctl restart grafana-server
# systemctl enable grafana-server
```

-
- 3. Open the server's firewall for network traffic to the Grafana service.

```
# firewall-cmd --permanent --add-service=grafana
success

# firewall-cmd --reload
success
```

Verification steps

- Ensure that the **grafana-server** is listening and responding to requests:

```
# ss -ntlp | grep 3000
LISTEN 0 128 *:3000 *:~ users:(("grafana-server",pid=19522,fd=7))
```

- Ensure that the **grafana-pcp** plugin is installed:

```
# grafana-cli plugins ls | grep performancecopilot-pcp-app

performancecopilot-pcp-app @ 3.1.0
```

Additional resources

- **pmproxy(1)** and **grafana-server** man pages

9.3. ACCESSING THE GRAFANA WEB UI

This procedure describes how to access the Grafana web interface.

Using the Grafana web interface, you can:

- add PCP Redis, PCP bpfttrace, and PCP Vector data sources
- create dashboard
- view an overview of any useful metrics
- create alerts in PCP Redis

Prerequisites

1. PCP is configured. For more information, see [Setting up PCP with pcp-zeroconf](#).
2. The **grafana-server** is configured. For more information, see [Setting up a grafana-server](#).

Procedure

1. On the client system, open a browser and access the **grafana-server** on port **3000**, using `http://192.0.2.0:3000` link.
Replace `192.0.2.0` with your machine IP.
2. For the first login, enter **admin** in both the **Email or username** and **Password** field.

Grafana prompts to set a **New password** to create a secured account. If you want to set it later, click **Skip**.



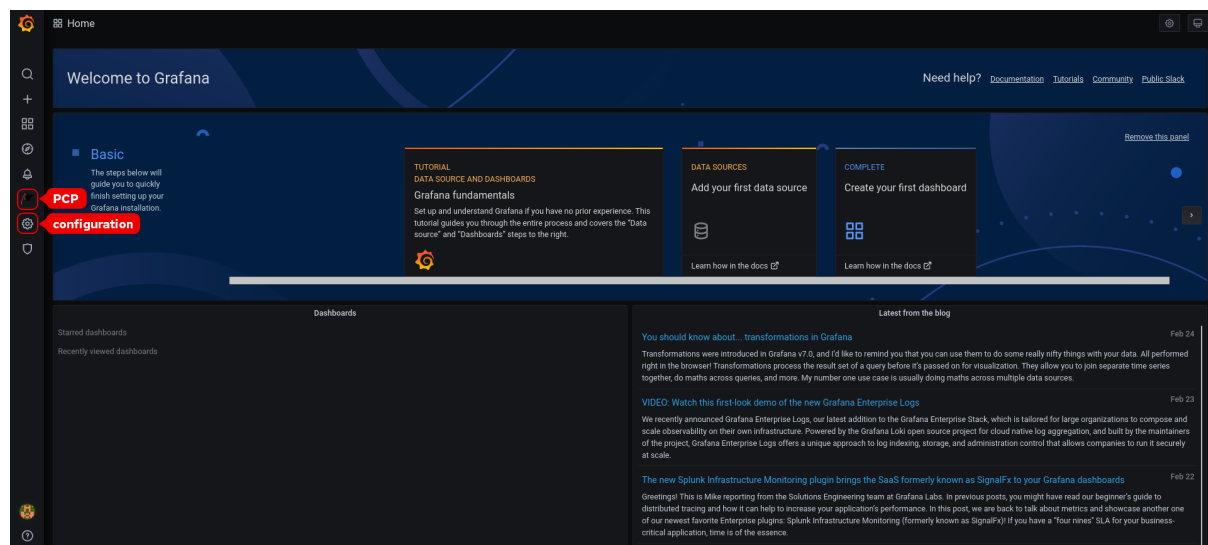


3. From the menu, hover over the  **Configuration** icon and then click **Plugins**.
4. In the **Plugins** tab, type performance co-pilot in the **Search by name or type** text box and then click **Performance Co-Pilot (PCP)** plugin.
5. In the **Plugins / Performance Co-Pilot** pane, click **Enable**.
6. Click Grafana  icon. The Grafana **Home** page is displayed.

Figure 9.1. Home Dashboard



NOTE

The top corner of the screen has a similar  icon, but it controls the general **Dashboard settings**.

7. In the Grafana **Home** page, click **Add your first data source** to add PCP Redis, PCP bpfftrace, and PCP Vector data sources. For more information about adding data source, see:
 - To add pcp redis data source, view default dashboard, create a panel, and an alert rule, see [Creating panels and alert in PCP Redis data source](#).
 - To add pcp bpfftrace data source and view the default dashboard, see [Viewing the PCP bpfftrace System Analysis dashboard](#).
 - To add pcp vector data source, view the default dashboard, and to view the vector checklist, see [Viewing the PCP Vector Checklist](#).
8. Optional: From the menu, hover over the **admin** profile  icon to change the **Preferences** including **Edit Profile**, **Change Password**, or to **Sign out**.

- **grafana-cli** and **grafana-server** man pages

9.4. CONFIGURING SECURE CONNECTIONS FOR GRAFANA

You can establish secure connections between Grafana and Performance Co-Pilot (PCP) components. Establishing secure connections between these components helps prevent unauthorized parties from accessing or modifying the data being collected and monitored.

Prerequisites

- PCP is installed. For more information, see [Installing and enabling PCP](#).
- The **grafana-server** is configured. For more information, see [Setting up a grafana-server](#).
- The private client key is stored in the **/etc/grafana/grafana.key** file. If you use a different path, modify the path in the corresponding steps of the procedure.
For details about creating a private key and certificate signing request (CSR), as well as how to request a certificate from a certificate authority (CA), see your CA's documentation.
- The TLS client certificate is stored in the **/etc/grafana/grafana.crt** file. If you use a different path, modify the path in the corresponding steps of the procedure.

Procedure

1. As a root user, open the **/etc/grafana/grana.ini** file and adjust the following options in the **[server]** section to reflect the following:

```
protocol = https
cert_key = /etc/grafana/grafana.key
cert_file = /etc/grafana/grafana.crt
```


2. Ensure grafana can access the certificates:

```
# su grafana -s /bin/bash -c \
'ls -l /etc/grafana/grafana.crt /etc/grafana/grafana.key'
/etc/grafana/grafana.crt
/etc/grafana/grafana.key
```

3. Restart and enable the Grafana service to apply the configuration changes:

```
# systemctl restart grafana-server
# systemctl enable grafana-server
```

Verification

1. On the client system, open a browser and access the **grafana-server** machine on port 3000, using the `https://192.0.2.0:3000` link. Replace 192.0.2.0 with your machine IP.
2. Confirm the  lock icon is displayed beside the address bar.



NOTE

If the protocol is set to **http** and an HTTPS connection is attempted, you will receive a **ERR_SSL_PROTOCOL_ERROR** error. If the protocol is set to **https** and an HTTP connection is attempted, the Grafana server responds with a “Client sent an HTTP request to an HTTPS server” message.

9.5. CONFIGURING PCP REDIS

Use the PCP Redis data source to:

- View data archives
- Query time series using pmseries language
- Analyze data across multiple hosts

Prerequisites

1. PCP is configured. For more information, see [Setting up PCP with pcp-zeroconf](#).
2. The **grafana-server** is configured. For more information, see [Setting up a grafana-server](#).

Procedure

1. Install the **redis** package:

```
# dnf install redis
```

2. Start and enable the following services:

```
# systemctl start pmproxy redis
# systemctl enable pmproxy redis
```

3. Mail transfer agent, for example, **sendmail** or **postfix** is installed and configured.
4. Ensure that the **allow_loading_unsigned_plugins** parameter is set to PCP Redis database in the **grafana.ini** file:

```
# vi /etc/grafana/grafana.ini

allow_loading_unsigned_plugins = pcp-redis-datasource
```

5. Restart the **grafana-server**:

```
# systemctl restart grafana-server
```

Verification steps

- Ensure that the **pmproxy** and **redis** are working:

```
# pmseries disk.dev.read
2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
```

This command does not return any data if the **redis** package is not installed.

Additional resources

- **pmseries(1)** man page

9.6. CONFIGURING SECURE CONNECTIONS FOR PCP REDIS

You can establish secure connections between performance co-pilot (PCP), Grafana, and PCP redis. Establishing secure connections between these components helps prevent unauthorized parties from accessing or modifying the data being collected and monitored.

Prerequisites

- PCP is installed. For more information, see [Installing and enabling PCP](#).
- The **grafana-server** is configured. For more information, see [Setting up a grafana-server](#).
- PCP redis is installed. For more information, see [Configuring PCP Redis](#).
- The private client key is stored in the **/etc/redis/client.key** file. If you use a different path, modify the path in the corresponding steps of the procedure.
For details about creating a private key and certificate signing request (CSR), as well as how to request a certificate from a certificate authority (CA), see your CA's documentation.
- The TLS client certificate is stored in the **/etc/redis/client.crt** file. If you use a different path, modify the path in the corresponding steps of the procedure.
- The TLS server key is stored in the **/etc/redis/redis.key** file. If you use a different path, modify the path in the corresponding steps of the procedure.
- The TLS server certificate is stored in the **/etc/redis/redis.crt** file. If you use a different path, modify the path in the corresponding steps of the procedure.
- The CA certificate is stored in the **/etc/redis/ca.crt** file. If you use a different path, modify the path in the corresponding steps of the procedure.

Additionally, for the **pmproxy** daemon:

- The private server key is stored in the **/etc/pcp/tls/server.key** file. If you use a different path, modify the path in the corresponding steps of the procedure.

Procedure

1. As a root user, open the **/etc/redis/redis.conf** file and adjust the TLS/SSL options to reflect the following properties:

```
port 0
tls-port 6379
tls-cert-file /etc/redis/redis.crt
tls-key-file /etc/redis/redis.key
tls-client-key-file /etc/redis/client.key
tls-client-cert-file /etc/redis/client.crt
tls-ca-cert-file /etc/redis/ca.crt
```

2. Ensure **redis** can access the TLS certificates:

```
# su redis -s /bin/bash -c \
'ls -l /etc/redis/ca.crt /etc/redis/redis.key /etc/redis/redis.crt'
/etc/redis/ca.crt
/etc/redis/redis.crt
/etc/redis/redis.key
```

3. Restart the **redis** server to apply the configuration changes:

```
# systemctl restart redis
```

Verification

- Confirm the TLS configuration works:

```
# redis-cli --tls --cert /etc/redis/client.crt \
--key /etc/redis/client.key \
--cacert /etc/redis/ca.crt <<< "PING"
PONG
```

Unsuccessful TLS configuration might result in the following error message:

```
Could not negotiate a TLS connection: Invalid CA Certificate File/Directory
```

9.7. CREATING PANELS AND ALERT IN PCP REDIS DATA SOURCE

After adding the PCP Redis data source, you can view the dashboard with an overview of useful metrics, add a query to visualize the load graph, and create alerts that help you to view the system issues after they occur.

Prerequisites

1. The PCP Redis is configured. For more information, see [Configuring PCP Redis](#).
2. The **grafana-server** is accessible. For more information, see [Accessing the Grafana web UI](#).

Procedure

1. Log into the Grafana web UI.
2. In the Grafana **Home** page, click **Add your first data source**
3. In the **Add data source** pane, type **redis** in the **Filter by name or type** text box and then click **PCP Redis**.
4. In the **Data Sources / PCP Redis** pane, perform the following:
 - a. Add **http://localhost:44322** in the **URL** field and then click **Save & Test**.
 - b. Click **Dashboards tab** → **Import** → **PCP Redis: Host Overview** to see a dashboard with an overview of any useful metrics.

Figure 9.2. PCP Redis: Host Overview

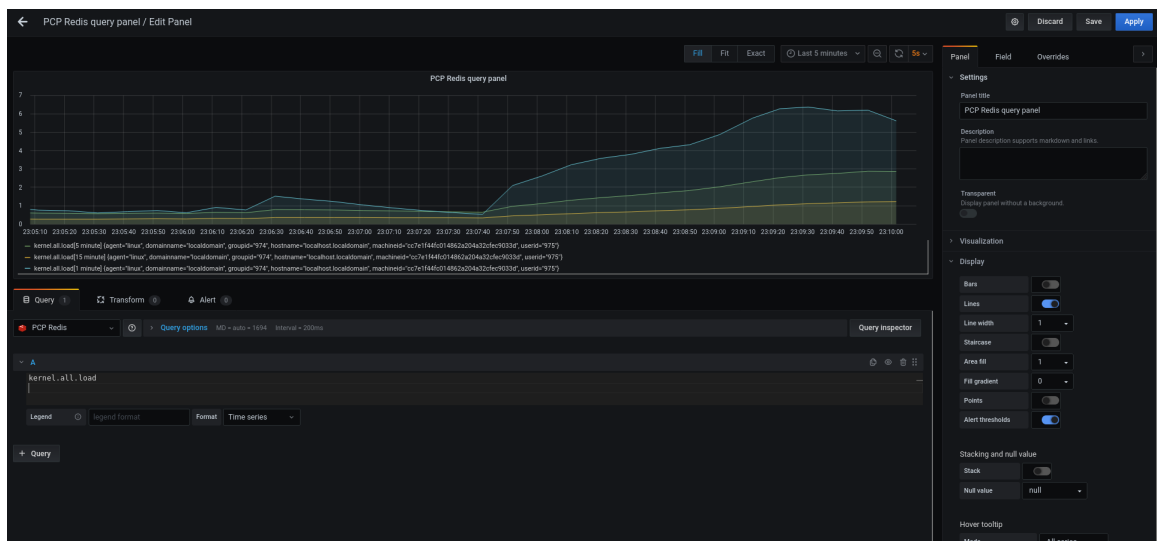


5. Add a new panel:



- From the menu, hover over the **icon** to add a panel.
- In the **Query** tab, select the **PCP Redis** from the query list instead of the selected **default** option and in the text field of **A**, enter metric, for example, **kernel.all.load** to visualize the kernel load graph.
- Optional: Add **Panel title** and **Description**, and update other options from the **Settings**.
- Click **Save** to apply changes and save the dashboard. Add **Dashboard name**.
- Click **Apply** to apply changes and go back to the dashboard.

Figure 9.3. PCP Redis query panel



6. Create an alert rule:

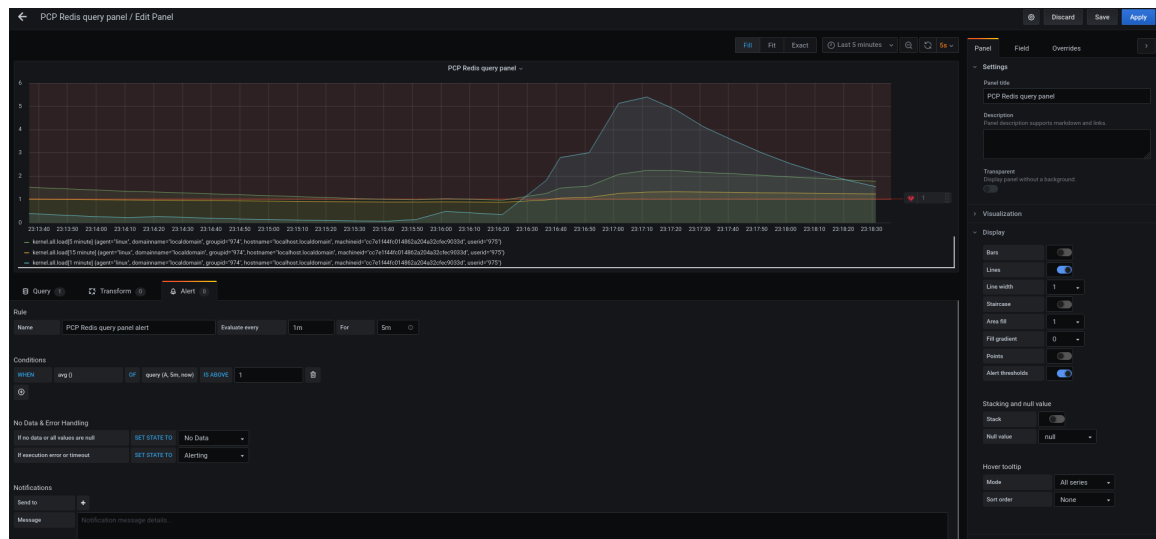
- a. In the **PCP Redis query panel**, click **Alert**.



Alert and then click **Create**

- b. Edit the **Name**, **Evaluate query**, and **For** fields from the **Rule**, and specify the **Conditions** for your alert.
- c. Click **Save** to apply changes and save the dashboard. Click **Apply** to apply changes and go back to the dashboard.

Figure 9.4. Creating alerts in the PCP Redis panel



- d. Optional: In the same panel, scroll down and click **Delete** icon to delete the created rule.



- e. Optional: From the menu, click **Alerting** icon to view the created alert rules with different alert statuses, to edit the alert rule, or to pause the existing rule from the **Alert Rules** tab.

To add a notification channel for the created alert rule to receive an alert notification from Grafana, see [Adding notification channels for alerts](#).

9.8. ADDING NOTIFICATION CHANNELS FOR ALERTS

By adding notification channels, you can receive an alert notification from Grafana whenever the alert rule conditions are met and the system needs further monitoring.

You can receive these alerts after selecting any one type from the supported list of notifiers, which includes **DingDing**, **Discord**, **Email**, **Google Hangouts Chat**, **HipChat**, **Kafka REST Proxy**, **LINE**, **Microsoft Teams**, **OpsGenie**, **PagerDuty**, **Prometheus Alertmanager**, **Pushover**, **Sensu**, **Slack**, **Telegram**, **Threema Gateway**, **VictorOps**, and **webhook**.

Prerequisites

1. The **grafana-server** is accessible. For more information, see [Accessing the Grafana web UI](#).
2. An alert rule is created. For more information, see [Creating panels and alert in PCP Redis data source](#).
3. Configure SMTP and add a valid sender's email address in the **grafana/grafana.ini** file:

```
# vi /etc/grafana/grafana.ini


[smtp]
enabled = true
from_address = abc@gmail.com
```

Replace *abc@gmail.com* by a valid email address.

4. Restart **grafana-server**

```
# systemctl restart grafana-server.service
```

Procedure

1. From the menu, hover over the  **Alerting icon** → click **Contact Points** → **New contact point**.
2. In the **New contact point** details view, perform the following:
 - a. Enter your name in the **Name** text box
 - b. Select the **Contact point type**, for example, Email and enter the email address. You can add many email addresses by using the ; separator.
 - c. Optional: Configure **Optional Email settings** and **Notification settings**.
3. Click **Save contact point**.
4. Select a notification channel in the alert rule:
 - a. From the menu, select **Notification policies** icon and then click + **New specific policy**.
 - b. Choose the **Contact point** you have just created
 - c. Click the **Save policy** button

Additional resources

- [Upstream Grafana documentation for alert notifications](#)

9.9. SETTING UP AUTHENTICATION BETWEEN PCP COMPONENTS

You can setup authentication using the **scram-sha-256** authentication mechanism, which is supported by PCP through the Simple Authentication Security Layer (SASL) framework.

Procedure

1. Install the **sasl** framework for the **scram-sha-256** authentication mechanism:

```
# dnf install cyrus-sasl-scram cyrus-sasl-lib
```

2. Specify the supported authentication mechanism and the user database path in the **pmcd.conf** file:

–


```
# vi /etc/sasl2/pmcd.conf

mech_list: scram-sha-256

sasldb_path: /etc/pcp/passwd.db
```

3. Create a new user:

```
# useradd -r metrics
```

Replace *metrics* by your user name.

4. Add the created user in the user database:

```
# saslpasswd2 -a pmcd metrics

Password:
Again (for verification):
```

To add the created user, you are required to enter the *metrics* account password.

5. Set the permissions of the user database:

```
# chown root:pcp /etc/pcp/passwd.db
# chmod 640 /etc/pcp/passwd.db
```

6. Restart the **pmcd** service:

```
# systemctl restart pmcd
```

Verification steps

- Verify the **sasl** configuration:

```
# pminfo -f -h "pcp://127.0.0.1?username=metrics" disk.dev.read
Password:
disk.dev.read
inst [0 or "sda"] value 19540
```

Additional resources

- **saslauthd(8)**, **pminfo(1)**, and **sha256** man pages
- [How can I setup authentication between PCP components, like PMDAs and pmcd in RHEL 8.2?](#)

9.10. INSTALLING PCP BPFTRACE

Install the PCP **bpfftrace** agent to introspect a system and to gather metrics from the kernel and user-space tracepoints.

The **bpfftrace** agent uses bpfftrace scripts to gather the metrics. The **bpfftrace** scripts use the enhanced Berkeley Packet Filter (**eBPF**).

This procedure describes how to install a **pcp bpfftrace**.

Prerequisites

1. PCP is configured. For more information, see [Setting up PCP with pcp-zeroconf](#).
2. The **grafana-server** is configured. For more information, see [Setting up a grafana-server](#).
3. The **scram-sha-256** authentication mechanism is configured. For more information, see [Setting up authentication between PCP components](#).

Procedure

1. Install the **pcp-pmda-bpfftrace** package:

```
# dnf install pcp-pmda-bpfftrace
```

2. Edit the **bpfftrace.conf** file and add the user that you have created in the {setting-up-authentication-between-pcp-components}:

```
# vi /var/lib/pcp/pmdas/bpfftrace/bpfftrace.conf

[dynamic_scripts]
enabled = true
auth_enabled = true
allowed_users = root,metrics
```

Replace *metrics* by your user name.

3. Install **bpfftrace** PMDA:

```
# cd /var/lib/pcp/pmdas/bpfftrace/
# ./Install
Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Check bpfftrace metrics have appeared ... 7 metrics and 6 values
```

The **pmda-bpfftrace** is now installed, and can only be used after authenticating your user. For more information, see [Viewing the PCP bpfftrace System Analysis dashboard](#).

Additional resources

- **pmdabpfftrace(1)** and **bpfftrace** man pages

9.11. VIEWING THE PCP BPFTRACE SYSTEM ANALYSIS DASHBOARD

Using the PCP bpfftrace data source, you can access the live data from sources which are not available as normal data from the **pmlogger** or archives

In the PCP bpfftrace data source, you can view the dashboard with an overview of useful metrics.

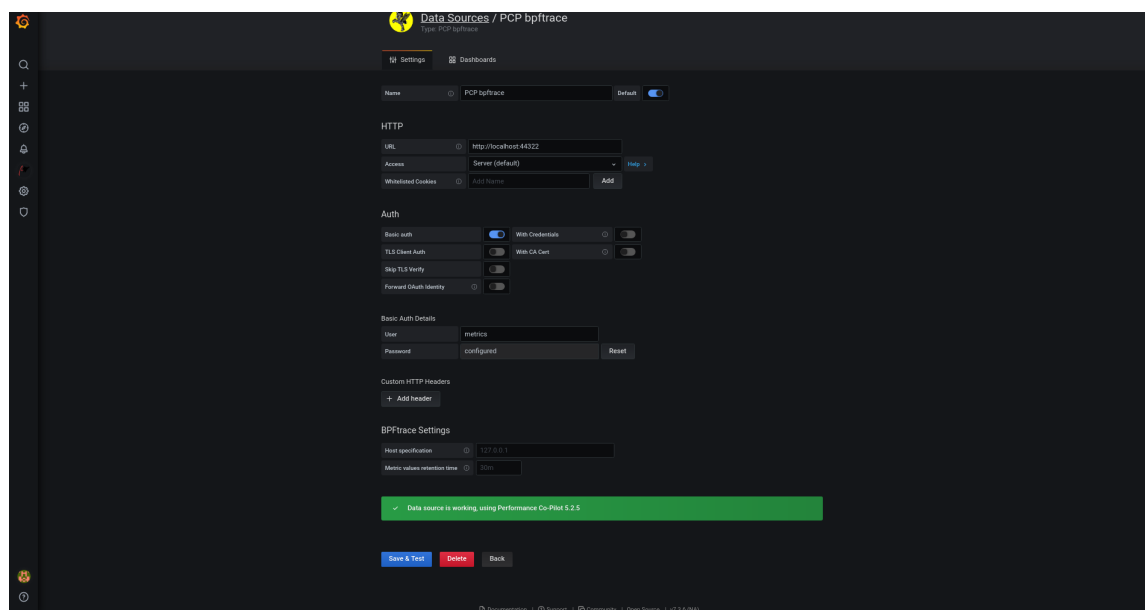
Prerequisites

1. The PCP bpfftrace is installed. For more information, see [Installing PCP bpfftrace](#).
2. The **grafana-server** is accessible. For more information, see [Accessing the Grafana web UI](#).

Procedure

1. Log into the Grafana web UI.
2. In the Grafana **Home** page, click **Add your first data source**
3. In the **Add data source** pane, type bpfftrace in the **Filter by name or type** text box and then click **PCP bpfftrace**.
4. In the **Data Sources / PCP bpfftrace** pane, perform the following:
 - a. Add **http://localhost:44322** in the **URL** field.
 - b. Toggle the **Basic Auth** option and add the created user credentials in the **User** and **Password** field.
 - c. Click **Save & Test**.

Figure 9.5. Adding PCP bpfftrace in the data source



- d. Click **Dashboards tab → Import → PCP bpfftrace: System Analysis** to see a dashboard with an overview of any useful metrics.

Figure 9.6. PCP bpfftrace: System Analysis



9.12. INSTALLING PCP VECTOR

This procedure describes how to install a **pcp vector**.

Prerequisites

1. PCP is configured. For more information, see [Setting up PCP with pcp-zeroconf](#).
2. The **grafana-server** is configured. For more information, see [Setting up a grafana-server](#).

Procedure

1. Install the **pcp-pmda-bcc** package:

```
# dnf install pcp-pmda-bcc
```

2. Install the **bcc** PMDA:

```
# cd /var/lib/pcp/pmdas/bcc
# ./Install
[Wed Apr 1 00:27:48] pmdabcc(22341) Info: Initializing, currently in 'notready' state.
[Wed Apr 1 00:27:48] pmdabcc(22341) Info: Enabled modules:
[Wed Apr 1 00:27:48] pmdabcc(22341) Info: ['biolatency', 'sysfork',
[...]
Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Check bcc metrics have appeared ... 1 warnings, 1 metrics and 0 values
```

Additional resources

- **pmdabcc(1)** man page

9.13. VIEWING THE PCP VECTOR CHECKLIST

The PCP Vector data source displays live metrics and uses the **pcp** metrics. It analyzes data for individual hosts.

After adding the PCP Vector data source, you can view the dashboard with an overview of useful metrics and view the related troubleshooting or reference links in the checklist.

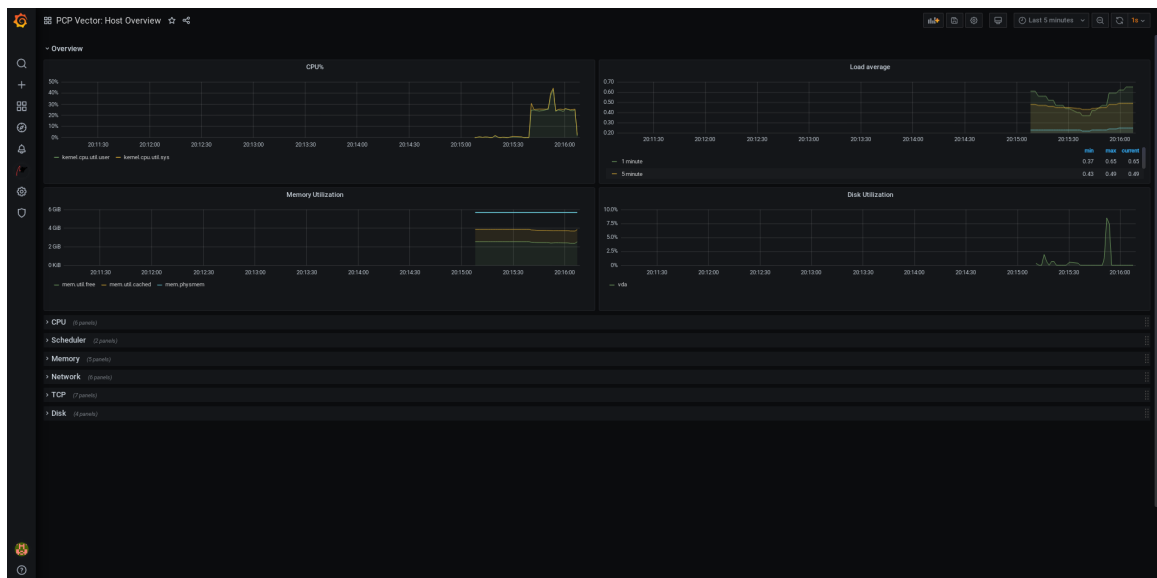
Prerequisites

1. The PCP Vector is installed. For more information, see [Installing PCP Vector](#).
2. The **grafana-server** is accessible. For more information, see [Accessing the Grafana web UI](#).

Procedure

1. Log into the Grafana web UI.
2. In the Grafana **Home** page, click **Add your first data source**
3. In the **Add data source** pane, type vector in the **Filter by name or type** text box and then click **PCP Vector**.
4. In the **Data Sources / PCP Vector** pane, perform the following:
 - a. Add **http://localhost:44322** in the **URL** field and then click **Save & Test**.
 - b. Click **Dashboards tab** → **Import** → **PCP Vector: Host Overview** to see a dashboard with an overview of any useful metrics.

Figure 9.7. PCP Vector: Host Overview



5. From the menu, hover over the  **Performance Co-Pilot** plugin and then click **PCP Vector Checklist**.



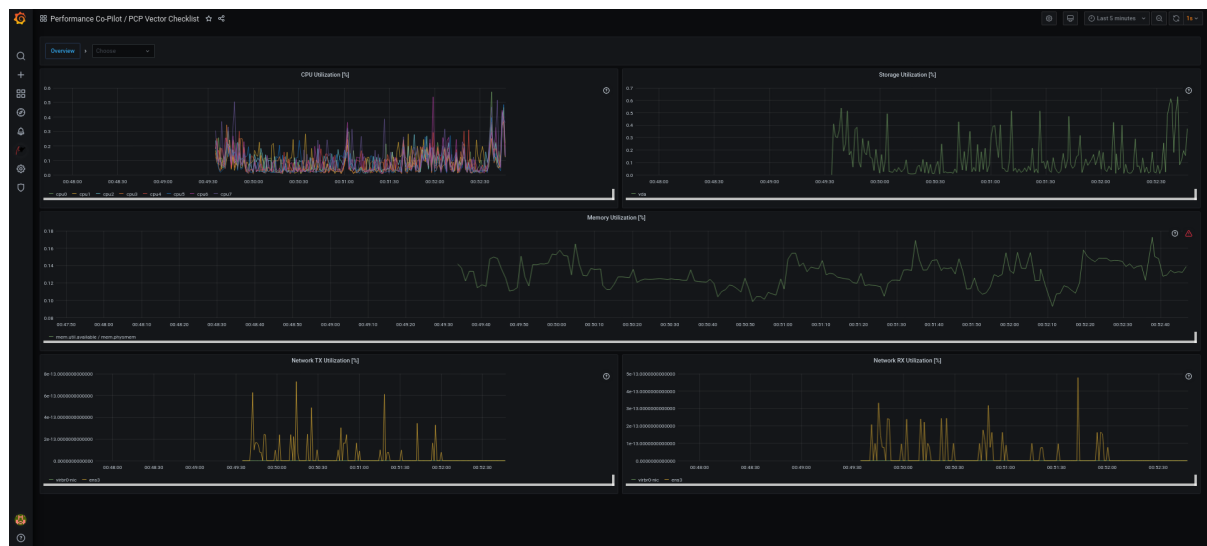
In the PCP checklist, click  help or  warning icon to view the related troubleshooting or reference links.

Figure 9.8. Performance Co-Pilot / PCP Vector Checklist



9.14. TROUBLESHOOTING GRAFANA ISSUES

It is sometimes necessary to troubleshoot Grafana issues, such as, Grafana does not display any data, the dashboard is black, or similar issues.

Procedure

- Verify that the **pmlogger** service is up and running by executing the following command:

```
$ systemctl status pmlogger
```

- Verify if files were created or modified to the disk by executing the following command:

```
$ ls /var/log/pcp/pmlogger/$(hostname)/ -rtl
total 4024
-rw-r--r--. 1 pcp pcp 45996 Oct 13 2019 20191013.20.07.meta.xz
-rw-r--r--. 1 pcp pcp 412 Oct 13 2019 20191013.20.07.index
-rw-r--r--. 1 pcp pcp 32188 Oct 13 2019 20191013.20.07.0.xz
-rw-r--r--. 1 pcp pcp 44756 Oct 13 2019 20191013.20.30-00.meta.xz
[..]
```

- Verify that the **pmproxy** service is running by executing the following command:

```
$ systemctl status pmproxy
```

- Verify that **pmproxy** is running, time series support is enabled, and a connection to Redis is established by viewing the **/var/log/pcp/pmproxy/pmproxy.log** file and ensure that it contains the following text:

```
pmproxy(1716) Info: Redis slots, command keys, schema version setup
```

Here, **1716** is the PID of **pmproxy**, which will be different for every invocation of **pmproxy**.

- Verify if the Redis database contains any keys by executing the following command:

```
$ redis-cli dbsize
(integer) 34837
```

- Verify if any PCP metrics are in the Redis database and **pmproxy** is able to access them by executing the following commands:

```
$ pmseries disk.dev.read
2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df

$ pmseries "disk.dev.read[count:10]"
2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
[Mon Jul 26 12:21:10.085468000 2021] 117971
70e83e88d4e1857a3a31605c6d1333755f2dd17c
[Mon Jul 26 12:21:00.087401000 2021] 117758
70e83e88d4e1857a3a31605c6d1333755f2dd17c
[Mon Jul 26 12:20:50.085738000 2021] 116688
70e83e88d4e1857a3a31605c6d1333755f2dd17c
[...]
```

```
$ redis-cli --scan --pattern "$$(pmseries 'disk.dev.read')"
```

```
pcp:metric.name:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:values:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:desc:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:labelvalue:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:instances:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:labelflags:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
```

- Verify if there are any errors in the Grafana logs by executing the following command:

```
$ journalctl -e -u grafana-server
-- Logs begin at Mon 2021-07-26 11:55:10 IST, end at Mon 2021-07-26 12:30:15 IST. --
Jul 26 11:55:17 localhost.localdomain systemd[1]: Starting Grafana instance...
Jul 26 11:55:17 localhost.localdomain grafana-server[1171]: t=2021-07-26T11:55:17+0530
lvl=info msg="Starting Grafana" logger=server version=7.3.6 c>
Jul 26 11:55:17 localhost.localdomain grafana-server[1171]: t=2021-07-26T11:55:17+0530
lvl=info msg="Config loaded from" logger=settings file=/usr/s>
Jul 26 11:55:17 localhost.localdomain grafana-server[1171]: t=2021-07-26T11:55:17+0530
lvl=info msg="Config loaded from" logger=settings file=/etc/g>
[...]
```

CHAPTER 10. OPTIMIZING THE SYSTEM PERFORMANCE USING THE WEB CONSOLE

Learn how to set a performance profile in the RHEL web console to optimize the performance of the system for a selected task.

10.1. PERFORMANCE TUNING OPTIONS IN THE WEB CONSOLE

Red Hat Enterprise Linux 9 provides several performance profiles that optimize the system for the following tasks:

- Systems using the desktop
- Throughput performance
- Latency performance
- Network performance
- Low power consumption
- Virtual machines

The **Tuned** service optimizes system options to match the selected profile.

In the web console, you can set which performance profile your system uses.

Additional resources

- [Getting started with Tuned](#)

10.2. SETTING A PERFORMANCE PROFILE IN THE WEB CONSOLE

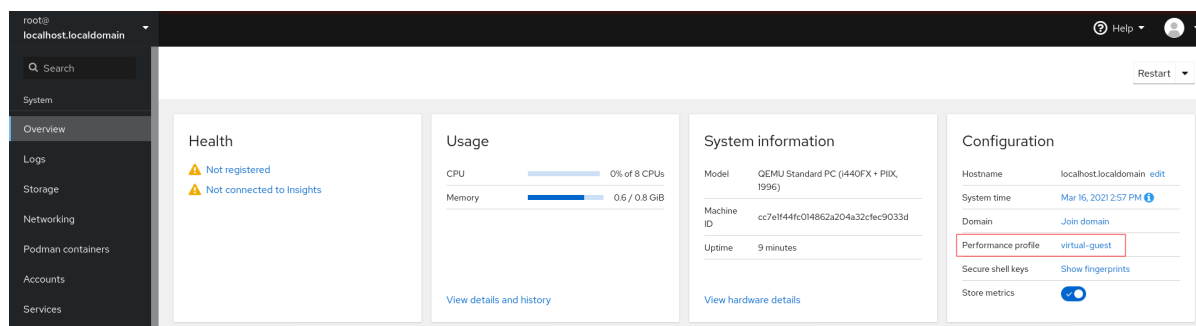
This procedure uses the web console to optimize the system performance for a selected task.

Prerequisites

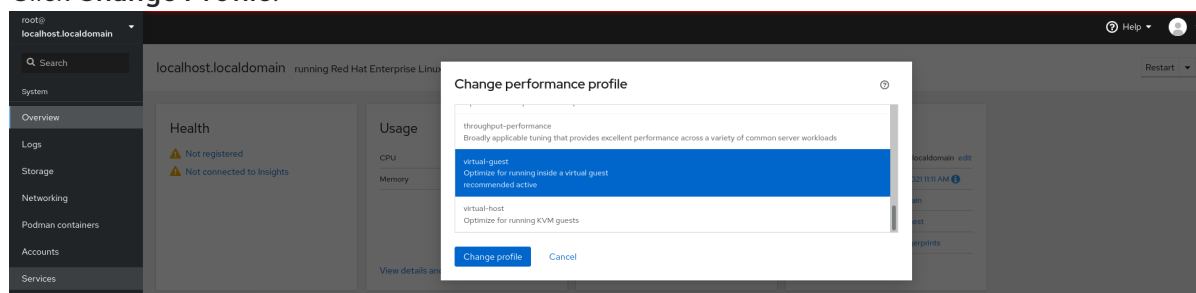
- Make sure the web console is installed and accessible. For details, see [Installing the web console](#).

Procedure

1. Log into the RHEL web console. For details, see [Logging in to the web console](#).
2. Click **Overview**.
3. In the **Performance Profile** field, click the current performance profile.



4. In the **Change Performance Profile** dialog box, change the profile if necessary.
5. Click **Change Profile**.



Verification steps

- The **Overview** tab now shows the selected performance profile.

10.3. MONITORING PERFORMANCE ON THE LOCAL SYSTEM USING THE WEB CONSOLE

Red Hat Enterprise Linux web console uses the Utilization Saturation and Errors (USE) Method for troubleshooting. The new performance metrics page has a historical view of your data organized chronologically with the newest data at the top.

In the **Metrics and history** page, you can view events, errors, and graphical representation for resource utilization and saturation.

Prerequisites

- The web console is installed and accessible. For details, see [Installing the web console](#).
- The **cockpit-pcp** package, which enables collecting the performance metrics, is installed:
 - a. To install the package from the web console interface:
 - i. Log in to the web console with administrative privileges. For details, see [Logging in to the web console](#).
 - ii. In the **Overview** page, click **View metrics and history**.
 - iii. Click the **Install cockpit-pcp** button.
 - iv. In the **Install software** dialog window, click **Install**.
 - b. To install the package from the command-line interface, use:

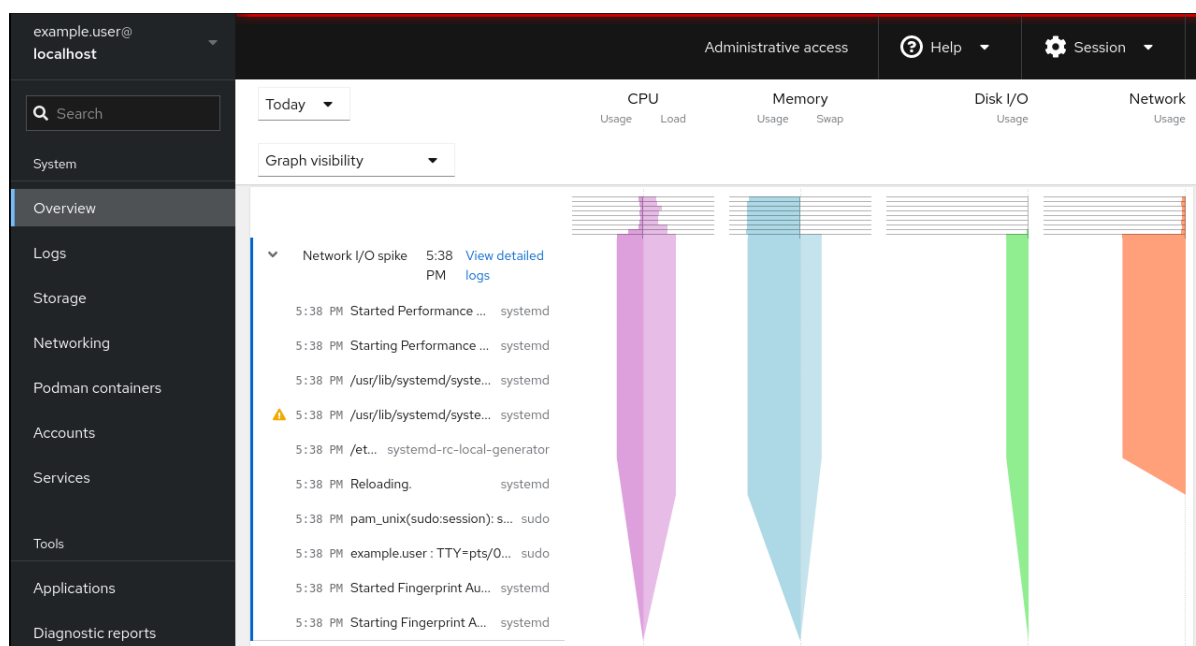
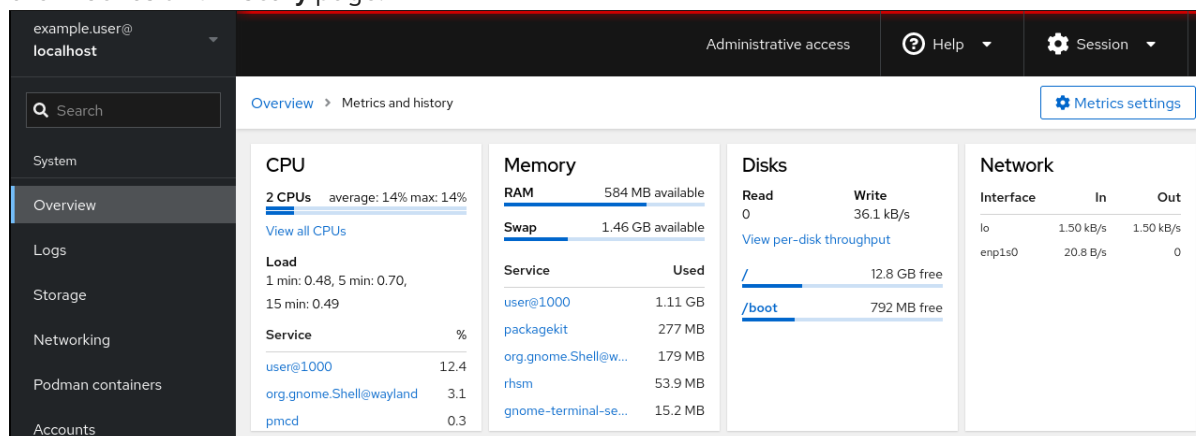
```
# dnf install cockpit-pcp
```

- The Performance Co-Pilot (PCP) service is enabled:

```
# systemctl enable --now pmlogger.service pmproxy.service
```

Procedure

- Log into the RHEL 9 web console. In the **Overview** page click **View metrics and history** to view the **Metrics and history** page.



10.4. MONITORING PERFORMANCE ON SEVERAL SYSTEMS USING THE WEB CONSOLE AND GRAFANA

Grafana enables you to collect data from several systems at once and review a graphical representation of their collected Performance Co-Pilot (PCP) metrics. You can set up performance metrics monitoring and export for several systems in the web console interface.

Prerequisites

- The web console must be installed and accessible. For details, see link:[Installing the web console](#).

- Install the **cockpit-pcp** package.
 1. From the web console interface:
 - a. Log in to the web console with administrative privileges. For details, see [Logging in to the web console](#).
 - b. In the **Overview** page, click **View details and history**.
 - c. Click the **Install cockpit-pcp** button.
 - d. In the **Install software** dialog window, click **Install**.
 - e. Log out and in again to see the metrics history.
 2. To install the package from the command-line interface, use:

```
# dnf install cockpit-pcp
```

- Enable the PCP service:

```
# systemctl enable --now pmlogger.service pmproxy.service
```

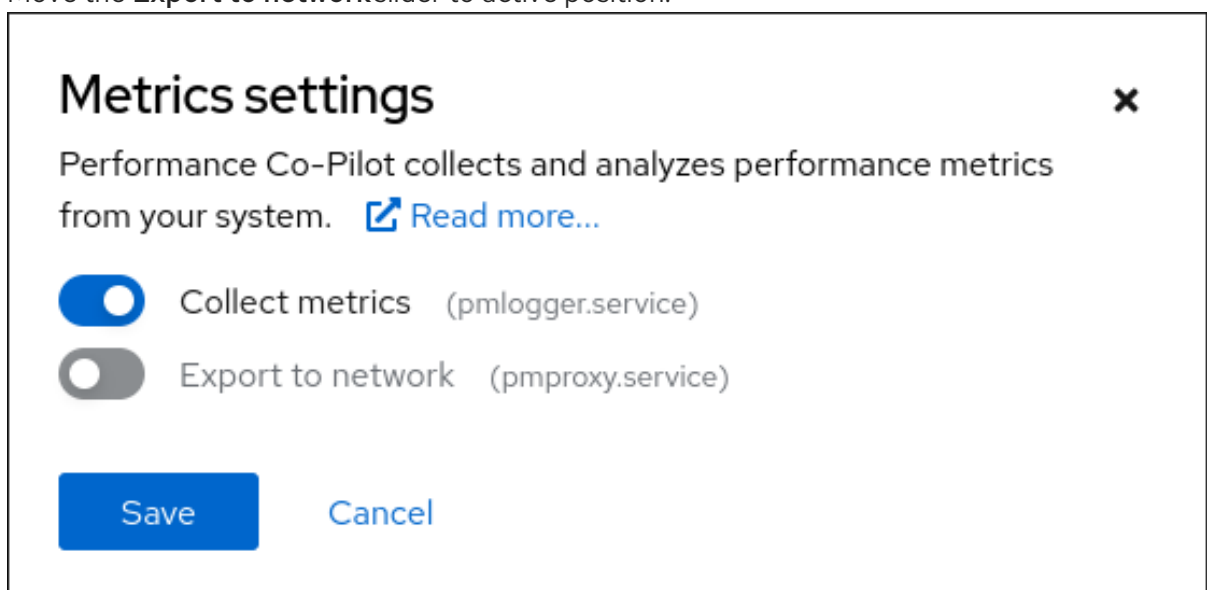
- Set up Grafana dashboard. For more information, see [Setting up a grafana-server](#).
- Install the **redis** package.

```
# dnf install redis
```

Alternatively, you can install the package from the web console interface later in the procedure.

Procedure

1. In the **Overview** page, click **View metrics and history** in the **Usage** table.
2. Click the **Metrics settings** button.
3. Move the **Export to network** slider to active position.



If you do not have the **redis** package installed, the web console prompts you to install it.

4. To open the **pmproxy** service, select a zone from a drop-down list and click the **Add pmproxy** button.
5. Click **Save**.

Verification

1. Click **Networking**.
2. In the **Firewall** table, click the **Edit rules and zones** button.
3. Search for **pmproxy** in your selected zone.



IMPORTANT

Repeat this procedure on all the systems you want to watch.

Additional resources

- [Setting up graphical representation of PCP metrics](#)

CHAPTER 11. SETTING THE DISK SCHEDULER

The disk scheduler is responsible for ordering the I/O requests submitted to a storage device.

You can configure the scheduler in several different ways:

- Set the scheduler using **TuneD**, as described in [Setting the disk scheduler using TuneD](#)
- Set the scheduler using **udev**, as described in [Setting the disk scheduler using udev rules](#)
- Temporarily change the scheduler on a running system, as described in [Temporarily setting a scheduler for a specific disk](#)



NOTE

In Red Hat Enterprise Linux 9, block devices support only multi-queue scheduling. This enables the block layer performance to scale well with fast solid-state drives (SSDs) and multi-core systems.

The traditional, single-queue schedulers, which were available in Red Hat Enterprise Linux 7 and earlier versions, have been removed.

11.1. AVAILABLE DISK SCHEDULERS

The following multi-queue disk schedulers are supported in Red Hat Enterprise Linux 9:

none

Implements a first-in first-out (FIFO) scheduling algorithm. It merges requests at the generic block layer through a simple last-hit cache.

mq-deadline

Attempts to provide a guaranteed latency for requests from the point at which requests reach the scheduler.

The **mq-deadline** scheduler sorts queued I/O requests into a read or write batch and then schedules them for execution in increasing logical block addressing (LBA) order. By default, read batches take precedence over write batches, because applications are more likely to block on read I/O operations. After **mq-deadline** processes a batch, it checks how long write operations have been starved of processor time and schedules the next read or write batch as appropriate.

This scheduler is suitable for most use cases, but particularly those in which the write operations are mostly asynchronous.

bfq

Targets desktop systems and interactive tasks.

The **bfq** scheduler ensures that a single application is never using all of the bandwidth. In effect, the storage device is always as responsive as if it was idle. In its default configuration, **bfq** focuses on delivering the lowest latency rather than achieving the maximum throughput.

bfq is based on **cfq** code. It does not grant the disk to each process for a fixed time slice but assigns a *budget* measured in number of sectors to the process.

This scheduler is suitable while copying large files and the system does not become unresponsive in this case.

kyber

The scheduler tunes itself to achieve a latency goal by calculating the latencies of every I/O request submitted to the block I/O layer. You can configure the target latencies for read, in the case of cache-misses, and synchronous write requests.

This scheduler is suitable for fast devices, for example NVMe, SSD, or other low latency devices.

11.2. DIFFERENT DISK SCHEDULERS FOR DIFFERENT USE CASES

Depending on the task that your system performs, the following disk schedulers are recommended as a baseline prior to any analysis and tuning tasks:

Table 11.1. Disk schedulers for different use cases

Use case	Disk scheduler
Traditional HDD with a SCSI interface	Use mq-deadline or bfq .
High-performance SSD or a CPU-bound system with fast storage	Use none , especially when running enterprise applications. Alternatively, use kyber .
Desktop or interactive tasks	Use bfq .
Virtual guest	Use mq-deadline . With a host bus adapter (HBA) driver that is multi-queue capable, use none .

11.3. THE DEFAULT DISK SCHEDULER

Block devices use the default disk scheduler unless you specify another scheduler.



NOTE

For **non-volatile Memory Express (NVMe)** block devices specifically, the default scheduler is **none** and Red Hat recommends not changing this.

The kernel selects a default disk scheduler based on the type of device. The automatically selected scheduler is typically the optimal setting. If you require a different scheduler, Red Hat recommends to use **udev** rules or the **TuneD** application to configure it. Match the selected devices and switch the scheduler only for those devices.

11.4. DETERMINING THE ACTIVE DISK SCHEDULER

This procedure determines which disk scheduler is currently active on a given block device.

Procedure

- Read the content of the **/sys/block/device/queue/scheduler** file:

```
# cat /sys/block/device/queue/scheduler
```

```
[mq-deadline] kyber bfq none
```

■

In the file name, replace *device* with the block device name, for example **sdc**.

The active scheduler is listed in square brackets (**[]**).

11.5. SETTING THE DISK SCHEDULER USING TUNED

This procedure creates and enables a **TuneD** profile that sets a given disk scheduler for selected block devices. The setting persists across system reboots.

In the following commands and configuration, replace:

- *device* with the name of the block device, for example **sdf**
- *selected-scheduler* with the disk scheduler that you want to set for the device, for example **bfq**

Prerequisites

- The **TuneD** service is installed and enabled. For details, see [Installing and enabling TuneD](#).

Procedure

1. Optional: Select an existing **TuneD** profile on which your profile will be based. For a list of available profiles, see [TuneD profiles distributed with RHEL](#).
To see which profile is currently active, use:

```
$ tuned-adm active
```

2. Create a new directory to hold your **TuneD** profile:

```
# mkdir /etc/tuned/my-profile
```

3. Find the system unique identifier of the selected block device:

```
$ udevadm info --query=property --name=/dev/device | grep -E '(WWN|SERIAL)'

ID_WWN=0x5002538d00000000_
ID_SERIAL=Generic-_SD_MMC_20120501030900000-0:0
ID_SERIAL_SHORT=20120501030900000
```



NOTE

The command in this example will return all values identified as a World Wide Name (WWN) or serial number associated with the specified block device. Although it is preferred to use a WWN, the WWN is not always available for a given device and any values returned by the example command are acceptable to use as the *device system unique ID*.

4. Create the **/etc/tuned/my-profile/tuned.conf** configuration file. In the file, set the following options:
 - a. Optional: Include an existing profile:

```
[main]
include=existing-profile
```

- b. Set the selected disk scheduler for the device that matches the WWN identifier:

```
[disk]
devices_udev_regex=IDNAME=device system unique id
elevator=selected-scheduler
```

Here:

- Replace *IDNAME* with the name of the identifier being used (for example, **ID_WWN**).
- Replace *device system unique id* with the value of the chosen identifier (for example, **0x5002538d00000000**).
To match multiple devices in the **devices_udev_regex** option, enclose the identifiers in parentheses and separate them with vertical bars:

```
devices_udev_regex=(ID_WWN=0x5002538d00000000)|
(ID_WWN=0x1234567800000000)
```

5. Enable your profile:

```
# tuned-adm profile my-profile
```

Verification steps

1. Verify that the TuneD profile is active and applied:

```
$ tuned-adm active
```

```
Current active profile: my-profile
```

```
$ tuned-adm verify
```

```
Verification succeeded, current system settings match the preset profile.
See TuneD log file ('/var/log/tuned/tuned.log') for details.
```

2. Read the contents of the **/sys/block/*device*/queue/scheduler** file:

```
# cat /sys/block/device/queue/scheduler
```

```
[mq-deadline] kyber bfq none
```

In the file name, replace *device* with the block device name, for example **sdc**.

The active scheduler is listed in square brackets (**[]**).

Additional resources

- [Customizing TuneD profiles.](#)

11.6. SETTING THE DISK SCHEDULER USING UDEV RULES

This procedure sets a given disk scheduler for specific block devices using **udev** rules. The setting persists across system reboots.

In the following commands and configuration, replace:

- *device* with the name of the block device, for example **sdf**
- *selected-scheduler* with the disk scheduler that you want to set for the device, for example **bfq**

Procedure

1. Find the system unique identifier of the block device:

```
$ udevadm info --name=/dev/device | grep -E '(WWN|SERIAL)'
E: ID_WWN=0x5002538d00000000
E: ID_SERIAL=Generic-SD_MMC_20120501030900000-0:0
E: ID_SERIAL_SHORT=20120501030900000
```



NOTE

The command in this example will return all values identified as a World Wide Name (WWN) or serial number associated with the specified block device. Although it is preferred to use a WWN, the WWN is not always available for a given device and any values returned by the example command are acceptable to use as the *device system unique ID*.

2. Configure the **udev** rule. Create the **/etc/udev/rules.d/99-scheduler.rules** file with the following content:

```
ACTION=="add|change", SUBSYSTEM=="block", ENV{IDNAME}=="device system unique id", ATTR{queue/scheduler}="selected-scheduler"
```

Here:

- Replace *IDNAME* with the name of the identifier being used (for example, **ID_WWN**).
- Replace *device system unique id* with the value of the chosen identifier (for example, **0x5002538d00000000**).

3. Reload **udev** rules:

```
# udevadm control --reload-rules
```

4. Apply the scheduler configuration:

```
# udevadm trigger --type=devices --action=change
```

Verification steps

- Verify the active scheduler:

```
# cat /sys/block/device/queue/scheduler
```

11.7. TEMPORARILY SETTING A SCHEDULER FOR A SPECIFIC DISK

This procedure sets a given disk scheduler for specific block devices. The setting does not persist across system reboots.

Procedure

- Write the name of the selected scheduler to the **/sys/block/*device*/queue/scheduler** file:

```
# echo selected-scheduler > /sys/block/device/queue/scheduler
```

In the file name, replace *device* with the block device name, for example **sdc**.

Verification steps

- Verify that the scheduler is active on the device:

```
# cat /sys/block/device/queue/scheduler
```

CHAPTER 12. TUNING THE PERFORMANCE OF A SAMBA SERVER

Learn what settings can improve the performance of Samba in certain situations, and which settings can have a negative performance impact.

Parts of this section were adopted from the [Performance Tuning](#) documentation published in the Samba Wiki. License: [CC BY 4.0](#). Authors and contributors: See the [history](#) tab on the Wiki page.

Prerequisites

- Samba is set up as a file or print server

12.1. SETTING THE SMB PROTOCOL VERSION

Each new SMB version adds features and improves the performance of the protocol. The recent Windows and Windows Server operating systems always supports the latest protocol version. If Samba also uses the latest protocol version, Windows clients connecting to Samba benefit from the performance improvements. In Samba, the default value of the server max protocol is set to the latest supported stable SMB protocol version.



NOTE

To always have the latest stable SMB protocol version enabled, do not set the **server max protocol** parameter. If you set the parameter manually, you will need to modify the setting with each new version of the SMB protocol, to have the latest protocol version enabled.

The following procedure explains how to use the default value in the **server max protocol** parameter.

Procedure

1. Remove the **server max protocol** parameter from the **[global]** section in the **/etc/samba/smb.conf** file.
2. Reload the Samba configuration

```
# smbcontrol all reload-config
```

12.2. TUNING SHARES WITH DIRECTORIES THAT CONTAIN A LARGE NUMBER OF FILES

Linux supports case-sensitive file names. For this reason, Samba needs to scan directories for uppercase and lowercase file names when searching or accessing a file. You can configure a share to create new files only in lowercase or uppercase, which improves the performance.

Prerequisites

- Samba is configured as a file server

Procedure

1. Rename all files on the share to lowercase.

**NOTE**

Using the settings in this procedure, files with names other than in lowercase will no longer be displayed.

2. Set the following parameters in the share's section:

```
case sensitive = true
default case = lower
preserve case = no
short preserve case = no
```

For details about the parameters, see their descriptions in the **smb.conf(5)** man page.

3. Verify the **/etc/samba/smb.conf** file:

```
# testparm
```

4. Reload the Samba configuration:

```
# smbcontrol all reload-config
```

After you applied these settings, the names of all newly created files on this share use lowercase. Because of these settings, Samba no longer needs to scan the directory for uppercase and lowercase, which improves the performance.

12.3. SETTINGS THAT CAN HAVE A NEGATIVE PERFORMANCE IMPACT

By default, the kernel in Red Hat Enterprise Linux is tuned for high network performance. For example, the kernel uses an auto-tuning mechanism for buffer sizes. Setting the **socket options** parameter in the **/etc/samba/smb.conf** file overrides these kernel settings. As a result, setting this parameter decreases the Samba network performance in most cases.

To use the optimized settings from the Kernel, remove the **socket options** parameter from the **[global]** section in the **/etc/samba/smb.conf**.

CHAPTER 13. OPTIMIZING VIRTUAL MACHINE PERFORMANCE

Virtual machines (VMs) always experience some degree of performance deterioration in comparison to the host. The following sections explain the reasons for this deterioration and provide instructions on how to minimize the performance impact of virtualization in RHEL 9, so that your hardware infrastructure resources can be used as efficiently as possible.

13.1. WHAT INFLUENCES VIRTUAL MACHINE PERFORMANCE

VMs are run as user-space processes on the host. The hypervisor therefore needs to convert the host's system resources so that the VMs can use them. As a consequence, a portion of the resources is consumed by the conversion, and the VM therefore cannot achieve the same performance efficiency as the host.

The impact of virtualization on system performance

More specific reasons for VM performance loss include:

- Virtual CPUs (vCPUs) are implemented as threads on the host, handled by the Linux scheduler.
- VMs do not automatically inherit optimization features, such as NUMA or huge pages, from the host kernel.
- Disk and network I/O settings of the host might have a significant performance impact on the VM.
- Network traffic typically travels to a VM through a software-based bridge.
- Depending on the host devices and their models, there might be significant overhead due to emulation of particular hardware.

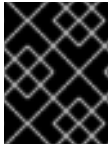
The severity of the virtualization impact on the VM performance is influenced by a variety of factors, which include:

- The number of concurrently running VMs.
- The amount of virtual devices used by each VM.
- The device types used by the VMs.

Reducing VM performance loss

RHEL 9 provides a number of features you can use to reduce the negative performance effects of virtualization. Notably:

- [The **Tuned** service](#) can automatically optimize the resource distribution and performance of your VMs.
- [Block I/O tuning](#) can improve the performances of the VM's block devices, such as disks.
- [NUMA tuning](#) can increase vCPU performance.
- [Virtual networking](#) can be optimized in various ways.

**IMPORTANT**

Tuning VM performance can have adverse effects on other virtualization functions. For example, it can make migrating the modified VM more difficult.

13.2. OPTIMIZING VIRTUAL MACHINE PERFORMANCE USING TUNED

The **Tuned** utility is a tuning profile delivery mechanism that adapts RHEL for certain workload characteristics, such as requirements for CPU-intensive tasks or storage-network throughput responsiveness. It provides a number of tuning profiles that are pre-configured to enhance performance and reduce power consumption in a number of specific use cases. You can edit these profiles or create new profiles to create performance solutions tailored to your environment, including virtualized environments.

To optimize RHEL 9 for virtualization, use the following profiles:

- For RHEL 9 virtual machines, use the **virtual-guest** profile. It is based on the generally applicable **throughput-performance** profile, but also decreases the swappiness of virtual memory.
- For RHEL 9 virtualization hosts, use the **virtual-host** profile. This enables more aggressive writeback of dirty memory pages, which benefits the host performance.

Prerequisites

- The **Tuned** service is [installed and enabled](#).

Procedure

To enable a specific **Tuned** profile:

1. List the available **Tuned** profiles.

```
# tuned-adm list
```

Available profiles:

```
- balanced          - General non-specialized Tuned profile
- desktop          - Optimize for the desktop use-case
[...]
- virtual-guest    - Optimize for running inside a virtual guest
- virtual-host     - Optimize for running KVM guests
Current active profile: balanced
```

2. **Optional:** Create a new **Tuned** profile or edit an existing **Tuned** profile.
For more information, see [Customizing Tuned profiles](#).
3. Activate a **Tuned** profile.

```
# tuned-adm profile selected-profile
```

- To optimize a virtualization host, use the *virtual-host* profile.

```
# tuned-adm profile virtual-host
```

- On a RHEL guest operating system, use the *virtual-guest* profile.

```
# tuned-adm profile virtual-guest
```

Additional resources

- [Monitoring and managing system status and performance](#)

13.3. OPTIMIZING LIBVIRT DAEMONS

The **libvirt** virtualization suite works as a management layer for the RHEL hypervisor, and your **libvirt** configuration significantly impacts your virtualization host. Notably, RHEL 9 contains two different types of **libvirt** daemons, monolithic or modular, and which type of daemons you use affects how granularly you can configure individual virtualization drivers.

13.3.1. Types of libvirt daemons

RHEL 9 supports the following **libvirt** daemon types:

Monolithic libvirt

The traditional **libvirt** daemon, **libvirtd**, controls a wide variety of virtualization drivers, using a single configuration file – `/etc/libvirt/libvirtd.conf`.

As such, **libvirtd** allows for centralized hypervisor configuration, but may use system resources inefficiently. Therefore, **libvirtd** will become unsupported in a future major release of RHEL.

However, if you updated to RHEL 9 from RHEL 8, your host still uses **libvirtd** by default.

Modular libvirt

Newly introduced in RHEL 9, modular **libvirt** provides a specific daemon for each virtualization driver. These include the following:

- **virtqemud** – A primary daemon for hypervisor management
- **virtinterfaced** – A secondary daemon for host NIC management
- **virtnetworkd** – A secondary daemon for virtual network management
- **virtnodevdevd** – A secondary daemon for host physical device management
- **virtnwfilterd** – A secondary daemon for host firewall management
- **virtsecret** – A secondary daemon for host secret management
- **virtstoraged** – A secondary daemon for storage management

Each of the daemons has a separate configuration file – for example `/etc/libvirt/virtqemud.conf`. As such, modular **libvirt** daemons provide better options for fine-tuning **libvirt** resource management.

If you performed a fresh install of RHEL 9, modular **libvirt** is configured by default.

Next steps

- If your RHEL 9 uses **libvirtd**, Red Hat recommends switching to modular daemons. For instructions, see [Enabling modular libvirt daemons](#).

13.3.2. Enabling modular libvirt daemons

In RHEL 9, the **libvirt** library uses modular daemons that handle individual virtualization driver sets on your host. For example, the **virtqemu** daemon handles QEMU drivers.

If you performed a fresh install of a RHEL 9 host, your hypervisor uses modular **libvirt** daemons by default. However, if you upgraded your host from RHEL 8 to RHEL 9, your hypervisor uses the monolithic **libvirtd** daemon, which is the default in RHEL 8.

If that is the case, Red Hat recommends enabling the modular **libvirt** daemons instead, because they provide better options for fine-tuning **libvirt** resource management. In addition, **libvirtd** will become unsupported in a future major release of RHEL.

Prerequisites

- Your hypervisor is using the monolithic **libvirtd** service.

```
# systemctl is-active libvirtd.service
active
```

If this command displays **active**, you are using **libvirtd**.

- Your virtual machines are shut down.

Procedure

- Stop **libvirtd** and its sockets.

```
# systemctl stop libvirtd.service
# systemctl stop libvirtd{,-ro,-admin,-tcp,-tls}.socket
```

- Disable **libvirtd** to prevent it from starting on boot.

```
$ systemctl disable libvirtd.service
$ systemctl disable libvirtd{,-ro,-admin,-tcp,-tls}.socket
```

- Enable the modular **libvirt** daemons.

```
# for drv in qemu interface network nodedev nwfilter secret storage; do systemctl unmask
virt${drv}d.service; systemctl unmask virt${drv}d{,-ro,-admin}.socket; systemctl enable
virt${drv}d.service; systemctl enable virt${drv}d{,-ro,-admin}.socket; done
```

- Start the sockets for the modular daemons.

```
# for drv in qemu network nodedev nwfilter secret storage; do systemctl start virt${drv}d{,-ro,-
admin}.socket; done
```

- Optional:** If you require connecting to your host from remote hosts, enable and start the virtualization proxy daemon.

- Check whether the **libvirtd-tls.socket** service is enabled on your system.


```
# cat /etc/libvirt/libvirt.conf | grep listen_tls

listen_tls = 0
```

- b. If **libvirtd-tls.socket** is not enabled (**listen_tls = 0**), activate **virtproxyd** as follows:

```
# systemctl unmask virtproxyd.service
# systemctl unmask virtproxyd{,-ro,-admin}.socket
# systemctl enable virtproxyd.service
# systemctl enable virtproxyd{,-ro,-admin}.socket
# systemctl start virtproxyd{,-ro,-admin}.socket
```

- c. If **libvirtd-tls.socket** is enabled (**listen_tls = 1**), activate **virtproxyd** as follows:

```
# systemctl unmask virtproxyd.service
# systemctl unmask virtproxyd{,-ro,-admin,-tls}.socket
# systemctl enable virtproxyd.service
# systemctl enable virtproxyd{,-ro,-admin,-tls}.socket
# systemctl start virtproxyd{,-ro,-admin,-tls}.socket
```

To enable the TLS socket of **virtproxyd**, your host must have TLS certificates configured to work with **libvirt**. For more information, see the [Upstream libvirt documentation](#).

Verification

1. Activate the enabled virtualization daemons.

```
# virsh uri
qemu:///system
```

2. Verify that your host is using the **virtqemud** modular daemon.

```
# systemctl is-active virtqemud.service
active
```

If the status is **active**, you have successfully enabled modular **libvirt** daemons.

13.4. CONFIGURING VIRTUAL MACHINE MEMORY

To improve the performance of a virtual machine (VM), you can assign additional host RAM to the VM. Similarly, you can decrease the amount of memory allocated to a VM so the host memory can be allocated to other VMs or tasks.

To perform these actions, you can use [the web console](#) or [the command-line interface](#).

13.4.1. Adding and removing virtual machine memory using the web console

To improve the performance of a virtual machine (VM) or to free up the host resources it is using, you can use the web console to adjust amount of memory allocated to the VM.

Prerequisites

- The guest OS is running the memory balloon drivers. To verify this is the case:

1. Ensure the VM's configuration includes the **memballoon** device:

```
# virsh dumpxml testguest | grep memballoon
<memballoon model='virtio'>
  </memballoon>
```

If this commands displays any output and the model is not set to **none**, the **memballoon** device is present.

2. Ensure the balloon drivers are running in the guest OS.
 - In Windows guests, the drivers are installed as a part of the **virtio-win** driver package. For instructions, see [Installing paravirtualized KVM drivers for Windows virtual machines](#).
 - In Linux guests, the drivers are generally included by default and activate when the **memballoon** device is present.
- The web console VM plug-in [is installed on your system](#).

Procedure

1. **Optional:** Obtain the information about the maximum memory and currently used memory for a VM. This will serve as a baseline for your changes, and also for verification.

```
# virsh dominfo testguest
Max memory: 2097152 KiB
Used memory: 2097152 KiB
```

2. In the **Virtual Machines** interface, click the VM whose information you want to see.
A new page opens with an Overview section with basic information about the selected VM and a Console section to access the VM's graphical interface.
3. Click **edit** next to the **Memory** line in the Overview pane.
The **Memory Adjustment** dialog appears.

Grid_v2 memory adjustment ✕

Current allocation 128 3072 3072 MiB ▼

Maximum allocation 128 15626 3072 MiB ▼

Save Cancel

4. Configure the virtual CPUs for the selected VM.
 - **Maximum allocation** - Sets the maximum amount of host memory that the VM can use for its processes. You can specify the maximum memory when creating the VM or increase it later. You can specify memory as multiples of MiB or GiB.
Adjusting maximum memory allocation is only possible on a shut-off VM.
 - **Current allocation** - Sets the actual amount of memory allocated to the VM. This value can

be less than the Maximum allocation but cannot exceed it. You can adjust the value to regulate the memory available to the VM for its processes. You can specify memory as multiples of MiB or GiB.

If you do not specify this value, the default allocation is the **Maximum allocation** value.

5. Click **Save**.

The memory allocation of the VM is adjusted.

Additional resources

- [Adding and removing virtual machine memory using the command-line interface](#)
- [Optimizing virtual machine CPU performance](#)

13.4.2. Adding and removing virtual machine memory using the command-line interface

To improve the performance of a virtual machine (VM) or to free up the host resources it is using, you can use the CLI to adjust amount of memory allocated to the VM.

Prerequisites

- The guest OS is running the memory balloon drivers. To verify this is the case:
 1. Ensure the VM's configuration includes the **memballoon** device:

```
# virsh dumpxml testguest | grep memballoon
<memballoon model='virtio'>
  </memballoon>
```

If this commands displays any output and the model is not set to **none**, the **memballoon** device is present.

2. Ensure the ballon drivers are running in the guest OS.
 - In Windows guests, the drivers are installed as a part of the **virtio-win** driver package. For instructions, see [Installing paravirtualized KVM drivers for Windows virtual machines](#).
 - In Linux guests, the drivers are generally included by default and activate when the **memballoon** device is present.

Procedure

1. **Optional:** Obtain the information about the maximum memory and currently used memory for a VM. This will serve as a baseline for your changes, and also for verification.

```
# virsh dominfo testguest
Max memory: 2097152 KiB
Used memory: 2097152 KiB
```

2. Adjust the maximum memory allocated to a VM. Increasing this value improves the performance potential of the VM, and reducing the value lowers the performance footprint the VM has on your host. Note that this change can only be performed on a shut-off VM, so adjusting a running VM requires a reboot to take effect.

For example, to change the maximum memory that the *testguest* VM can use to 4096 MiB:

```
# virt-xml testguest --edit --memory memory=4096,currentMemory=4096
Domain 'testguest' defined successfully.
Changes will take effect after the domain is fully powered off.
```

To increase the maximum memory of a running VM, you can attach a memory device to the VM. This is also referred to as **memory hot plug**. For details, see [Attaching devices to virtual machines](#).



WARNING

Removing memory devices from a running VM (also referred to as a memory hot unplug) is not supported, and highly discouraged by Red Hat.

3. **Optional:** You can also adjust the memory currently used by the VM, up to the maximum allocation. This regulates the memory load that the VM has on the host until the next reboot, without changing the maximum VM allocation.

```
# virsh setmem testguest --current 2048
```

Verification

1. Confirm that the memory used by the VM has been updated:

```
# virsh dominfo testguest
Max memory: 4194304 KiB
Used memory: 2097152 KiB
```

2. **Optional:** If you adjusted the current VM memory, you can obtain the memory balloon statistics of the VM to evaluate how effectively it regulates its memory use.

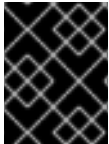
```
# virsh domstats --balloon testguest
Domain: 'testguest'
balloon.current=365624
balloon.maximum=4194304
balloon.swap_in=0
balloon.swap_out=0
balloon.major_fault=306
balloon.minor_fault=156117
balloon.unused=3834448
balloon.available=4035008
balloon.usable=3746340
balloon.last-update=1587971682
balloon.disk_caches=75444
balloon.hugetlb_pgalloc=0
balloon.hugetlb_pgfail=0
balloon.rss=1005456
```

Additional resources

- [Adding and removing virtual machine memory using the web console](#)
- [Optimizing virtual machine CPU performance](#)

13.4.3. Adding and removing virtual machine memory by using virtio-mem

As a Technology Preview, RHEL 9 provides the **virtio-mem** paravirtualized memory device. **virtio-mem** can be used to dynamically add or remove host memory in virtual machines (VMs). For example, you can use this device to move memory resources between running VMs or to resize VM memory in cloud setups based on your current requirements.



IMPORTANT

virtio-mem is included in RHEL 9 as a Technology Preview, which means it is not supported.

13.4.3.1. Prerequisites

- The host has Intel 64 or AMD64 CPU architecture.
- The host and VMs use RHEL 9 as the operating system.

13.4.3.2. Overview of virtio-mem

virtio-mem is a paravirtualized memory device that can be used to dynamically add or remove host memory in virtual machines (VMs). For example, you can use this device to move memory resources between running VMs or to resize VM memory in cloud setups based on your current requirements.

Using **virtio-mem** you can increase the memory of a VM beyond its initial size, and shrink it back to its original size, in units that can have the size of 2 to several hundred mebibytes (MiBs). Note, however, that **virtio-mem** also relies on a specific guest operating system configuration, especially to reliably unplug memory.



IMPORTANT

virtio-mem is included in RHEL 9 as a Technology Preview, which means it is not supported.

virtio-mem Technology Preview limitations

virtio-mem is currently not compatible with the following features:

- Using pre-allocation of memory on the host
- Using HugePages on the host
- Using memory locking for real-time applications on the host
- Using encrypted virtualization on the host
- Dynamic memory metadata allocation in the host
- Combining **virtio-mem** with **memballoon** inflation and deflation on the host

- Unplugging the **virtio-mem** device from a VM
- Unloading or reloading the **virtio_mem** driver in a VM
- Hibernating or suspending a VM with the **virtio_mem** driver loaded

Memory onlineing in virtual machines

When attaching memory to a running RHEL VM (also known as memory hot-plugging), you must set the hot-plugged memory to an online state in the VM operating system. Otherwise, the system will not be able to use the memory.

The following table summarizes the main considerations when choosing between the available memory onlineing configurations.

Table 13.1. Comparison of memory onlineing configurations

Configuration name	Unplugging memory from a VM	A risk of creating a memory zone imbalance	A potential use case	Memory requirements of the intended workload
online_movable	Hot-plugged memory can be reliably unplugged.	Yes	Hot-plugging a comparatively small amount of memory	Mostly user-space memory
auto-movable	Movable portions of hot-plugged memory can be reliably unplugged.	Minimal	Hot-plugging a large amount of memory	Mostly user-space memory
online_kernel	Hot-plugged memory cannot be reliably unplugged.	No	Unreliable memory unplugging is acceptable.	User-space or kernel-space memory

A *zone imbalance* is a lack of available memory pages in one of the Linux memory zones. A *zone imbalance* can negatively impact the system performance. For example, the kernel might crash if it runs out of free memory for unmovable allocations. Usually, movable allocations contain mostly user-space memory pages and unmovable allocations contain mostly kernel-space memory pages.

For a more detailed explanation of memory onlineing considerations, see: [Onlineing and Offlining Memory Blocks](#) and [Zone Imbalances](#)

Additional resources

- [Configuring memory onlineing in virtual machines](#)
- [Attaching a virtio-mem device to virtual machines](#)
- [Onlineing and Offlining Memory Blocks](#)
- [Zone Imbalances](#)

13.4.3.3. Configuring memory online in virtual machines

Before using **virtio-mem** to attach memory to a running virtual machine (also known as memory hot-plugging), you must configure the virtual machine (VM) operating system to automatically set the hot-plugged memory to an online state. Otherwise, the guest operating system is not able to use the additional memory. You can choose from one of the following configurations for memory onlineing:

- **online_movable**
- **online_kernel**
- **auto-movable**

To learn about differences between these configurations, see: [Memory onlineing in virtual machines](#)

Memory onlineing is configured with udev rules by default in RHEL. However, when using **virtio-mem**, it is recommended to configure memory onlineing directly in the kernel.



IMPORTANT

virtio-mem is included in RHEL 9 as a Technology Preview, which means it is not supported.

Prerequisites

- The host has Intel 64 or AMD64 CPU architecture.
- The host and VMs use RHEL 9 as the operating system.

Procedure

- To set memory onlineing to use the **online_movable** configuration in the VM:
 1. Set the **memhp_default_state** kernel command line parameter to **online_movable**:

```
# grubby --update-kernel=ALL --remove-args=memhp_default_state --
args=memhp_default_state=online_movable
```

2. Reboot the VM.

- To set memory onlineing to use the **online_kernel** configuration in the VM:
 1. Set the **memhp_default_state** kernel command line parameter to **online_kernel**:

```
# grubby --update-kernel=ALL --remove-args=memhp_default_state --
args=memhp_default_state=online_kernel
```

2. Reboot the VM.

- To use the **auto-movable** memory onlineing policy in the VM:
 1. Set the **memhp_default_state** kernel command line parameter to **online**:

```
# grubby --update-kernel=ALL --remove-args=memhp_default_state --
args=memhp_default_state=online
```

2. Set the **memory_hotplug.online_policy** kernel command line parameter to **auto-movable**:

```
# grubby --update-kernel=ALL --remove-args="memory_hotplug.online_policy" --
args=memory_hotplug.online_policy=auto-movable
```

3. **Optional:** To further tune the **auto-movable** onlining policy, change the **memory_hotplug.auto_movable_ratio** and **memory_hotplug.auto_movable_numa_aware** parameters:

```
# grubby --update-kernel=ALL --remove-args="memory_hotplug.auto_movable_ratio" --
args=memory_hotplug.auto_movable_ratio=<percentage>
```

```
# grubby --update-kernel=ALL --remove-
args="memory_hotplug.memory_auto_movable_numa_aware" --
args=memory_hotplug.auto_movable_numa_aware=<y/n>
```

- The **memory_hotplug.auto_movable_ratio** parameter sets the maximum ratio of memory only available for movable allocations compared to memory available for any allocations. The ratio is expressed in percents and the default value is: 301 (%), which is a 3:1 ratio.
- The **memory_hotplug.auto_movable_numa_aware** parameter controls whether the **memory_hotplug.auto_movable_ratio** parameter applies to memory across all available NUMA nodes or only for memory within a single NUMA node. The default value is: *y* (yes)
For example, if the maximum ratio is set to 301% and the **memory_hotplug.auto_movable_numa_aware** is set to *y* (yes), then the 3:1 ratio is applied even within the NUMA node with the attached **virtio-mem** device. If the parameter is set to *n* (no), the maximum 3:1 ratio is applied only for all the NUMA nodes as a whole.

Additionally, if the ratio is not exceeded, the newly hot-plugged memory will be available only for movable allocations. Otherwise, the newly hot-plugged memory will be available for both movable and unmovable allocations.

4. Reboot the VM.

Verification

- To see if the **online_movable** configuration has been set correctly, check the current value of the **memhp_default_state** kernel parameter:

```
# cat /sys/devices/system/memory/auto_online_blocks

online_movable
```

- To see if the **online_kernel** configuration has been set correctly, check the current value of the **memhp_default_state** kernel parameter:

```
# cat /sys/devices/system/memory/auto_online_blocks

online_kernel
```


- To see if the **auto-movable** configuration has been set correctly, check the following kernel parameters:

- **memhp_default_state:**

```
# cat /sys/devices/system/memory/auto_online_blocks
online
```

- **memory_hotplug.online_policy:**

```
# cat /sys/module/memory_hotplug/parameters/online_policy
auto-movable
```

- **memory_hotplug.auto_movable_ratio:**

```
# cat /sys/module/memory_hotplug/parameters/auto_movable_ratio
301
```

- **memory_hotplug.auto_movable_numa_aware:**

```
# cat /sys/module/memory_hotplug/parameters/auto_movable_numa_aware
y
```

Additional resources

- [Overview of virtio-mem](#)
- [Attaching a virtio-mem device to virtual machines](#)
- [Configuring Memory Hot\(Un\)Plug](#)

13.4.3.4. Attaching a virtio-mem device to virtual machines

To attach additional memory to a running virtual machine (also known as memory hot-plugging) and afterwards be able to resize the hot-plugged memory, you can use a **virtio-mem** device. Specifically, you can use libvirt XML configuration files and **virsh** commands to define and attach **virtio-mem** devices to virtual machines (VMs).



IMPORTANT

virtio-mem is included in RHEL 9 as a Technology Preview, which means it is not supported.

Prerequisites

- The host has Intel 64 or AMD64 CPU architecture.
- The host and VMs use RHEL 9 as the operating system.

- The VM has memory online configured. For instructions, see: [Configuring memory online in virtual machines](#)

Procedure

1. Ensure the XML configuration of the target VM includes the **maxMemory** parameter:

```
# virsh edit testguest1

<domain type='kvm'>
  <name>testguest1</name>
  ...
  <maxMemory slots='2' unit='GiB'>128</maxMemory>
  ...
</domain>
```

In this example, the XML configuration of the **testguest1** VM defines a **maxMemory** parameter with 2 slots and 128 gibibyte (GiB) size. The **maxMemory** size specifies the maximum memory the VM can use, which includes both initial and hot-plugged memory. Currently, you must reserve one slot for each attached **virtio-mem** device.

2. Ensure the XML configuration of the target VM has at least one NUMA node defined, for example:

```
# virsh edit testguest1

<domain type='kvm'>
  <name>testguest1</name>
  ...
  <vcpu placement='static'>8</vcpu>
  ...
  <cpu ...>
    <numa>
      <cell id='0' cpus='0-7' memory='16' unit='GiB'/>
    </numa>
  ...
</domain>
```

In this example, one NUMA node with 16 GiB of initial memory is defined and is assigned to 8 CPUs in the **testguest1** VM. To use memory devices in a VM, you must have at least one NUMA node defined.

3. Create and open an XML file to define **virtio-mem** devices on the host, for example:

```
# vim virtio-mem-device.xml
```

4. Add XML definitions of **virtio-mem** devices to the file and save it:

```
<memory model='virtio-mem'>
  <target>
    <size unit='GiB'>48</size>
    <node>0</node>
    <block unit='MiB'>2</block>
    <requested unit='GiB'>16</requested>
    <current unit='GiB'>16</current>
```

```

    </target>
    <alias name='ua-virtiomem0'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
</memory>
<memory model='virtio-mem'>
  <target>
    <size unit='GiB'>48</size>
    <node>1</node>
    <block unit='MiB'>2</block>
    <requested unit='GiB'>0</requested>
    <current unit='GiB'>0</current>
  </target>
  <alias name='ua-virtiomem1'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
</memory>

```

In this example, two **virtio-mem** devices are defined with the following parameters:

- **size**: This is the maximum size of the device. In the example, it is 48 GiB. The **size** must be a multiple of the **block** size.
 - **node**: This is the assigned vNUMA node for the **virtio-mem** device.
 - **block**: This is the block size of the device. It must be at least the size of the Transparent Huge Page (THP), which is 2 MiB on Intel 64 or AMD64 CPU architecture. The 2 MiB block size on Intel 64 or AMD64 architecture is usually a good default choice. When using **virtio-mem** with *Virtual Function I/O (VFIO)* or *mediated devices (mdev)*, the total number of blocks across all **virtio-mem** devices must not be larger than 32768, otherwise the plugging of RAM might fail.
 - **requested**: This is the amount of memory you attach to the VM with the **virtio-mem** device. However, it is just a request towards the VM and it might not be resolved successfully, for example if the VM is not properly configured. The **requested** size must be a multiple of the **block** size and cannot exceed the maximum defined **size**.
 - **current**: This represents the current size the **virtio-mem** device provides to the VM. The **current** size can differ from the **requested** size, for example when requests cannot be completed or when rebooting the VM.
 - **alias**: This is an optional user-defined alias that you can use to specify the intended **virtio-mem** device, for example when editing the device with libvirt commands. All user-defined aliases in libvirt must start with the "ua-" prefix.
Apart from these specific parameters, **libvirt** handles the **virtio-mem** device like any other PCI device.
5. Use the XML file to attach the defined **virtio-mem** devices to a VM. For example, to permanently attach the two devices defined in the **virtio-mem-device.xml** to the running **testguest1** VM:

```
# virsh attach-device testguest1 virtio-mem-device.xml --live --config
```

The **--live** option attaches the device to a running VM only, without persistence between boots. The **--config** option makes the configuration changes persistent. You can also attach the device to a shutdown VM without the **--live** option.

6. **Optional:** To dynamically change the **requested** size of a **virtio-mem** device attached to a running VM, use the **virsh update-memory-device** command:

```
# virsh update-memory-device testguest1 --alias ua-virtiomem0 --requested-size 4GiB
```

In this example:

- **testguest1** is the VM you want to update.
- **--alias ua-virtiomem0** is the **virtio-mem** device specified by a previously defined alias.
- **--requested-size 4GiB** changes the **requested** size of the **virtio-mem** device to 4 GiB.

Verification

- In the VM, check the available RAM and see if the total amount now includes the hot-plugged memory:

```
# free -h
```

```
      total   used   free   shared  buff/cache   available
Mem:   31Gi   5.5Gi  14Gi   1.3Gi   11Gi        23Gi
Swap:  8.0Gi   0B    8.0Gi
```

```
# numactl -H
```

```
available: 1 nodes (0)
node 0 cpus: 0 1 2 3 4 5 6 7
node 0 size: 29564 MB
node 0 free: 13351 MB
node distances:
node 0
 0: 10
```

- The current amount of plugged-in RAM can be also viewed on the host by displaying the XML configuration of the running VM:

```
# virsh dumpxml testguest1
```

```
<domain type='kvm'>
  <name>testguest1</name>
  ...
  <currentMemory unit='GiB'>31</currentMemory>
  ...
  <memory model='virtio-mem'>
    <target>
      <size unit='GiB'>48</size>
      <node>0</node>
      <block unit='MiB'>2</block>
      <requested unit='GiB'>16</requested>
      <current unit='GiB'>16</current>
    </target>
    <alias name='ua-virtiomem0'/>
  </memory>
</domain>
```

```

    <address type='pci' domain='0x0000' bus='0x08' slot='0x00' function='0x0' />
    ...
  </domain>

```

In this example:

- **<currentMemory unit='GiB'>31</currentMemory>** represents the total RAM available in the VM from all sources.
- **<current unit='GiB'>16</current>** represents the current size of the plugged-in RAM provided by the **virtio-mem** device.

Additional resources

- [Overview of virtio-mem](#)
- [Configuring memory online in virtual machines](#)

13.4.4. Additional resources

- Attaching devices to virtual machines [Attaching devices to virtual machines](#).

13.5. OPTIMIZING VIRTUAL MACHINE I/O PERFORMANCE

The input and output (I/O) capabilities of a virtual machine (VM) can significantly limit the VM's overall efficiency. To address this, you can optimize a VM's I/O by configuring block I/O parameters.

13.5.1. Tuning block I/O in virtual machines

When multiple block devices are being used by one or more VMs, it might be important to adjust the I/O priority of specific virtual devices by modifying their *I/O weights*.

Increasing the I/O weight of a device increases its priority for I/O bandwidth, and therefore provides it with more host resources. Similarly, reducing a device's weight makes it consume less host resources.



NOTE

Each device's **weight** value must be within the **100** to **1000** range. Alternatively, the value can be **0**, which removes that device from per-device listings.

Procedure

To display and set a VM's block I/O parameters:

1. Display the current **<blkio>** parameters for a VM:
virsh dumpxml VM-name

```

<domain>
[...]
```

```

  <blkiotune>
    <weight>800</weight>
  <device>
    <path>/dev/sda</path>
    <weight>1000</weight>
  </device>

```

```

    </device>
    <device>
      <path>/dev/sdb</path>
      <weight>500</weight>
    </device>
  </blkio tune>
  [...]
</domain>

```

2. Edit the I/O weight of a specified device:

```
# virsh blkio tune VM-name --device-weights device, I/O-weight
```

For example, the following changes the weight of the `/dev/sda` device in the `liftbrul` VM to 500.

```
# virsh blkio tune liftbrul --device-weights /dev/sda, 500
```

13.5.2. Disk I/O throttling in virtual machines

When several VMs are running simultaneously, they can interfere with system performance by using excessive disk I/O. Disk I/O throttling in KVM virtualization provides the ability to set a limit on disk I/O requests sent from the VMs to the host machine. This can prevent a VM from over-utilizing shared resources and impacting the performance of other VMs.

To enable disk I/O throttling, set a limit on disk I/O requests sent from each block device attached to VMs to the host machine.

Procedure

1. Use the **virsh domblklist** command to list the names of all the disk devices on a specified VM.

```
# virsh domblklist rollin-coal
Target    Source
-----
vda       /var/lib/libvirt/images/rollin-coal.qcow2
sda       -
sdb       /home/horridly-demanding-processes.iso
```

2. Find the host block device where the virtual disk that you want to throttle is mounted.
For example, if you want to throttle the **sdb** virtual disk from the previous step, the following output shows that the disk is mounted on the **/dev/nvme0n1p3** partition.

```
$ lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
zram0                              252:0    0   4G  0 disk  [SWAP]
nvme0n1                             259:0    0 238.5G  0 disk
├─nvme0n1p1                         259:1    0 600M  0 part  /boot/efi
├─nvme0n1p2                         259:2    0   1G  0 part  /boot
└─nvme0n1p3                         259:3    0 236.9G  0 part
   └─luks-a1123911-6f37-463c-b4eb-fxzy1ac12fea 253:0    0 236.9G  0 crypt /home
```

3. Set I/O limits for the block device using the **virsh blkio tune** command.

```
# virsh blkiotune VM-name --parameter device,limit
```

The following example throttles the **sdb** disk on the **rollin-coal** VM to 1000 read and write I/O operations per second and to 50 MB per second read and write throughput.

```
# virsh blkiotune rollin-coal --device-read-iops-sec /dev/nvme0n1p3,1000 --device-
write-iops-sec /dev/nvme0n1p3,1000 --device-write-bytes-sec
/dev/nvme0n1p3,52428800 --device-read-bytes-sec /dev/nvme0n1p3,52428800
```

Additional information

- Disk I/O throttling can be useful in various situations, for example when VMs belonging to different customers are running on the same host, or when quality of service guarantees are given for different VMs. Disk I/O throttling can also be used to simulate slower disks.
- I/O throttling can be applied independently to each block device attached to a VM and supports limits on throughput and I/O operations.
- Red Hat does not support using the **virsh blkdeviotune** command to configure I/O throttling in VMs. For more information about unsupported features when using RHEL 9 as a VM host, see [Unsupported features in RHEL 9 virtualization](#).

13.5.3. Enabling multi-queue virtio-scsi

When using **virtio-scsi** storage devices in your virtual machines (VMs), the *multi-queue virtio-scsi* feature provides improved storage performance and scalability. It enables each virtual CPU (vCPU) to have a separate queue and interrupt to use without affecting other vCPUs.

Procedure

- To enable multi-queue virtio-scsi support for a specific VM, add the following to the VM's XML configuration, where *N* is the total number of vCPU queues:

```
<controller type='scsi' index='0' model='virtio-scsi'>
  <driver queues='N' />
</controller>
```

13.6. OPTIMIZING VIRTUAL MACHINE CPU PERFORMANCE

Much like physical CPUs in host machines, vCPUs are critical to virtual machine (VM) performance. As a result, optimizing vCPUs can have a significant impact on the resource efficiency of your VMs. To optimize your vCPU:

1. Adjust how many host CPUs are assigned to the VM. You can do this using [the CLI](#) or [the web console](#).
2. Ensure that the vCPU model is aligned with the CPU model of the host. For example, to set the *testquest1* VM to use the CPU model of the host:

```
# virt-xml testquest1 --edit --cpu host-model
```

3. [Manage kernel same-page merging \(KSM\)](#).

4. If your host machine uses Non-Uniform Memory Access (NUMA), you can also **configure NUMA** for its VMs. This maps the host's CPU and memory processes onto the CPU and memory processes of the VM as closely as possible. In effect, NUMA tuning provides the vCPU with a more streamlined access to the system memory allocated to the VM, which can improve the vCPU processing effectiveness.

For details, see [Configuring NUMA in a virtual machine](#) and [Sample vCPU performance tuning scenario](#).

13.6.1. Adding and removing virtual CPUs using the command-line interface

To increase or optimize the CPU performance of a virtual machine (VM), you can add or remove virtual CPUs (vCPUs) assigned to the VM.

When performed on a running VM, this is also referred to as vCPU hot plugging and hot unplugging. However, note that vCPU hot unplug is not supported in RHEL 9, and Red Hat highly discourages its use.

Prerequisites

- **Optional:** View the current state of the vCPUs in the targeted VM. For example, to display the number of vCPUs on the *testquest* VM:

```
# virsh vcpucount testquest
maximum    config    4
maximum    live       2
current     config    2
current     live       1
```

This output indicates that *testquest* is currently using 1 vCPU, and 1 more vCPU can be hot plugged to it to increase the VM's performance. However, after reboot, the number of vCPUs *testquest* uses will change to 2, and it will be possible to hot plug 2 more vCPUs.

Procedure

1. Adjust the maximum number of vCPUs that can be attached to a VM, which takes effect on the VM's next boot.

For example, to increase the maximum vCPU count for the *testquest* VM to 8:

```
# virsh setvcpus testquest 8 --maximum --config
```

Note that the maximum may be limited by the CPU topology, host hardware, the hypervisor, and other factors.

2. Adjust the current number of vCPUs attached to a VM, up to the maximum configured in the previous step. For example:

- To increase the number of vCPUs attached to the running *testquest* VM to 4:

```
# virsh setvcpus testquest 4 --live
```

This increases the VM's performance and host load footprint of *testquest* until the VM's next boot.

- To permanently decrease the number of vCPUs attached to the *testquest* VM to 1:

```
# virsh setvcpus testquest 1 --config
```


■

This decreases the VM's performance and host load footprint of *testquest* after the VM's next boot. However, if needed, additional vCPUs can be hot plugged to the VM to temporarily increase its performance.

Verification

- Confirm that the current state of vCPU for the VM reflects your changes.

```
# virsh vcpucount testquest
maximum    config    8
maximum    live      4
current     config    1
current     live      4
```

Additional resources

- [Managing virtual CPUs using the web console](#)

13.6.2. Managing virtual CPUs using the web console

Using the RHEL 9 web console, you can review and configure virtual CPUs used by virtual machines (VMs) to which the web console is connected.

Prerequisites

- The web console VM plug-in [is installed on your system](#).

Procedure

1. In the **Virtual Machines** interface, click the VM whose information you want to see.
A new page opens with an Overview section with basic information about the selected VM and a Console section to access the VM's graphical interface.
2. Click **edit** next to the number of vCPUs in the Overview pane.
The vCPU details dialog appears.

Grid_v2 vCPU details

vCPU count ⓘ

2

Sockets ⓘ

1

vCPU maximum ⓘ

2

Cores per socket

1

Threads per core

1

Apply

Cancel

1. Configure the virtual CPUs for the selected VM.

- **vCPU Count** - The number of vCPUs currently in use.

**NOTE**

The vCPU count cannot be greater than the vCPU Maximum.

- **vCPU Maximum** – The maximum number of virtual CPUs that can be configured for the VM. If this value is higher than the **vCPU Count**, additional vCPUs can be attached to the VM.
- **Sockets** – The number of sockets to expose to the VM.
- **Cores per socket** – The number of cores for each socket to expose to the VM.
- **Threads per core** – The number of threads for each core to expose to the VM.
Note that the **Sockets**, **Cores per socket** and **Threads per core** options adjust the CPU topology of the VM. This may be beneficial for vCPU performance and may impact the functionality of certain software in the guest OS. If a different setting is not required by your deployment, keep the default values.

2. Click **Apply**.

The virtual CPUs for the VM are configured.

**NOTE**

Changes to virtual CPU settings only take effect after the VM is restarted.

Additional resources

- [Adding and removing virtual CPUs using the command-line interface](#)

13.6.3. Configuring NUMA in a virtual machine

The following methods can be used to configure Non-Uniform Memory Access (NUMA) settings of a virtual machine (VM) on a RHEL 9 host.

Prerequisites

- The host is a NUMA-compatible machine. To detect whether this is the case, use the **virsh nodeinfo** command and see the **NUMA cell(s)** line:

```
# virsh nodeinfo
CPU model:      x86_64
CPU(s):         48
CPU frequency:  1200 MHz
CPU socket(s):  1
Core(s) per socket: 12
Thread(s) per core: 2
NUMA cell(s):   2
Memory size:    67012964 KiB
```

If the value of the line is 2 or greater, the host is NUMA-compatible.

Procedure

For ease of use, you can set up a VM's NUMA configuration using automated utilities and services. However, manual NUMA setup is more likely to yield a significant performance improvement.

Automatic methods

- Set the VM's NUMA policy to **Preferred**. For example, to do so for the *testguest5* VM:

```
# virt-xml testguest5 --edit --vcpus placement=auto
# virt-xml testguest5 --edit --numatune mode=preferred
```

- Enable automatic NUMA balancing on the host:

```
# echo 1 > /proc/sys/kernel/numa_balancing
```

- Use the **numad** command to automatically align the VM CPU with memory resources.

```
# numad
```

Manual methods

1. Pin specific vCPU threads to a specific host CPU or range of CPUs. This is also possible on non-NUMA hosts and VMs, and is recommended as a safe method of vCPU performance improvement.

For example, the following commands pin vCPU threads 0 to 5 of the *testguest6* VM to host CPUs 1, 3, 5, 7, 9, and 11, respectively:

```
# virsh vcpupin testguest6 0 1
# virsh vcpupin testguest6 1 3
# virsh vcpupin testguest6 2 5
# virsh vcpupin testguest6 3 7
# virsh vcpupin testguest6 4 9
# virsh vcpupin testguest6 5 11
```

Afterwards, you can verify whether this was successful:

```
# virsh vcpupin testguest6
VCPU  CPU Affinity
-----
0     1
1     3
2     5
3     7
4     9
5    11
```

2. After pinning vCPU threads, you can also pin QEMU process threads associated with a specified VM to a specific host CPU or range of CPUs. For example, the following commands pin the QEMU process thread of *testguest6* to CPUs 13 and 15, and verify this was successful:

```
# virsh emulatorpin testguest6 13,15
# virsh emulatorpin testguest6
emulator: CPU Affinity
-----
*: 13,15
```

- Finally, you can also specify which host NUMA nodes will be assigned specifically to a certain VM. This can improve the host memory usage by the VM's vCPU. For example, the following commands set *testguest6* to use host NUMA nodes 3 to 5, and verify this was successful:

```
# virsh numatune testguest6 --nodeset 3-5
# virsh numatune testguest6
```



NOTE

For best performance results, it is recommended to use all of the manual tuning methods listed above

Known issues

- [NUMA tuning currently cannot be performed on IBM Z hosts](#) .

Additional resources

- [Sample vCPU performance tuning scenario](#)
- [View the current NUMA configuration of your system](#) using the **numastat** utility

13.6.4. Sample vCPU performance tuning scenario

To obtain the best vCPU performance possible, Red Hat recommends using manual **vcupin**, **emulatorpin**, and **numatune** settings together, for example like in the following scenario.

Starting scenario

- Your host has the following hardware specifics:
 - 2 NUMA nodes
 - 3 CPU cores on each node
 - 2 threads on each core

The output of **virsh nodeinfo** of such a machine would look similar to:

```
# virsh nodeinfo
CPU model:      x86_64
CPU(s):         12
CPU frequency:  3661 MHz
CPU socket(s):  2
Core(s) per socket: 3
Thread(s) per core: 2
NUMA cell(s):   2
Memory size:    31248692 KiB
```

- You intend to modify an existing VM to have 8 vCPUs, which means that it will not fit in a single NUMA node.
Therefore, you should distribute 4 vCPUs on each NUMA node and make the vCPU topology resemble the host topology as closely as possible. This means that vCPUs that run as sibling threads of a given physical CPU should be pinned to host threads on the same core. For details,

see the *Solution* below:

Solution

1. Obtain the information about the host topology:

```
# virsh capabilities
```

The output should include a section that looks similar to the following:

```
<topology>
<cells num="2">
  <cell id="0">
    <memory unit="KiB">15624346</memory>
    <pages unit="KiB" size="4">3906086</pages>
    <pages unit="KiB" size="2048">0</pages>
    <pages unit="KiB" size="1048576">0</pages>
    <distances>
      <sibling id="0" value="10" />
      <sibling id="1" value="21" />
    </distances>
    <cpus num="6">
      <cpu id="0" socket_id="0" core_id="0" siblings="0,3" />
      <cpu id="1" socket_id="0" core_id="1" siblings="1,4" />
      <cpu id="2" socket_id="0" core_id="2" siblings="2,5" />
      <cpu id="3" socket_id="0" core_id="0" siblings="0,3" />
      <cpu id="4" socket_id="0" core_id="1" siblings="1,4" />
      <cpu id="5" socket_id="0" core_id="2" siblings="2,5" />
    </cpus>
  </cell>
  <cell id="1">
    <memory unit="KiB">15624346</memory>
    <pages unit="KiB" size="4">3906086</pages>
    <pages unit="KiB" size="2048">0</pages>
    <pages unit="KiB" size="1048576">0</pages>
    <distances>
      <sibling id="0" value="21" />
      <sibling id="1" value="10" />
    </distances>
    <cpus num="6">
      <cpu id="6" socket_id="1" core_id="3" siblings="6,9" />
      <cpu id="7" socket_id="1" core_id="4" siblings="7,10" />
      <cpu id="8" socket_id="1" core_id="5" siblings="8,11" />
      <cpu id="9" socket_id="1" core_id="3" siblings="6,9" />
      <cpu id="10" socket_id="1" core_id="4" siblings="7,10" />
      <cpu id="11" socket_id="1" core_id="5" siblings="8,11" />
    </cpus>
  </cell>
</cells>
</topology>
```

2. **Optional:** Test the performance of the VM using [the applicable tools and utilities](#).
3. Set up and mount 1 GiB huge pages on the host:

- a. Add the following line to the host's kernel command line:

```
default_hugepagesz=1G hugepagesz=1G
```

- b. Create the **/etc/systemd/system/hugetlb-gigantic-pages.service** file with the following content:

```
[Unit]
Description=HugeTLB Gigantic Pages Reservation
DefaultDependencies=no
Before=dev-hugepages.mount
ConditionPathExists=/sys/devices/system/node
ConditionKernelCommandLine=hugepagesz=1G

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/etc/systemd/hugetlb-reserve-pages.sh

[Install]
WantedBy=sysinit.target
```

- c. Create the **/etc/systemd/hugetlb-reserve-pages.sh** file with the following content:

```
#!/bin/sh

nodes_path=/sys/devices/system/node/
if [ ! -d $nodes_path ]; then
    echo "ERROR: $nodes_path does not exist"
    exit 1
fi

reserve_pages()
{
    echo $1 > $nodes_path/$2/hugepages/hugepages-1048576kB/nr_hugepages
}

reserve_pages 4 node1
reserve_pages 4 node2
```

This reserves four 1GiB huge pages from *node1* and four 1GiB huge pages from *node2*.

- d. Make the script created in the previous step executable:

```
# chmod +x /etc/systemd/hugetlb-reserve-pages.sh
```

- e. Enable huge page reservation on boot:

```
# systemctl enable hugetlb-gigantic-pages
```

4. Use the **virsh edit** command to edit the XML configuration of the VM you wish to optimize, in this example *super-VM*:

```
# virsh edit super-vm
```

5. Adjust the XML configuration of the VM in the following way:

- a. Set the VM to use 8 static vCPUs. Use the `<vcpu/>` element to do this.
- b. Pin each of the vCPU threads to the corresponding host CPU threads that it mirrors in the topology. To do so, use the `<vcpupin/>` elements in the `<cputune>` section.
Note that, as shown by the **virsh capabilities** utility above, host CPU threads are not ordered sequentially in their respective cores. In addition, the vCPU threads should be pinned to the highest available set of host cores on the same NUMA node. For a table illustration, see the **Sample topology** section below.

The XML configuration for steps a. and b. can look similar to:

```
<cputune>
  <vcpupin vcpu='0' cpuset='1' />
  <vcpupin vcpu='1' cpuset='4' />
  <vcpupin vcpu='2' cpuset='2' />
  <vcpupin vcpu='3' cpuset='5' />
  <vcpupin vcpu='4' cpuset='7' />
  <vcpupin vcpu='5' cpuset='10' />
  <vcpupin vcpu='6' cpuset='8' />
  <vcpupin vcpu='7' cpuset='11' />
  <emulatorpin cpuset='6,9' />
</cputune>
```

- c. Set the VM to use 1 GiB huge pages:

```
<memoryBacking>
  <hugepages>
    <page size='1' unit='GiB' />
  </hugepages>
</memoryBacking>
```

- d. Configure the VM's NUMA nodes to use memory from the corresponding NUMA nodes on the host. To do so, use the `<memnode/>` elements in the `<numatune/>` section:

```
<numatune>
  <memory mode="preferred" nodeset="1" />
  <memnode cellid="0" mode="strict" nodeset="0" />
  <memnode cellid="1" mode="strict" nodeset="1" />
</numatune>
```

- e. Ensure the CPU mode is set to **host-passthrough**, and that the CPU uses cache in **passthrough** mode:

```
<cpu mode="host-passthrough">
  <topology sockets="2" cores="2" threads="2" />
  <cache mode="passthrough" />
</cpu>
```

Verification

1. Confirm that the resulting XML configuration of the VM includes a section similar to the following:

```
[...]
<memoryBacking>
  <hugepages>
    <page size='1' unit='GiB' />
  </hugepages>
</memoryBacking>
<vcpu placement='static'>8</vcpu>
<cputune>
  <vcpupin vcpu='0' cpuset='1' />
  <vcpupin vcpu='1' cpuset='4' />
  <vcpupin vcpu='2' cpuset='2' />
  <vcpupin vcpu='3' cpuset='5' />
  <vcpupin vcpu='4' cpuset='7' />
  <vcpupin vcpu='5' cpuset='10' />
  <vcpupin vcpu='6' cpuset='8' />
  <vcpupin vcpu='7' cpuset='11' />
  <emulatorpin cpuset='6,9' />
</cputune>
<numatune>
  <memory mode="preferred" nodeset="1" />
  <memnode cellid="0" mode="strict" nodeset="0" />
  <memnode cellid="1" mode="strict" nodeset="1" />
</numatune>
<cpu mode="host-passthrough">
  <topology sockets="2" cores="2" threads="2" />
  <cache mode="passthrough" />
  <numa>
    <cell id="0" cpus="0-3" memory="2" unit="GiB">
      <distances>
        <sibling id="0" value="10" />
        <sibling id="1" value="21" />
      </distances>
    </cell>
    <cell id="1" cpus="4-7" memory="2" unit="GiB">
      <distances>
        <sibling id="0" value="21" />
        <sibling id="1" value="10" />
      </distances>
    </cell>
  </numa>
</cpu>
</domain>
```

2. **Optional:** Test the performance of the VM using [the applicable tools and utilities](#) to evaluate the impact of the VM's optimization.

Sample topology

- The following tables illustrate the connections between the vCPUs and the host CPUs they should be pinned to:

Table 13.2. Host topology

CPU threads	0	3	1	4	2	5	6	9	7	10	8	11
-------------	---	---	---	---	---	---	---	---	---	----	---	----

Cores	0	1	2	3	4	5
Sockets	0			1		
NUMA nodes	0			1		

Table 13.3. VM topology

vCPU threads	0	1	2	3	4	5	6	7
Cores	0		1		2		3	
Sockets	0				1			
NUMA nodes	0				1			

Table 13.4. Combined host and VM topology

vCPU threads			0	1	2	3			4	5	6	7
Host CPU threads	0	3	1	4	2	5	6	9	7	10	8	11
Cores	0		1		2		3		4		5	
Sockets	0						1					
NUMA nodes	0						1					

In this scenario, there are 2 NUMA nodes and 8 vCPUs. Therefore, 4 vCPU threads should be pinned to each node.

In addition, Red Hat recommends leaving at least a single CPU thread available on each node for host system operations.

Because in this example, each NUMA node houses 3 cores, each with 2 host CPU threads, the set for node 0 translates as follows:

```
<vcpupin vcpu='0' cpuset='1'/>
<vcpupin vcpu='1' cpuset='4'/>
<vcpupin vcpu='2' cpuset='2'/>
<vcpupin vcpu='3' cpuset='5'/>
```

13.6.5. Managing kernel same-page merging

Kernel Same-Page Merging (KSM) improves memory density by sharing identical memory pages between virtual machines (VMs). However, enabling KSM increases CPU utilization, and might adversely affect overall performance depending on the workload.

Depending on your requirements, you can either enable or disable KSM for a single session or persistently.



NOTE

In RHEL 9 and later, KSM is disabled by default.

Prerequisites

- Root access to your host system.

Procedure

- Disable KSM:
 - To deactivate KSM for a single session, use the **systemctl** utility to stop **ksm** and **ksmtuned** services.
- To deactivate KSM persistently, use the **systemctl** utility to disable **ksm** and **ksmtuned** services.

```
# systemctl stop ksm
```

```
# systemctl stop ksmtuned
```

```
# systemctl disable ksm
```

```
Removed /etc/systemd/system/multi-user.target.wants/ksm.service.
```

```
# systemctl disable ksmtuned
```

```
Removed /etc/systemd/system/multi-user.target.wants/ksmtuned.service.
```



NOTE

Memory pages shared between VMs before deactivating KSM will remain shared. To stop sharing, delete all the **PageKSM** pages in the system using the following command:

```
# echo 2 > /sys/kernel/mm/ksm/run
```

After anonymous pages replace the KSM pages, the **khugepaged** kernel service will rebuild transparent hugepages on the VM's physical memory.

- Enable KSM:



WARNING

Enabling KSM increases CPU utilization and affects overall CPU performance.

1. Install the **ksmtuned** service:
dnf install ksmtuned

2. Start the service:

- To enable KSM for a single session, use the **systemctl** utility to start the **kvm** and **ksmtuned** services.

```
# systemctl start kvm
# systemctl start ksmtuned
```

- To enable KSM persistently, use the **systemctl** utility to enable the **kvm** and **ksmtuned** services.

```
# systemctl enable kvm
Created symlink /etc/systemd/system/multi-user.target.wants/kvm.service →
/usr/lib/systemd/system/kvm.service

# systemctl enable ksmtuned
Created symlink /etc/systemd/system/multi-user.target.wants/ksmtuned.service →
/usr/lib/systemd/system/ksmtuned.service
```

13.7. OPTIMIZING VIRTUAL MACHINE NETWORK PERFORMANCE

Due to the virtual nature of a VM's network interface card (NIC), the VM loses a portion of its allocated host network bandwidth, which can reduce the overall workload efficiency of the VM. The following tips can minimize the negative impact of virtualization on the virtual NIC (vNIC) throughput.

Procedure

Use any of the following methods and observe if it has a beneficial effect on your VM network performance:

Enable the vhost_net module

On the host, ensure the **vhost_net** kernel feature is enabled:

```
# lsmod | grep vhost
vhost_net      32768  1
vhost          53248  1 vhost_net
tap            24576  1 vhost_net
tun            57344  6 vhost_net
```

If the output of this command is blank, enable the **vhost_net** kernel module:

```
# modprobe vhost_net
```

Set up multi-queue virtio-net

To set up the *multi-queue virtio-net* feature for a VM, use the **virsh edit** command to edit to the XML configuration of the VM. In the XML, add the following to the **<devices>** section, and replace **N** with the number of vCPUs in the VM, up to 16:

```
<interface type='network'>
  <source network='default'/>
  <model type='virtio'/>
  <driver name='vhost' queues='N'/>
</interface>
```

If the VM is running, restart it for the changes to take effect.

Batching network packets

In Linux VM configurations with a long transmission path, batching packets before submitting them to the kernel may improve cache utilization. To set up packet batching, use the following command on the host, and replace *tap0* with the name of the network interface that the VMs use:

```
# ethtool -C tap0 rx-frames 64
```

SR-IOV

If your host NIC supports SR-IOV, use SR-IOV device assignment for your vNICs. For more information, see [Managing SR-IOV devices](#).

Additional resources

- [Understanding virtual networking](#)

13.8. VIRTUAL MACHINE PERFORMANCE MONITORING TOOLS

To identify what consumes the most VM resources and which aspect of VM performance needs optimization, performance diagnostic tools, both general and VM-specific, can be used.

Default OS performance monitoring tools

For standard performance evaluation, you can use the utilities provided by default by your host and guest operating systems:

- On your RHEL 9 host, as root, use the **top** utility or the **system monitor** application, and look for **qemu** and **virt** in the output. This shows how much host system resources your VMs are consuming.
 - If the monitoring tool displays that any of the **qemu** or **virt** processes consume a large portion of the host CPU or memory capacity, use the **perf** utility to investigate. For details, see below.
 - In addition, if a **vhost_net** thread process, named for example *vhost_net-1234*, is displayed as consuming an excessive amount of host CPU capacity, consider using [virtual network optimization features](#), such as **multi-queue virtio-net**.
- On the guest operating system, use performance utilities and applications available on the system to evaluate which processes consume the most system resources.
 - On Linux systems, you can use the **top** utility.
 - On Windows systems, you can use the **Task Manager** application.

perf kvm

You can use the **perf** utility to collect and analyze virtualization-specific statistics about the performance of your RHEL 9 host. To do so:

1. On the host, install the *perf* package:

```
# dnf install perf
```

- Use one of the **perf kvm stat** commands to display perf statistics for your virtualization host:
 - For real-time monitoring of your hypervisor, use the **perf kvm stat live** command.
 - To log the perf data of your hypervisor over a period of time, activate the logging using the **perf kvm stat record** command. After the command is canceled or interrupted, the data is saved in the **perf.data.guest** file, which can be analyzed using the **perf kvm stat report** command.
- Analyze the **perf** output for types of **VM-EXIT** events and their distribution. For example, the **PAUSE_INSTRUCTION** events should be infrequent, but in the following output, the high occurrence of this event suggests that the host CPUs are not handling the running vCPUs well. In such a scenario, consider shutting down some of your active VMs, removing vCPUs from these VMs, or [tuning the performance of the vCPUs](#).

perf kvm stat report

Analyze events for all VMs, all VCPUs:

VM-EXIT	Samples	Samples%	Time%	Min Time	Max Time	Avg time
EXTERNAL_INTERRUPT	365634	31.59%	18.04%	0.42us	58780.59us	204.08us (+- 0.99%)
MSR_WRITE	293428	25.35%	0.13%	0.59us	17873.02us	1.80us (+- 4.63%)
PREEMPTION_TIMER	276162	23.86%	0.23%	0.51us	21396.03us	3.38us (+- 5.19%)
PAUSE_INSTRUCTION	189375	16.36%	11.75%	0.72us	29655.25us	256.77us (+- 0.70%)
HLT	20440	1.77%	69.83%	0.62us	79319.41us	14134.56us (+- 0.79%)
VMCALL	12426	1.07%	0.03%	1.02us	5416.25us	8.77us (+- 7.36%)
EXCEPTION_NMI	27	0.00%	0.00%	0.69us	1.34us	0.98us (+- 3.50%)
EPT_MISCONFIG	5	0.00%	0.00%	5.15us	10.85us	7.88us (+- 11.67%)

Total Samples:1157497, Total events handled time:413728274.66us.

Other event types that can signal problems in the output of **perf kvm stat** include:

- INSN_EMULATION** - suggests suboptimal [VM I/O configuration](#).

For more information about using **perf** to monitor virtualization performance, see the **perf-kvm** man page.

numastat

To see the current NUMA configuration of your system, you can use the **numastat** utility, which is provided by installing the **numactl** package.

The following shows a host with 4 running VMs, each obtaining memory from multiple NUMA nodes. This is not optimal for vCPU performance, and [warrants adjusting](#):

```
# numastat -c qemu-kvm
```

Per-node process memory usage (in MBs)

PID	Node 0	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Node 7	Total
51722 (qemu-kvm)	68	16	357	6936	2	3	147	598	8128
51747 (qemu-kvm)	245	11	5	18	5172	2532	1	92	8076
53736 (qemu-kvm)	62	432	1661	506	4851	136	22	445	8116
53773 (qemu-kvm)	1393	3	1	2	12	0	0	6702	8114
Total	1769	463	2024	7462	10037	2672	169	7837	32434

In contrast, the following shows memory being provided to each VM by a single node, which is significantly more efficient.

numastat -c qemu-kvm

Per-node process memory usage (in MBs)

PID	Node 0	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Node 7	Total
51747 (qemu-kvm)	0	0	7	0	8072	0	1	0	8080
53736 (qemu-kvm)	0	0	7	0	0	0	8113	0	8120
53773 (qemu-kvm)	0	0	7	0	0	0	1	8110	8118
59065 (qemu-kvm)	0	0	8050	0	0	0	0	0	8051
Total	0	0	8072	0	8072	0	8114	8110	32368

13.9. ADDITIONAL RESOURCES

- [Optimizing Windows virtual machines](#)

CHAPTER 14. IMPORTANCE OF POWER MANAGEMENT

Reducing the overall power consumption of computer systems helps to save cost. Effectively optimizing energy consumption of each system component includes studying different tasks that your system performs, and configuring each component to ensure that its performance is correct for that job. Lowering the power consumption of a specific component or of the system as a whole leads to lower heat and performance.

Proper power management results in:

- heat reduction for servers and computing centers
- reduced secondary costs, including cooling, space, cables, generators, and uninterruptible power supplies (UPS)
- extended battery life for laptops
- lower carbon dioxide output
- meeting government regulations or legal requirements regarding Green IT, for example, Energy Star
- meeting company guidelines for new systems

This section describes the information regarding power management of your Red Hat Enterprise Linux systems.

14.1. POWER MANAGEMENT BASICS

Effective power management is built on the following principles:

An idle CPU should only wake up when needed

Since Red Hat Enterprise Linux 6, the kernel runs **tickless**, which means the previous periodic timer interrupts have been replaced with on-demand interrupts. Therefore, idle CPUs are allowed to remain idle until a new task is queued for processing, and CPUs that have entered lower power states can remain in these states longer. However, benefits from this feature can be offset if your system has applications that create unnecessary timer events. Polling events, such as checks for volume changes or mouse movement, are examples of such events.

Red Hat Enterprise Linux includes tools using which you can identify and audit applications on the basis of their CPU usage. For more information see, [Audit and analysis overview](#) and [Tools for auditing](#).

Unused hardware and devices should be disabled completely

This is true for devices that have moving parts, for example, hard disks. In addition to this, some applications may leave an unused but enabled device "open"; when this occurs, the kernel assumes that the device is in use, which can prevent the device from going into a power saving state.

Low activity should translate to low wattage

In many cases, however, this depends on modern hardware and correct BIOS configuration or UEFI on modern systems, including non-x86 architectures. Make sure that you are using the latest official firmware for your systems and that in the power management or device configuration sections of the BIOS the power management features are enabled. Some features to look for include:

- Collaborative Processor Performance Controls (CPPC) support for ARM64

- PowerNV support for IBM Power Systems
- SpeedStep
- PowerNow!
- Cool'n'Quiet
- ACPI (C-state)
- Smart

If your hardware has support for these features and they are enabled in the BIOS, Red Hat Enterprise Linux uses them by default.

Different forms of CPU states and their effects

Modern CPUs together with Advanced Configuration and Power Interface (ACPI) provide different power states. The three different states are:

- Sleep (C-states)
- Frequency and voltage (P-states)
- Heat output (T-states or thermal states)

A CPU running on the lowest sleep state, consumes the least amount of watts, but it also takes considerably more time to wake it up from that state when needed. In very rare cases this can lead to the CPU having to wake up immediately every time it just went to sleep. This situation results in an effectively permanently busy CPU and loses some of the potential power saving if another state had been used.

A turned off machine uses the least amount of power

One of the best ways to save power is to turn off systems. For example, your company can develop a corporate culture focused on "green IT" awareness with a guideline to turn off machines during lunch break or when going home. You also might consolidate several physical servers into one bigger server and virtualize them using the virtualization technology, which is shipped with Red Hat Enterprise Linux.

14.2. AUDIT AND ANALYSIS OVERVIEW

The detailed manual audit, analysis, and tuning of a single system is usually the exception because the time and cost spent to do so typically outweighs the benefits gained from these last pieces of system tuning.

However, performing these tasks once for a large number of nearly identical systems where you can reuse the same settings for all systems can be very useful. For example, consider the deployment of thousands of desktop systems, or an HPC cluster where the machines are nearly identical. Another reason to do auditing and analysis is to provide a basis for comparison against which you can identify regressions or changes in system behavior in the future. The results of this analysis can be very helpful in cases where hardware, BIOS, or software updates happen regularly and you want to avoid any surprises with regard to power consumption. Generally, a thorough audit and analysis gives you a much better idea of what is really happening on a particular system.

Auditing and analyzing a system with regard to power consumption is relatively hard, even with the most modern systems available. Most systems do not provide the necessary means to measure power use via software. Exceptions exist though:

- iLO management console of Hewlett Packard server systems has a power management module that you can access through the web.
- IBM provides a similar solution in their BladeCenter power management module.
- On some Dell systems, the IT Assistant offers power monitoring capabilities as well.

Other vendors are likely to offer similar capabilities for their server platforms, but as can be seen there is no single solution available that is supported by all vendors. Direct measurements of power consumption are often only necessary to maximize savings as far as possible.

14.3. TOOLS FOR AUDITING

Red Hat Enterprise Linux 8 offers tools using which you can perform system auditing and analysis. Most of them can be used as supplementary sources of information in case you want to verify what you have discovered already or in case you need more in-depth information about certain parts.

Many of these tools are used for performance tuning as well, which include:

PowerTOP

It identifies specific components of kernel and user-space applications that frequently wake up the CPU. Use the **powertop** command as root to start the **PowerTop** tool and **powertop --calibrate** to calibrate the power estimation engine. For more information about PowerTop, see [Managing power consumption with PowerTOP](#).

Diskdevstat and netdevstat

They are SystemTap tools that collect detailed information about the disk activity and network activity of all applications running on a system. Using the collected statistics by these tools, you can identify applications that waste power with many small I/O operations rather than fewer, larger operations. Using the **dnf install tuned-utils-systemtap kernel-debuginfo** command as root, install the **diskdevstat** and **netdevstat** tool.

To view the detailed information about the disk and network activity, use:

```
# diskdevstat
```

```
PID UID DEV WRITE_CNT WRITE_MIN WRITE_MAX WRITE_AVG READ_CNT
READ_MIN READ_MAX READ_AVG COMMAND
3575 1000 dm-2 59 0.000 0.365 0.006 5 0.000 0.000 0.000
mozStorage #5
3575 1000 dm-2 7 0.000 0.000 0.000 0 0.000 0.000 0.000
localStorage DB
[...]
```

```
# netdevstat
```

```
PID UID DEV XMIT_CNT XMIT_MIN XMIT_MAX XMIT_AVG RECV_CNT
RECV_MIN RECV_MAX RECV_AVG COMMAND
3572 991 enp0s31f6 40 0.000 0.882 0.108 0 0.000 0.000 0.000
openvpn
3575 1000 enp0s31f6 27 0.000 1.363 0.160 0 0.000 0.000 0.000
Socket Thread
[...]
```

With these commands, you can specify three parameters: **update_interval**, **total_duration**, and **display_histogram**.

TuneD

It is a profile-based system tuning tool that uses the **udev** device manager to monitor connected devices, and enables both static and dynamic tuning of system settings. You can use the **tuned-adm recommend** command to determine which profile Red Hat recommends as the most suitable for a particular product. For more information about TuneD, see [Getting started with TuneD](#) and [Customizing TuneD profiles](#). Using the **powertop2tuned** utility, you can create custom TuneD profiles from **PowerTOP** suggestions. For information about the **powertop2tuned** utility, see [Optimizing power consumption](#).

Virtual memory statistics (vmstat)

It is provided by the **procps-ng** package. Using this tool, you can view the detailed information about processes, memory, paging, block I/O, traps, and CPU activity.

To view this information, use:

```
$ vmstat
procs -----memory----- ---swap-- -----io-----system-- -----cpu-----
 r b swpd free  buff  cache  si  so bi  bo  in cs us sy id wa st
 1 0 0 5805576 380856 4852848 0 0 119 73 814 640 2 2 96 0 0
```

Using the **vmstat -a** command, you can display active and inactive memory. For more information about other **vmstat** options, see the **vmstat** man page.

iostat

It is provided by the **sysstat** package. This tool is similar to **vmstat**, but only for monitoring I/O on block devices. It also provides more verbose output and statistics.

To monitor the system I/O, use:

```
$ iostat
avg-cpu: %user  %nice %system %iowait  %steal   %idle
          2.05   0.46   1.55   0.26   0.00  95.67

Device  tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
nvme0n1 53.54     899.48     616.99    3445229    2363196
dm-0    42.84     753.72     238.71    2886921     914296
dm-1     0.03       0.60       0.00      2292         0
dm-2    24.15     143.12     379.80    548193    1454712
```

blktrace

It provides detailed information about how time is spent in the I/O subsystem.

To view this information in human readable format, use:

```
# blktrace -d /dev/dm-0 -o - | blkparse -i -

253,0 1 1 0.000000000 17694 Q W 76423384 + 8 [kworker/u16:1]
253,0 2 1 0.001926913 0 C W 76423384 + 8 [0]
[...]
```

Here, The first column, **253,0** is the device major and minor tuple. The second column, **1**, gives information about the CPU, followed by columns for timestamps and PID of the process issuing the IO process.

The sixth column, **Q**, shows the event type, the 7th column, **W** for write operation, the 8th column, **76423384**, is the block number, and the **+ 8** is the number of requested blocks.

The last field, **[kworker/u16:1]**, is the process name.

By default, the **blktrace** command runs forever until the process is explicitly killed. Use the **-w** option to specify the run-time duration.

turbostat

It is provided by the **kernel-tools** package. It reports on processor topology, frequency, idle power-state statistics, temperature, and power usage on x86-64 processors.

To view this summary, use:

```
# turbostat

CUID(0): GenuineIntel 0x16 CUID levels; 0x80000008 xlevels; family:model:stepping 0x6:8e:a
(6:142:10)
CUID(1): SSE3 MONITOR SMX EIST TM2 TSC MSR ACPI-TM HT TM
CUID(6): APERF, TURBO, DTS, PTM, HWP, HWPnotify, HWPwindow, HWPcapp, No-HWPpkg,
EPB
[...]
```

By default, **turbostat** prints a summary of counter results for the entire screen, followed by counter results every 5 seconds. Specify a different period between counter results with the **-i** option, for example, execute **turbostat -i 10** to print results every 10 seconds instead.

Turbostat is also useful for identifying servers that are inefficient in terms of power usage or idle time. It also helps to identify the rate of system management interrupts (SMIs) occurring on the system. It can also be used to verify the effects of power management tuning.

cpupower

It is a collection of tools to examine and tune power saving related features of processors. Use the **cpupower** command with the **frequency-info**, **frequency-set**, **idle-info**, **idle-set**, **set**, **info**, and **monitor** options to display and set processor related values.

For example, to view available cpufreq governors, use:

```
$ cpupower frequency-info --governors
analyzing CPU 0:
available cpufreq governors: performance powersave
```

For more information about **cpupower**, see Viewing CPU related information.

GNOME Power Manager

It is a daemon that is installed as part of the GNOME desktop environment. GNOME Power Manager notifies you of changes in your system's power status; for example, a change from battery to AC power. It also reports battery status, and warns you when battery power is low.

Additional resources

- **powertop(1)**, **diskdevstat(8)**, **netdevstat(8)**, **tuned(8)**, **vmstat(8)**, **iostat(1)**, **blktrace(8)**, **blkparse(8)**, and **turbostat(8)** man pages

- **cpupower(1)**, **cpupower-set(1)**, **cpupower-info(1)**, **cpupower-idle(1)**, **cpupower-frequency-set(1)**, **cpupower-frequency-info(1)**, and **cpupower-monitor(1)** man pages

CHAPTER 15. MANAGING POWER CONSUMPTION WITH POWERTOP

As a system administrator, you can use the **PowerTOP** tool to analyze and manage power consumption.

15.1. THE PURPOSE OF POWERTOP

PowerTOP is a program that diagnoses issues related to power consumption and provides suggestions on how to extend battery lifetime.

The **PowerTOP** tool can provide an estimate of the total power usage of the system and also individual power usage for each process, device, kernel worker, timer, and interrupt handler. The tool can also identify specific components of kernel and user-space applications that frequently wake up the CPU.

Red Hat Enterprise Linux 9 uses version 2.x of **PowerTOP**.

15.2. USING POWERTOP

Prerequisites

- To be able to use **PowerTOP**, make sure that the **powertop** package has been installed on your system:

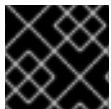
```
# dnf install powertop
```

15.2.1. Starting PowerTOP

Procedure

- To run **PowerTOP**, use the following command:

```
# powertop
```



IMPORTANT

Laptops should run on battery power when running the **powertop** command.

15.2.2. Calibrating PowerTOP

Procedure

1. On a laptop, you can calibrate the power estimation engine by running the following command:

```
# powertop --calibrate
```

2. Let the calibration finish without interacting with the machine during the process. Calibration takes time because the process performs various tests, cycles through brightness levels and switches devices on and off.

3. When the calibration process is completed, **PowerTOP** starts as normal. Let it run for approximately an hour to collect data.
When enough data is collected, power estimation figures will be displayed in the first column of the output table.

**NOTE**

Note that **powertop --calibrate** can only be used on laptops.

15.2.3. Setting the measuring interval

By default, **PowerTOP** takes measurements in 20 seconds intervals.

If you want to change this measuring frequency, use the following procedure:

Procedure

- Run the **powertop** command with the **--time** option:

```
# powertop --time=time in seconds
```

15.2.4. Additional resources

For more details on how to use **PowerTOP**, see the **powertop** man page.

15.3. POWERTOP STATISTICS

While it runs, **PowerTOP** gathers statistics from the system.

PowerTOP's output provides multiple tabs:

- **Overview**
- **Idle stats**
- **Frequency stats**
- **Device stats**
- **Tunables**
- **WakeUp**

You can use the **Tab** and **Shift+Tab** keys to cycle through these tabs.

15.3.1. The Overview tab

In the **Overview** tab, you can view a list of the components that either send wakeups to the CPU most frequently or consume the most power. The items within the **Overview** tab, including processes, interrupts, devices, and other resources, are sorted according to their utilization.

The adjacent columns within the **Overview** tab provide the following pieces of information:

Usage

Power estimation of how the resource is being used.

Events/s

Wakeups per second. The number of wakeups per second indicates how efficiently the services or the devices and drivers of the kernel are performing. Less wakeups means that less power is consumed. Components are ordered by how much further their power usage can be optimized.

Category

Classification of the component; such as process, device, or timer.

Description

Description of the component.

If properly calibrated, a power consumption estimation for every listed item in the first column is shown as well.

Apart from this, the **Overview** tab includes the line with summary statistics such as:

- Total power consumption
- Remaining battery life (only if applicable)
- Summary of total wakeups per second, GPU operations per second, and virtual file system operations per second

15.3.2. The Idle stats tab

The **Idle stats** tab shows usage of C-states for all processors and cores, while the **Frequency stats** tab shows usage of P-states including the Turbo mode, if applicable, for all processors and cores. The duration of C- or P-states is an indication of how well the CPU usage has been optimized. The longer the CPU stays in the higher C- or P-states (for example C4 is higher than C3), the better the CPU usage optimization is. Ideally, residency is 90% or more in the highest C- or P-state when the system is idle.

15.3.3. The Device stats tab

The **Device stats** tab provides similar information to the **Overview** tab but only for devices.

15.3.4. The Tunables tab

The **Tunables** tab contains **PowerTOP**'s suggestions for optimizing the system for lower power consumption.

Use the **up** and **down** keys to move through suggestions, and the **enter** key to toggle the suggestion on or off.

15.3.5. The WakeUp tab

The **WakeUp** tab displays the device wakeup settings available for users to change as and when required.

Use the **up** and **down** keys to move through the available settings, and the **enter** key to enable or disable a setting.

Figure 15.1. PowerTOP output

PowerTOP 2.14

Overview

Idle stats

Frequency stats

Device stats

Tunables

WakeUp

Summary: 164.7 wakeups/second, 0.0 GPU ops/seconds, 0.0 VFS ops/sec and 6.1% CPU use

Usage	Events/s	Category	Description
100.0%		Device	Audio codec hwC0D0: QEMU
46.1 ms/s	47.2	Process	[PID 1785] /usr/bin/gnome-shell
1.6 ms/s	27.9	Timer	tick_sched timer
424.7 µs/s	18.3	Process	[PID 671] [xfsaild/dm-0]
181.4 µs/s	15.4	Process	[PID 11] [rcu_sched]
680.8 µs/s	7.7	Interrupt	[7] sched(softirq)
261.6 µs/s	5.8	Timer	hrtimer wakeup
261.2 µs/s	5.8	Process	[PID 3745] /usr/libexec/gsd-smartcard
2.9 ms/s	3.9	Process	[PID 6584] /usr/libexec/gnome-terminal-server
43.1 µs/s	3.9	Timer	watchdog timer_fn
578.4 µs/s	2.9	Process	[PID 4303] /usr/libexec/platform-python /usr/libexec/rhsm-service
251.0 µs/s	2.9	kWork	commit_work
55.1 µs/s	2.9	kWork	virtio_gpu_dequeue_ctrl_func
4.1 ms/s	1.0	Process	[PID 9655] powertop
14.3 µs/s	1.9	kWork	gc_worker
8.0 µs/s	1.9	kWork	kfree_rcu_work
230.3 µs/s	1.0	Process	[PID 1521] /usr/libexec/platform-python -Es /usr/sbin/tuned -l -P

<ESC> Exit | <TAB> / <Shift + TAB> Navigate |

Additional resources

For more details on **PowerTOP**, see [PowerTOP's home page](#).

15.4. WHY POWERTOP DOES NOT DISPLAY FREQUENCY STATS VALUES IN SOME INSTANCES

While using the Intel P-State driver, PowerTOP only displays values in the **Frequency Stats** tab if the driver is in passive mode. But, even in this case, the values may be incomplete.

In total, there are three possible modes of the Intel P-State driver:

- Active mode with Hardware P-States (HWP)
- Active mode without HWP
- Passive mode

Switching to the ACPI CPUfreq driver results in complete information being displayed by PowerTOP. However, it is recommended to keep your system on the default settings.

To see what driver is loaded and in what mode, run:

```
# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_driver
```

- **intel_pstate** is returned if the Intel P-State driver is loaded and in active mode.
- **intel_cpufreq** is returned if the Intel P-State driver is loaded and in passive mode.
- **acpi-cpufreq** is returned if the ACPI CPUfreq driver is loaded.

While using the Intel P-State driver, add the following argument to the kernel boot command line to force the driver to run in passive mode:

```
intel_pstate=passive
```


To disable the Intel P-State driver and use, instead, the ACPI CPUfreq driver, add the following argument to the kernel boot command line:

```
intel_pstate=disable
```

15.5. GENERATING AN HTML OUTPUT

Apart from the **powertop**'s output in terminal, you can also generate an HTML report.

Procedure

- Run the **powertop** command with the **--html** option:

```
# powertop --html=htmlfile.html
```

Replace the **htmlfile.html** parameter with the required name for the output file.

15.6. OPTIMIZING POWER CONSUMPTION

To optimize power consumption, you can use either the **powertop** service or the **powertop2tuned** utility.

15.6.1. Optimizing power consumption using the powertop service

You can use the **powertop** service to automatically enable all **PowerTOP**'s suggestions from the **Tunables** tab on the boot:

Procedure

- Enable the **powertop** service:

```
# systemctl enable powertop
```

15.6.2. The powertop2tuned utility

The **powertop2tuned** utility allows you to create custom **TuneD** profiles from **PowerTOP** suggestions.

By default, **powertop2tuned** creates profiles in the **/etc/tuned/** directory, and bases the custom profile on the currently selected **TuneD** profile. For safety reasons, all **PowerTOP** tunings are initially disabled in the new profile.

To enable the tunings, you can:

- Uncomment them in the **/etc/tuned/profile_name/tuned.conf** file.
- Use the **--enable** or **-e** option to generate a new profile that enables most of the tunings suggested by **PowerTOP**.
Certain potentially problematic tunings, such as the USB autosuspend, are disabled by default and need to be uncommented manually.

15.6.3. Optimizing power consumption using the powertop2tuned utility

Prerequisites

- The **powertop2tuned** utility is installed on the system:

```
# dnf install tuned-utils
```

Procedure

1. Create a custom profile:

```
# powertop2tuned new_profile_name
```

2. Activate the new profile:

```
# tuned-adm profile new_profile_name
```

Additional information

- For a complete list of options that **powertop2tuned** supports, use:

```
$ powertop2tuned --help
```

15.6.4. Comparison of **powertop.service** and **powertop2tuned**

Optimizing power consumption with **powertop2tuned** is preferred over **powertop.service** for the following reasons:

- The **powertop2tuned** utility represents integration of **PowerTOP** into **TuneD**, which enables to benefit of advantages of both tools.
- The **powertop2tuned** utility allows for fine-grained control of enabled tuning.
- With **powertop2tuned**, potentially dangerous tuning are not automatically enabled.
- With **powertop2tuned**, rollback is possible without reboot.

CHAPTER 16. GETTING STARTED WITH PERF

As a system administrator, you can use the **perf** tool to collect and analyze performance data of your system.

16.1. INTRODUCTION TO PERF

The **perf** user-space tool interfaces with the kernel-based subsystem *Performance Counters for Linux* (PCL). **perf** is a powerful tool that uses the Performance Monitoring Unit (PMU) to measure, record, and monitor a variety of hardware and software events. **perf** also supports tracepoints, kprobes, and uprobes.

16.2. INSTALLING PERF

This procedure installs the **perf** user-space tool.

Procedure

- Install the **perf** tool:

```
# dnf install perf
```

16.3. COMMON PERF COMMANDS

perf stat

This command provides overall statistics for common performance events, including instructions executed and clock cycles consumed. Options allow for selection of events other than the default measurement events.

perf record

This command records performance data into a file, **perf.data**, which can be later analyzed using the **perf report** command.

perf report

This command reads and displays the performance data from the **perf.data** file created by **perf record**.

perf list

This command lists the events available on a particular machine. These events will vary based on performance monitoring hardware and software configuration of the system.

perf top

This command performs a similar function to the **top** utility. It generates and displays a performance counter profile in realtime.

perf trace

This command performs a similar function to the **strace** tool. It monitors the system calls used by a specified thread or process and all signals received by that application.

perf help

This command displays a complete list of **perf** commands.

Additional resources

- Add the **--help** option to a subcommand to open the man page.

CHAPTER 17. PROFILING CPU USAGE IN REAL TIME WITH PERF TOP

You can use the **perf top** command to measure CPU usage of different functions in real time.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).

17.1. THE PURPOSE OF PERF TOP

The **perf top** command is used for real time system profiling and functions similarly to the **top** utility. However, where the **top** utility generally shows you how much CPU time a given process or thread is using, **perf top** shows you how much CPU time each specific function uses. In its default state, **perf top** tells you about functions being used across all CPUs in both the user-space and the kernel-space. To use **perf top** you need root access.

17.2. PROFILING CPU USAGE WITH PERF TOP

This procedure activates **perf top** and profiles CPU usage in real time.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).
- You have root access

Procedure

- Start the **perf top** monitoring interface:

```
# perf top
```

The monitoring interface looks similar to the following:

```
Samples: 8K of event 'cycles', 2000 Hz, Event count (approx.): 4579432780 lost: 0/0 drop: 0/0
Overhead Shared Object    Symbol
 2.20% [kernel]           [k] do_syscall_64
 2.17% [kernel]           [k] module_get_kallsym
 1.49% [kernel]           [k] copy_user_enhanced_fast_string
 1.37% libpthread-2.29.so  [...] pthread_mutex_lock 1.31% [unknown] [...] 0000000000000000
 1.07% [kernel] [k] psi_task_change 1.04% [kernel] [k] switch_mm_irqs_off 0.94% [kernel] [k]
fget
 0.74% [kernel]           [k] entry_SYSCALL_64
 0.69% [kernel]           [k] syscall_return_via_sysret
 0.69% libxul.so           [...] 0x000000000113f9b0
 0.67% [kernel]           [k] kallsyms_expand_symbol.constprop.0
 0.65% firefox             [...] moz_xmalloc
 0.65% libpthread-2.29.so  [...] __pthread_mutex_unlock_usercnt
 0.60% firefox             [...] free
 0.60% libxul.so           [...] 0x0000000000241d1cd
 0.60% [kernel]           [k] do_sys_poll
```

0.58%	[kernel]	[k] menu_select
0.56%	[kernel]	[k] _raw_spin_lock_irqsave
0.55%	perf	[.] 0x00000000002ae0f3

In this example, the kernel function **do_syscall_64** is using the most CPU time.

Additional resources

- **perf-top(1)** man page

17.3. INTERPRETATION OF PERF TOP OUTPUT

The **perf top** monitoring interface displays the data in several columns:

The "Overhead" column

Displays the percent of CPU a given function is using.

The "Shared Object" column

Displays name of the program or library which is using the function.

The "Symbol" column

Displays the function name or symbol. Functions executed in the kernel-space are identified by **[k]** and functions executed in the user-space are identified by **[.]**.

17.4. WHY PERF DISPLAYS SOME FUNCTION NAMES AS RAW FUNCTION ADDRESSES

For kernel functions, **perf** uses the information from the **/proc/kallsyms** file to map the samples to their respective function names or symbols. For functions executed in the user space, however, you might see raw function addresses because the binary is stripped.

The **debuginfo** package of the executable must be installed or, if the executable is a locally developed application, the application must be compiled with debugging information turned on (the **-g** option in GCC) to display the function names or symbols in such a situation.



NOTE

It is not necessary to re-run the **perf record** command after installing the **debuginfo** associated with an executable. Simply re-run the **perf report** command.

Additional Resources

- [Enabling debugging with debugging information](#)

17.5. ENABLING DEBUG AND SOURCE REPOSITORIES

A standard installation of Red Hat Enterprise Linux does not enable the debug and source repositories. These repositories contain information needed to debug the system components and measure their performance.

Procedure

- Enable the source and debug information package channels: The **\$(uname -i)** part is automatically replaced with a matching value for architecture of your system:

Architecture name	Value
64-bit Intel and AMD	x86_64
64-bit ARM	aarch64
IBM POWER	ppc64le
64-bit IBM Z	s390x

17.6. GETTING DEBUGINFO PACKAGES FOR AN APPLICATION OR LIBRARY USING GDB

Debugging information is required to debug code. For code that is installed from a package, the GNU Debugger (GDB) automatically recognizes missing debug information, resolves the package name and provides concrete advice on how to get the package.

Prerequisites

- The application or library you want to debug must be installed on the system.
- GDB and the **debuginfo-install** tool must be installed on the system. For details, see [Setting up to debug applications](#).
- Repositories providing **debuginfo** and **debugsource** packages must be configured and enabled on the system. For details, see [Enabling debug and source repositories](#).

Procedure

1. Start GDB attached to the application or library you want to debug. GDB automatically recognizes missing debugging information and suggests a command to run.

```
$ gdb -q /bin/ls
Reading symbols from /bin/ls...Reading symbols from .gnu_debugdata for /usr/bin/ls...(no
debugging symbols found)...done.
(no debugging symbols found)...done.
Missing separate debuginfos, use: dnf debuginfo-install coreutils-8.30-6.el8.x86_64
(gdb)
```

2. Exit GDB: type **q** and confirm with **Enter**.

```
(gdb) q
```

3. Run the command suggested by GDB to install the required **debuginfo** packages:

```
# dnf debuginfo-install coreutils-8.30-6.el8.x86_64
```

The **dnf** package management tool provides a summary of the changes, asks for confirmation and once you confirm, downloads and installs all the necessary files.

4. In case GDB is not able to suggest the **debuginfo** package, follow the procedure described in [Getting debuginfo packages for an application or library manually](#).

Additional resources

- [How can I download or install debuginfo packages for RHEL systems?](#) – Red Hat Knowledgebase solution

CHAPTER 18. COUNTING EVENTS DURING PROCESS EXECUTION WITH PERF STAT

You can use the **perf stat** command to count hardware and software events during process execution.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).

18.1. THE PURPOSE OF PERF STAT

The **perf stat** command executes a specified command, keeps a running count of hardware and software event occurrences during the commands execution, and generates statistics of these counts. If you do not specify any events, then **perf stat** counts a set of common hardware and software events.

18.2. COUNTING EVENTS WITH PERF STAT

You can use **perf stat** to count hardware and software event occurrences during command execution and generate statistics of these counts. By default, **perf stat** operates in per-thread mode.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).

Procedure

- Count the events.
 - Running the **perf stat** command without root access will only count events occurring in the user space:

```
$ perf stat ls
```

Example 18.1. Output of perf stat ran without root access

```
Desktop Documents Downloads Music Pictures Public Templates Videos
```

```
Performance counter stats for 'ls':
```

```
1.28 msec task-clock:u          # 0.165 CPUs utilized
0      context-switches:u       # 0.000 M/sec
0      cpu-migrations:u        # 0.000 K/sec
104    page-faults:u           # 0.081 M/sec
1,054,302 cycles:u              # 0.823 GHz
1,136,989 instructions:u        # 1.08 insn per cycle
228,531 branches:u             # 178.447 M/sec
11,331 branch-misses:u         # 4.96% of all branches
```

```
0.007754312 seconds time elapsed
```

```
0.000000000 seconds user
```

```
0.007717000 seconds sys
```

■

As you can see in the previous example, when **perf stat** runs without root access the event names are followed by **:u**, indicating that these events were counted only in the user-space.

- To count both user-space and kernel-space events, you must have root access when running **perf stat**:

```
# perf stat ls
```

Example 18.2. Output of perf stat ran with root access

```
Desktop Documents Downloads Music Pictures Public Templates Videos
```

```
Performance counter stats for 'ls':
```

```

      3.09 msec task-clock           # 0.119 CPUs utilized
      18   context-switches        # 0.006 M/sec
       3   cpu-migrations          # 0.969 K/sec
     108   page-faults             # 0.035 M/sec
 6,576,004 cycles                   # 2.125 GHz
 5,694,223 instructions            # 0.87  insn per cycle
 1,092,372 branches               # 352.960 M/sec
   31,515 branch-misses           # 2.89% of all branches
```

```
0.026020043 seconds time elapsed
```

```
0.000000000 seconds user
```

```
0.014061000 seconds sys
```

- By default, **perf stat** operates in per-thread mode. To change to CPU-wide event counting, pass the **-a** option to **perf stat**. To count CPU-wide events, you need root access:

```
# perf stat -a ls
```

Additional resources

- **perf-stat(1)** man page

18.3. INTERPRETATION OF PERF STAT OUTPUT

perf stat executes a specified command and counts event occurrences during the commands execution and displays statistics of these counts in three columns:

1. The number of occurrences counted for a given event
2. The name of the event that was counted
3. When related metrics are available, a ratio or percentage is displayed after the hash sign (**#**) in the right-most column.

For example, when running in default mode, **perf stat** counts both cycles and instructions and, therefore, calculates and displays instructions per cycle in the right-most column. You can see similar behavior with regard to branch-misses as a percent of all branches since both events are

counted by default.

18.4. ATTACHING PERF STAT TO A RUNNING PROCESS

You can attach **perf stat** to a running process. This will instruct **perf stat** to count event occurrences only in the specified processes during the execution of a command.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).

Procedure

- Attach **perf stat** to a running process:

```
$ perf stat -p ID1,ID2 sleep seconds
```

The previous example counts events in the processes with the IDs of ***ID1*** and ***ID2*** for a time period of ***seconds*** seconds as dictated by using the **sleep** command.

Additional resources

- **perf-stat(1)** man page

CHAPTER 19. RECORDING AND ANALYZING PERFORMANCE PROFILES WITH PERF

The **perf** tool allows you to record performance data and analyze it at a later time.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).

19.1. THE PURPOSE OF PERF RECORD

The **perf record** command samples performance data and stores it in a file, **perf.data**, which can be read and visualized with other **perf** commands. **perf.data** is generated in the current directory and can be accessed at a later time, possibly on a different machine.

If you do not specify a command for **perf record** to record during, it will record until you manually stop the process by pressing **Ctrl+C**. You can attach **perf record** to specific processes by passing the **-p** option followed by one or more process IDs. You can run **perf record** without root access, however, doing so will only sample performance data in the user space. In the default mode, **perf record** uses CPU cycles as the sampling event and operates in per-thread mode with inherit mode enabled.

19.2. RECORDING A PERFORMANCE PROFILE WITHOUT ROOT ACCESS

You can use **perf record** without root access to sample and record performance data in the user-space only.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).

Procedure

- Sample and record the performance data:

```
$ perf record command
```

Replace ***command*** with the command you want to sample data during. If you do not specify a command, then **perf record** will sample data until you manually stop it by pressing **Ctrl+C**.

Additional resources

- **perf-record(1)** man page

19.3. RECORDING A PERFORMANCE PROFILE WITH ROOT ACCESS

You can use **perf record** with root access to sample and record performance data in both the user-space and the kernel-space simultaneously.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).

- You have root access.

Procedure

- Sample and record the performance data:

```
# perf record command
```

Replace ***command*** with the command you want to sample data during. If you do not specify a command, then **perf record** will sample data until you manually stop it by pressing **Ctrl+C**.

Additional resources

- **perf-record(1)** man page

19.4. RECORDING A PERFORMANCE PROFILE IN PER-CPU MODE

You can use **perf record** in per-CPU mode to sample and record performance data in both user-space and the kernel-space simultaneously across all threads on a monitored CPU. By default, per-CPU mode monitors all online CPUs.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).

Procedure

- Sample and record the performance data:

```
# perf record -a command
```

Replace ***command*** with the command you want to sample data during. If you do not specify a command, then **perf record** will sample data until you manually stop it by pressing **Ctrl+C**.

Additional resources

- **perf-record(1)** man page

19.5. CAPTURING CALL GRAPH DATA WITH PERF RECORD

You can configure the **perf record** tool so that it records which function is calling other functions in the performance profile. This helps to identify a bottleneck if several processes are calling the same function.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).

Procedure

- Sample and record performance data with the **--call-graph** option:

```
$ perf record --call-graph method command
```

- Replace **command** with the command you want to sample data during. If you do not specify a command, then **perf record** will sample data until you manually stop it by pressing **Ctrl+C**.
- Replace *method* with one of the following unwinding methods:

fp

Uses the frame pointer method. Depending on compiler optimization, such as with binaries built with the GCC option **--fomit-frame-pointer**, this may not be able to unwind the stack.

dwarf

Uses DWARF Call Frame Information to unwind the stack.

lbr

Uses the last branch record hardware on Intel processors.

Additional resources

- **perf-record(1)** man page

19.6. ANALYZING PERF.DATA WITH PERF REPORT

You can use **perf report** to display and analyze a **perf.data** file.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).
- There is a **perf.data** file in the current directory.
- If the **perf.data** file was created with root access, you need to run **perf report** with root access too.

Procedure

- Display the contents of the **perf.data** file for further analysis:

```
# perf report
```

This command displays output similar to the following:

```
Samples: 2K of event 'cycles', Event count (approx.): 235462960
Overhead Command      Shared Object      Symbol
 2.36% kswapd0        [kernel.kallsyms]  [k] page_vma_mapped_walk
 2.13% sssd_kcm       libc-2.28.so       [.] memset_avx2_erms 2.13% perf
[kernel.kallsyms] [k] smp_call_function_single 1.53% gnome-shell libc-2.28.so [.]
strcmp_avx2
 1.17% gnome-shell    libglib-2.0.so.0.5600.4 [.] g_hash_table_lookup
 0.93% Xorg           libc-2.28.so       [.] memmove_avx_unaligned_erms 0.89%
gnome-shell libgobject-2.0.so.0.5600.4 [.] g_object_unref 0.87% kswapd0 [kernel.kallsyms]
[k] page_referenced_one 0.86% gnome-shell libc-2.28.so [.] memmove_avx_unaligned_erms
 0.83% Xorg           [kernel.kallsyms]  [k] alloc_vmap_area
 0.63% gnome-shell    libglib-2.0.so.0.5600.4 [.] g_slice_alloc
 0.53% gnome-shell    libgirepository-1.0.so.1.0.0 [.] g_base_info_unref
```

```

0.53% gnome-shell ld-2.28.so      [.] _dl_find_dso_for_object
0.49% kswapd0      [kernel.kallsyms] [k] vma_interval_tree_iter_next
0.48% gnome-shell libpthread-2.28.so      [.] pthread_getspecific 0.47% gnome-
shell libgirepository-1.0.so.1.0.0 [.] 0x00000000000013b1d 0.45% gnome-shell libglib-
2.0.so.0.5600.4 [.] g_slice_free1 0.45% gnome-shell libgobject-2.0.so.0.5600.4 [.]
g_type_check_instance_is_fundamentally_a 0.44% gnome-shell libc-2.28.so [.] malloc 0.41%
swapper [kernel.kallsyms] [k] apic_timer_interrupt 0.40% gnome-shell ld-2.28.so [.]
_dl_lookup_symbol_x 0.39% kswapd0 [kernel.kallsyms] [k]
raw_callee_save___pv_queued_spin_unlock

```

Additional resources

- **perf-report(1)** man page

19.7. INTERPRETATION OF PERF REPORT OUTPUT

The table displayed by running the **perf report** command sorts the data into several columns:

The 'Overhead' column

Indicates what percentage of overall samples were collected in that particular function.

The 'Command' column

Tells you which process the samples were collected from.

The 'Shared Object' column

Displays the name of the ELF image where the samples come from (the name [kernel.kallsyms] is used when the samples come from the kernel).

The 'Symbol' column

Displays the function name or symbol.

In default mode, the functions are sorted in descending order with those with the highest overhead displayed first.

19.8. GENERATING A PERF.DATA FILE THAT IS READABLE ON A DIFFERENT DEVICE

You can use the **perf** tool to record performance data into a **perf.data** file to be analyzed on a different device.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).
- The kernel **debuginfo** package is installed. For more information, see [Getting debuginfo packages for an application or library using GDB](#).

Procedure

1. Capture performance data you are interested in investigating further:

```
# perf record -a --call-graph fp sleep seconds
```

This example would generate a **perf.data** over the entire system for a period of **seconds** seconds as dictated by the use of the **sleep** command. It would also capture call graph data using the frame pointer method.

2. Generate an archive file containing debug symbols of the recorded data:

```
# perf archive
```

Verification steps

- Verify that the archive file has been generated in your current active directory:

```
# ls perf.data*
```

The output will display every file in your current directory that begins with **perf.data**. The archive file will be named either:

```
perf.data.tar.gz
```

or

```
perf.data.tar.bz2
```

Additional resources

- [Recording and analyzing performance profiles with perf](#)
- [Capturing call graph data with perf record](#)

19.9. ANALYZING A PERF.DATA FILE THAT WAS CREATED ON A DIFFERENT DEVICE

You can use the **perf** tool to analyze a **perf.data** file that was generated on a different device.

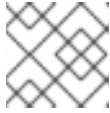
Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).
- A **perf.data** file and associated archive file generated on a different device are present on the current device being used.

Procedure

1. Copy both the **perf.data** file and the archive file into your current active directory.
2. Extract the archive file into **~/debug**:

```
# mkdir -p ~/debug  
# tar xf perf.data.tar.bz2 -C ~/debug
```


**NOTE**

The archive file might also be named ***perf.data.tar.gz***.

- Open the **perf.data** file for further analysis:

```
# perf report
```

19.10. WHY PERF DISPLAYS SOME FUNCTION NAMES AS RAW FUNCTION ADDRESSES

For kernel functions, **perf** uses the information from the **/proc/kallsyms** file to map the samples to their respective function names or symbols. For functions executed in the user space, however, you might see raw function addresses because the binary is stripped.

The **debuginfo** package of the executable must be installed or, if the executable is a locally developed application, the application must be compiled with debugging information turned on (the **-g** option in GCC) to display the function names or symbols in such a situation.

**NOTE**

It is not necessary to re-run the **perf record** command after installing the **debuginfo** associated with an executable. Simply re-run the **perf report** command.

Additional Resources

- [Enabling debugging with debugging information](#)

19.11. ENABLING DEBUG AND SOURCE REPOSITORIES

A standard installation of Red Hat Enterprise Linux does not enable the debug and source repositories. These repositories contain information needed to debug the system components and measure their performance.

Procedure

- Enable the source and debug information package channels: The **\$(uname -i)** part is automatically replaced with a matching value for architecture of your system:

Architecture name	Value
64-bit Intel and AMD	x86_64
64-bit ARM	aarch64
IBM POWER	ppc64le
64-bit IBM Z	s390x

19.12. GETTING DEBUGINFO PACKAGES FOR AN APPLICATION OR LIBRARY USING GDB

Debugging information is required to debug code. For code that is installed from a package, the GNU Debugger (GDB) automatically recognizes missing debug information, resolves the package name and provides concrete advice on how to get the package.

Prerequisites

- The application or library you want to debug must be installed on the system.
- GDB and the **debuginfo-install** tool must be installed on the system. For details, see [Setting up to debug applications](#).
- Repositories providing **debuginfo** and **debugsource** packages must be configured and enabled on the system. For details, see [Enabling debug and source repositories](#).

Procedure

1. Start GDB attached to the application or library you want to debug. GDB automatically recognizes missing debugging information and suggests a command to run.

```
$ gdb -q /bin/l  
Reading symbols from /bin/l...Reading symbols from .gnu_debugdata for /usr/bin/l...(no  
debugging symbols found)...done.  
(no debugging symbols found)...done.  
Missing separate debuginfos, use: dnf debuginfo-install coreutils-8.30-6.el8.x86_64  
(gdb)
```

2. Exit GDB: type **q** and confirm with **Enter**.

```
(gdb) q
```

3. Run the command suggested by GDB to install the required **debuginfo** packages:

```
# dnf debuginfo-install coreutils-8.30-6.el8.x86_64
```

The **dnf** package management tool provides a summary of the changes, asks for confirmation and once you confirm, downloads and installs all the necessary files.

4. In case GDB is not able to suggest the **debuginfo** package, follow the procedure described in [Getting debuginfo packages for an application or library manually](#).

Additional resources

- [How can I download or install debuginfo packages for RHEL systems?](#) – Red Hat Knowledgebase solution

CHAPTER 20. INVESTIGATING BUSY CPUS WITH PERF

When investigating performance issues on a system, you can use the **perf** tool to identify and monitor the busiest CPUs in order to focus your efforts.

20.1. DISPLAYING WHICH CPU EVENTS WERE COUNTED ON WITH PERF STAT

You can use **perf stat** to display which CPU events were counted on by disabling CPU count aggregation. You must count events in system-wide mode by using the **-a** flag in order to use this functionality.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).

Procedure

- Count the events with CPU count aggregation disabled:

```
# perf stat -a -A sleep seconds
```

The previous example displays counts of a default set of common hardware and software events recorded over a time period of ***seconds*** seconds, as dictated by using the **sleep** command, over each individual CPU in ascending order, starting with **CPU0**. As such, it may be useful to specify an event such as cycles:

```
# perf stat -a -A -e cycles sleep seconds
```

20.2. DISPLAYING WHICH CPU SAMPLES WERE TAKEN ON WITH PERF REPORT

The **perf record** command samples performance data and stores this data in a **perf.data** file which can be read with the **perf report** command. The **perf record** command always records which CPU samples were taken on. You can configure **perf report** to display this information.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).
- There is a **perf.data** file created with **perf record** in the current directory. If the **perf.data** file was created with root access, you need to run **perf report** with root access too.

Procedure

- Display the contents of the **perf.data** file for further analysis while sorting by CPU:

```
# perf report --sort cpu
```

- You can sort by CPU and command to display more detailed information about where CPU time is being spent:

```
# perf report --sort cpu,comm
```

This example will list commands from all monitored CPUs by total overhead in descending order of overhead usage and identify the CPU the command was executed on.

Additional resources

- [Recording and analyzing performance profiles with perf](#)

20.3. DISPLAYING SPECIFIC CPUS DURING PROFILING WITH PERF TOP

You can configure **perf top** to display specific CPUs and their relative usage while profiling your system in real time.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).

Procedure

- Start the **perf top** interface while sorting by CPU:

```
# perf top --sort cpu
```

This example will list CPUs and their respective overhead in descending order of overhead usage in real time.

- You can sort by CPU and command for more detailed information of where CPU time is being spent:

```
# perf top --sort cpu,comm
```

This example will list commands by total overhead in descending order of overhead usage and identify the CPU the command was executed on in real time.

20.4. MONITORING SPECIFIC CPUS WITH PERF RECORD AND PERF REPORT

You can configure **perf record** to only sample specific CPUs of interest and analyze the generated **perf.data** file with **perf report** for further analysis.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).

Procedure

1. Sample and record the performance data in the specific CPU's, generating a **perf.data** file:
 - Using a comma separated list of CPUs:

```
# perf record -C 0,1 sleep seconds
```

The previous example samples and records data in CPUs 0 and 1 for a period of **seconds** seconds as dictated by the use of the **sleep** command.

- Using a range of CPUs:

```
# perf record -C 0-2 sleep seconds
```

The previous example samples and records data in all CPUs from CPU 0 to 2 for a period of **seconds** seconds as dictated by the use of the **sleep** command.

2. Display the contents of the **perf.data** file for further analysis:

```
# perf report
```

This example will display the contents of **perf.data**. If you are monitoring several CPUs and want to know which CPU data was sampled on, see [Displaying which CPU samples were taken on with perf report](#).

CHAPTER 21. MONITORING APPLICATION PERFORMANCE WITH PERF

You can use the **perf** tool to monitor and analyze application performance.

21.1. ATTACHING PERF RECORD TO A RUNNING PROCESS

You can attach **perf record** to a running process. This will instruct **perf record** to only sample and record performance data in the specified processes.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).

Procedure

- Attach **perf record** to a running process:

```
$ perf record -p ID1,ID2 sleep seconds
```

The previous example samples and records performance data of the processes with the process ID's **ID1** and **ID2** for a time period of **seconds** seconds as dictated by using the **sleep** command. You can also configure **perf** to record events in specific threads:

```
$ perf record -t ID1,ID2 sleep seconds
```



NOTE

When using the **-t** flag and stipulating thread ID's, **perf** disables inheritance by default. You can enable inheritance by adding the **--inherit** option.

21.2. CAPTURING CALL GRAPH DATA WITH PERF RECORD

You can configure the **perf record** tool so that it records which function is calling other functions in the performance profile. This helps to identify a bottleneck if several processes are calling the same function.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).

Procedure

- Sample and record performance data with the **--call-graph** option:

```
$ perf record --call-graph method command
```

- Replace **command** with the command you want to sample data during. If you do not specify a command, then **perf record** will sample data until you manually stop it by pressing **Ctrl+C**.
- Replace **method** with one of the following unwinding methods:

• **libunwind**

ip

Uses the frame pointer method. Depending on compiler optimization, such as with binaries built with the GCC option **--fomit-frame-pointer**, this may not be able to unwind the stack.

dwarf

Uses DWARF Call Frame Information to unwind the stack.

lbr

Uses the last branch record hardware on Intel processors.

Additional resources

- **perf-record(1)** man page

21.3. ANALYZING PERF.DATA WITH PERF REPORT

You can use **perf report** to display and analyze a **perf.data** file.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).
- There is a **perf.data** file in the current directory.
- If the **perf.data** file was created with root access, you need to run **perf report** with root access too.

Procedure

- Display the contents of the **perf.data** file for further analysis:

```
# perf report
```

This command displays output similar to the following:

```
Samples: 2K of event 'cycles', Event count (approx.): 235462960
Overhead Command      Shared Object      Symbol
 2.36% kswapd0      [kernel.kallsyms]  [k] page_vma_mapped_walk
 2.13% sssd_kcm      libc-2.28.so        [.] memset_avx2_erms 2.13% perf
[kernel.kallsyms] [k] smp_call_function_single 1.53% gnome-shell libc-2.28.so [.]
strcmp_avx2
 1.17% gnome-shell  libglib-2.0.so.0.5600.4 [.] g_hash_table_lookup
 0.93% Xorg          libc-2.28.so        [.] memmove_avx_unaligned_erms 0.89%
gnome-shell libgobject-2.0.so.0.5600.4 [.] g_object_unref 0.87% kswapd0 [kernel.kallsyms]
[k] page_referenced_one 0.86% gnome-shell libc-2.28.so [.] memmove_avx_unaligned_erms
 0.83% Xorg          [kernel.kallsyms]  [k] alloc_vmap_area
 0.63% gnome-shell  libglib-2.0.so.0.5600.4 [.] g_slice_alloc
 0.53% gnome-shell  libgirepository-1.0.so.1.0.0 [.] g_base_info_unref
 0.53% gnome-shell  ld-2.28.so          [.] dl_find_dso_for_object
 0.49% kswapd0      [kernel.kallsyms]  [k] vma_interval_tree_iter_next
 0.48% gnome-shell  libpthread-2.28.so  [.] pthread_getspecific 0.47% gnome-
shell libgirepository-1.0.so.1.0.0 [.] 0x0000000000013b1d 0.45% gnome-shell libglib-
2.0.so.0.5600.4 [.] g_slice_free1 0.45% gnome-shell libgobject-2.0.so.0.5600.4 [.]
g_type_check_instance_is_fundamentally_a 0.44% gnome-shell libc-2.28.so [.] malloc 0.41%
```

```
swapper [kernel.kallsyms] [k] apic_timer_interrupt 0.40% gnome-shell ld-2.28.so [.]  
_dl_lookup_symbol_x 0.39% kswapd0 [kernel.kallsyms] [k]  
raw_callee_save___pv_queued_spin_unlock
```

Additional resources

- **perf-report(1)** man page

CHAPTER 22. CREATING UPROBES WITH PERF

22.1. CREATING UPROBES AT THE FUNCTION LEVEL WITH PERF

You can use the **perf** tool to create dynamic tracepoints at arbitrary points in a process or application. These tracepoints can then be used in conjunction with other **perf** tools such as **perf stat** and **perf record** to better understand the process or applications behavior.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).

Procedure

1. Create the uprobe in the process or application you are interested in monitoring at a location of interest within the process or application:

```
# perf probe -x /path/to/executable -a function
Added new event:
  probe_executable:function (on function in /path/to/executable)
```

You can now use it in all perf tools, such as:

```
perf record -e probe_executable:function -aR sleep 1
```

Additional resources

- **perf-probe** man page
- [Recording and analyzing performance profiles with perf](#)
- [Counting events during process execution with perf stat](#)

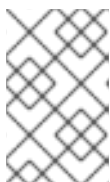
22.2. CREATING UPROBES ON LINES WITHIN A FUNCTION WITH PERF

These tracepoints can then be used in conjunction with other **perf** tools such as **perf stat** and **perf record** to better understand the process or applications behavior.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).
- You have gotten the debugging symbols for your executable:

```
# objdump -t ./your_executable | head
```



NOTE

To do this, the **debuginfo** package of the executable must be installed or, if the executable is a locally developed application, the application must be compiled with debugging information, the **-g** option in GCC.

Procedure

1. View the function lines where you can place a uprobe:

```
$ perf probe -x ./your_executable -L main
```

Output of this command looks similar to:

```
<main@/home/user/my_executable:0>
  0 int main(int argc, const char **argv)
  1 {
      int err;
      const char *cmd;
      char sbuf[STRERR_BUFSIZE];

      /* libsubcmd init */
  7   exec_cmd_init("perf", PREFIX, PERF_EXEC_PATH,
EXEC_PATH_ENVIRONMENT);
  8   pager_init(PERF_PAGER_ENVIRONMENT);
```

2. Create the uprobe for the desired function line:

```
# perf probe -x ./my_executable main:8
Added new event:
  probe_my_executable:main_L8 (on main:8 in /home/user/my_executable)
```

You can now use it in all perf tools, such as:

```
perf record -e probe_my_executable:main_L8 -aR sleep 1
```

22.3. PERF SCRIPT OUTPUT OF DATA RECORDED OVER UPROBES

A common method to analyze data collected using uprobes is using the **perf script** command to read a **perf.data** file and display a detailed trace of the recorded workload.

In the perf script example output:

- A uprobe is added to the function **isprime()** in a program called **my_prog**
- **a** is a function argument added to the uprobe. Alternatively, **a** could be an arbitrary variable visible in the code scope of where you add your uprobe:

```
# perf script
my_prog 1367 [007] 10802159.906593: probe_my_prog:isprime: (400551) a=2
my_prog 1367 [007] 10802159.906623: probe_my_prog:isprime: (400551) a=3
my_prog 1367 [007] 10802159.906625: probe_my_prog:isprime: (400551) a=4
my_prog 1367 [007] 10802159.906627: probe_my_prog:isprime: (400551) a=5
my_prog 1367 [007] 10802159.906629: probe_my_prog:isprime: (400551) a=6
my_prog 1367 [007] 10802159.906631: probe_my_prog:isprime: (400551) a=7
my_prog 1367 [007] 10802159.906633: probe_my_prog:isprime: (400551) a=13
my_prog 1367 [007] 10802159.906635: probe_my_prog:isprime: (400551) a=17
my_prog 1367 [007] 10802159.906637: probe_my_prog:isprime: (400551) a=19
```

CHAPTER 23. PROFILING MEMORY ACCESSES WITH PERF MEM

You can use the **perf mem** command to sample memory accesses on your system.

23.1. THE PURPOSE OF PERF MEM

The **mem** subcommand of the **perf** tool enables the sampling of memory accesses (loads and stores). The **perf mem** command provides information about memory latency, types of memory accesses, functions causing cache hits and misses, and, by recording the data symbol, the memory locations where these hits and misses occur.

23.2. SAMPLING MEMORY ACCESS WITH PERF MEM

This procedure describes how to use the **perf mem** command to sample memory accesses on your system. The command takes the same options as **perf record** and **perf report** as well as some options exclusive to the **mem** subcommand. The recorded data is stored in a **perf.data** file in the current directory for later analysis.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).

Procedure

1. Sample the memory accesses:

```
# perf mem record -a sleep seconds
```

This example samples memory accesses across all CPUs for a period of *seconds* seconds as dictated by the **sleep** command. You can replace the **sleep** command for any command during which you want to sample memory access data. By default, **perf mem** samples both memory loads and stores. You can select only one memory operation by using the **-t** option and specifying either "load" or "store" between **perf mem** and **record**. For loads, information over the memory hierarchy level, TLB memory accesses, bus snoops, and memory locks is captured.

2. Open the **perf.data** file for analysis:

```
# perf mem report
```

If you have used the example commands, the output is:

```
Available samples
35k cpu/mem-loads,ldlat=30/P
54k cpu/mem-stores/P
```

The **cpu/mem-loads,ldlat=30/P** line denotes data collected over memory loads and the **cpu/mem-stores/P** line denotes data collected over memory stores. Highlight the category of interest and press **Enter** to view the data:

```
Samples: 35K of event 'cpu/mem-loads,ldlat=30/P', Event count (approx.): 4067062
Overhead    Samples Local Weight Memory access      Symbol
```

Shared Object		Data Symbol	Data Object
Snoop	TLB access	Locked	
0.07%	29 98	L1 or L1 hit	[.] 0x000000000000a255
libspeexdsp.so.1.5.0		[.] 0x00007f697a3cd0f0	anon
None	L1 or L2 hit	No	
0.06%	26 97	L1 or L1 hit	[.] 0x000000000000a255
libspeexdsp.so.1.5.0		[.] 0x00007f697a3cd0f0	anon
None	L1 or L2 hit	No	
0.06%	25 96	L1 or L1 hit	[.] 0x000000000000a255
libspeexdsp.so.1.5.0		[.] 0x00007f697a3cd0f0	anon
None	L1 or L2 hit	No	
0.06%	1 2325	Uncached or N/A hit	[k] pci_azx_readl
[kernel.kallsyms]		[k] 0xffffb092c06e9084	[kernel.kallsyms]
None	L1 or L2 hit	No	
0.06%	1 2247	Uncached or N/A hit	[k] pci_azx_readl
[kernel.kallsyms]		[k] 0xffffb092c06e8164	[kernel.kallsyms]
None	L1 or L2 hit	No	
0.05%	1 2166	L1 or L1 hit	[.] 0x00000000038140d6
libxul.so		[.] 0x00007ffd7b84b4a8	[stack]
None	L1 or L2 hit	No	
0.05%	1 2117	Uncached or N/A hit	[k] check_for_unclaimed_mmio
[kernel.kallsyms]		[k] 0xffffb092c1842300	[kernel.kallsyms]
None	L1 or L2 hit	No	
0.05%	22 95	L1 or L1 hit	[.] 0x000000000000a255
libspeexdsp.so.1.5.0		[.] 0x00007f697a3cd0f0	anon
None	L1 or L2 hit	No	
0.05%	1 1898	L1 or L1 hit	[.] 0x0000000002a30e07
libxul.so		[.] 0x00007f610422e0e0	anon
None	L1 or L2 hit	No	
0.05%	1 1878	Uncached or N/A hit	[k] pci_azx_readl
[kernel.kallsyms]		[k] 0xffffb092c06e8164	[kernel.kallsyms]
None	L2 miss	No	
0.04%	18 94	L1 or L1 hit	[.] 0x000000000000a255
libspeexdsp.so.1.5.0		[.] 0x00007f697a3cd0f0	anon
None	L1 or L2 hit	No	
0.04%	1 1593	Local RAM or RAM hit	[.] 0x00000000026f907d
libxul.so		[.] 0x00007f3336d50a80	anon
Hit	L2 miss	No	
0.03%	1 1399	L1 or L1 hit	[.] 0x00000000037cb5f1
libxul.so		[.] 0x00007f6e81ef5d78	libxul.so
None	L1 or L2 hit	No	
0.03%	1 1229	LFB or LFB hit	[.] 0x0000000002962aad
libxul.so		[.] 0x00007fb6f1be2b28	anon
None	L2 miss	No	
0.03%	1 1202	LFB or LFB hit	[.] __pthread_mutex_lock
libpthread-2.29.so		[.] 0x00007fb75583ef20	anon
None	L1 or L2 hit	No	
0.03%	1 1193	Uncached or N/A hit	[k] pci_azx_readl
[kernel.kallsyms]		[k] 0xffffb092c06e9164	[kernel.kallsyms]
None	L2 miss	No	
0.03%	1 1191	L1 or L1 hit	[k] azx_get_delay_from_lpi
[kernel.kallsyms]		[k] 0xffffb092ca7efcf0	[kernel.kallsyms]
None	L1 or L2 hit	No	

Alternatively, you can sort your results to investigate different aspects of interest when displaying the data. For example, to sort data over memory loads by type of memory accesses occurring during the sampling period in descending order of overhead they account for:

```
# perf mem -t load report --sort=mem
```

For example, the output can be:

```
Samples: 35K of event 'cpu/mem-loads,ldlat=30/P', Event count (approx.): 40670
Overhead   Samples  Memory access
31.53%     9725   LFB or LFB hit
29.70%    12201  L1 or L1 hit
23.03%     9725   L3 or L3 hit
12.91%     2316   Local RAM or RAM hit
 2.37%      743   L2 or L2 hit
 0.34%        9   Uncached or N/A hit
 0.10%       69   I/O or N/A hit
 0.02%      825   L3 miss
```

Additional resources

- **perf-mem(1)** man page.

23.3. INTERPRETATION OF PERF MEM REPORT OUTPUT

The table displayed by running the **perf mem report** command without any modifiers sorts the data into several columns:

The 'Overhead' column

Indicates percentage of overall samples collected in that particular function.

The 'Samples' column

Displays the number of samples accounted for by that row.

The 'Local Weight' column

Displays the access latency in processor core cycles.

The 'Memory Access' column

Displays the type of memory access that occurred.

The 'Symbol' column

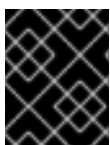
Displays the function name or symbol.

The 'Shared Object' column

Displays the name of the ELF image where the samples come from (the name [kernel.kallsyms] is used when the samples come from the kernel).

The 'Data Symbol' column

Displays the address of the memory location that row was targeting.



IMPORTANT

Oftentimes, due to dynamic allocation of memory or stack memory being accessed, the 'Data Symbol' column will display a raw address.

The "Snoop" column

Displays bus transactions.

The 'TLB Access' column

Displays TLB memory accesses.

The 'Locked' column

Indicates if a function was or was not memory locked.

In default mode, the functions are sorted in descending order with those with the highest overhead displayed first.

CHAPTER 24. DETECTING FALSE SHARING

False sharing occurs when a processor core on a Symmetric Multi Processing (SMP) system modifies data items on the same cache line that is in use by other processors to access other data items that are not being shared between the processors.

This initial modification requires that the other processors using the cache line invalidate their copy and request an updated one despite the processors not needing, or even necessarily having access to, an updated version of the modified data item.

You can use the **perf c2c** command to detect false sharing.

24.1. THE PURPOSE OF PERF C2C

The **c2c** subcommand of the **perf** tool enables Shared Data Cache-to-Cache (C2C) analysis. You can use the **perf c2c** command to inspect cache-line contention to detect both true and false sharing.

Cache-line contention occurs when a processor core on a Symmetric Multi Processing (SMP) system modifies data items on the same cache line that is in use by other processors. All other processors using this cache-line must then invalidate their copy and request an updated one. This can lead to degraded performance.

The **perf c2c** command provides the following information:

- Cache lines where contention has been detected
- Processes reading and writing the data
- Instructions causing the contention
- The Non-Uniform Memory Access (NUMA) nodes involved in the contention

24.2. DETECTING CACHE-LINE CONTENTION WITH PERF C2C

Use the **perf c2c** command to detect cache-line contention in a system.

The **perf c2c** command supports the same options as **perf record** as well as some options exclusive to the **c2c** subcommand. The recorded data is stored in a **perf.data** file in the current directory for later analysis.

Prerequisites

- The **perf** user space tool is installed. For more information, see [installing perf](#).

Procedure

- Use **perf c2c** to detect cache-line contention:

```
# perf c2c record -a sleep seconds
```

This example samples and records cache-line contention data across all CPU's for a period of **seconds** as dictated by the **sleep** command. You can replace the **sleep** command with any command you want to collect cache-line contention data over.

Additional resources

- **perf-c2c(1)** man page

24.3. VISUALIZING A PERF.DATA FILE RECORDED WITH PERF C2C RECORD

This procedure describes how to visualize the **perf.data** file, which is recorded using the **perf c2c** command.

Prerequisites

- The **perf** user space tool is installed. For more information, see [Installing perf](#).
- A **perf.data** file recorded using the **perf c2c** command is available in the current directory. For more information, see [Detecting cache-line contention with perf c2c](#).

Procedure

1. Open the **perf.data** file for further analysis:

```
# perf c2c report --stdio
```

This command visualizes the **perf.data** file into several graphs within the terminal:

```
=====
                Trace Event Information
=====
Total records           :   329219
Locked Load/Store Operations :   14654
Load Operations         :   69679
Loads - uncacheable     :         0
Loads - IO              :         0
Loads - Miss            :    3972
Loads - no mapping      :         0
Load Fill Buffer Hit     :   11958
Load L1D hit            :   17235
Load L2D hit            :        21
Load LLC hit            :   14219
Load Local HITM         :    3402
Load Remote HITM        :   12757
Load Remote HIT         :    5295
Load Local DRAM         :    976
Load Remote DRAM        :    3246
Load MESI State Exclusive :   4222
Load MESI State Shared  :         0
Load LLC Misses         :   22274
LLC Misses to Local DRAM :    4.4%
LLC Misses to Remote DRAM :   14.6%
LLC Misses to Remote cache (HIT) :   23.8%
LLC Misses to Remote cache (HITM) :   57.3%
Store Operations        :  259539
Store - uncacheable     :         0
Store - no mapping      :        11
```



```

Store L1D Hit           : 256696
Store L1D Miss          : 2832
No Page Map Rejects     : 2376
Unable to parse data source : 1

```

```

=====
Global Shared Cache Line Event Information
=====

```

```

Total Shared Cache Lines : 55
Load HITs on shared lines : 55454
Fill Buffer Hits on shared lines : 10635
L1D hits on shared lines : 16415
L2D hits on shared lines : 0
LLC hits on shared lines : 8501
Locked Access on shared lines : 14351
Store HITs on shared lines : 109953
Store L1D hits on shared lines : 109449
Total Merged records : 126112

```

```

=====
c2c details
=====

```

```

Events : cpu/mem-loads,ldlat=30/P
        : cpu/mem-stores/P
Cachelines sort on : Remote HITMs
Cacheline data grouping : offset,pid,iaddr

```

```

=====
Shared Data Cache Line Table
=====

```

```

#
#           Total   Rmt  ----- LLC Load Hitm ----- --- Store Reference ---- --- Load
Drum ----   LLC   Total  ----- Core Load Hit ----- -- LLC Load Hit --
# Index      Cacheline records  Hitm  Total   Lcl  Rmt  Total  L1Hit  L1Miss
Lcl   Rmt  Ld Miss  Loads    FB   L1   L2   Llc   Rmt
# .....
#
# 0          0x602180 149904 77.09% 12103 2269 9834 109504 109036 468
727 2657 13747 40400 5355 16154 0 2875 529
# 1          0x602100 12128 22.20% 3951 1119 2832 0 0 0 65
200 3749 12128 5096 108 0 2056 652
# 2 0xffff883ffb6a7e80 260 0.09% 15 3 12 161 161 0 1
1 15 99 25 50 0 6 1
# 3 0xffffffff81aec000 157 0.07% 9 0 9 1 0 1 0 7
20 156 50 59 0 27 4
# 4 0xffffffff81e3f540 179 0.06% 9 1 8 117 97 20 0 10
25 62 11 1 0 24 7

```

```

=====
Shared Cache Line Distribution Pareto
=====

```

```

#
# ----- HITM ----- -- Store Refs -- Data address ----- cycles --
----- cpu Shared
# Num Rmt Lcl L1 Hit L1 Miss Offset Pid Code address rmt hitm lcl

```

```

hitm    load    cnt      Symbol      Object      Source:Line Node{cpu list}
# .....
#
-----
0    9834    2269  109036    468      0x602180
-----
65.51% 55.88% 75.20% 0.00%      0x0  14604      0x400b4f  27161
26039 26017    9 [...] read_write_func no_false_sharing.exe
false_sharing_example.c:144 0{0-1,4} 1{24-25,120} 2{48,54} 3{169}
0.41% 0.35% 0.00% 0.00%      0x0  14604      0x400b56  18088
12601 26671    9 [...] read_write_func no_false_sharing.exe
false_sharing_example.c:145 0{0-1,4} 1{24-25,120} 2{48,54} 3{169}
0.00% 0.00% 24.80% 100.00%      0x0  14604      0x400b61    0    0
0    9 [...] read_write_func no_false_sharing.exe false_sharing_example.c:145 0{0-1,4}
1{24-25,120} 2{48,54} 3{169}
7.50% 9.92% 0.00% 0.00%      0x20  14604      0x400ba7  2470
1729 1897    2 [...] read_write_func no_false_sharing.exe
false_sharing_example.c:154 1{122} 2{144}
17.61% 20.89% 0.00% 0.00%      0x28  14604      0x400bc1  2294
1575 1649    2 [...] read_write_func no_false_sharing.exe
false_sharing_example.c:158 2{53} 3{170}
8.97% 12.96% 0.00% 0.00%      0x30  14604      0x400bdb  2325
1897 1828    2 [...] read_write_func no_false_sharing.exe
false_sharing_example.c:162 0{96} 3{171}
-----
1    2832    1119    0    0      0x602100
-----
29.13% 36.19% 0.00% 0.00%      0x20  14604      0x400bb3  1964
1230 1788    2 [...] read_write_func no_false_sharing.exe
false_sharing_example.c:155 1{122} 2{144}
43.68% 34.41% 0.00% 0.00%      0x28  14604      0x400bcd  2274
1566 1793    2 [...] read_write_func no_false_sharing.exe
false_sharing_example.c:159 2{53} 3{170}
27.19% 29.40% 0.00% 0.00%      0x30  14604      0x400be7  2045
1247 2011    2 [...] read_write_func no_false_sharing.exe
false_sharing_example.c:163 0{96} 3{171}

```

24.4. INTERPRETATION OF PERF C2C REPORT OUTPUT

The visualization displayed by running the **perf c2c report --stdio** command sorts the data into several tables:

Trace Events Information

This table provides a high level summary of all the load and store samples, which are collected by the **perf c2c record** command.

Global Shared Cache Line Event Information

This table provides statistics over the shared cache lines.

c2c Details

This table provides information about what events were sampled and how the **perf c2c report** data is organized within the visualization.

Shared Data Cache Line Table

This table provides a one line summary for the hottest cache lines where false sharing is detected and is sorted in descending order by the amount of remote **Hitm** detected per cache line by default.

Shared Cache Line Distribution Pareto

This tables provides a variety of information about each cache line experiencing contention:

- The cache lines are numbered in the **NUM** column, starting at **0**.
- The virtual address of each cache line is contained in the **Data address Offset** column and followed subsequently by the offset into the cache line where different accesses occurred.
- The **Pid** column contains the process ID.
- The **Code Address** column contains the instruction pointer code address.
- The columns under the **cycles** label show average load latencies.
- The **cpu cnt** column displays how many different CPUs samples came from (essentially, how many different CPUs were waiting for the data indexed at that given location).
- The **Symbol** column displays the function name or symbol.
- The **Shared Object** column displays the name of the ELF image where the samples come from (the name **[kernel.kallsyms]** is used when the samples come from the kernel).
- The **Source:Line** column displays the source file and line number.
- The **Node{cpu list}** column displays which specific CPUs samples came from for each node.

24.5. DETECTING FALSE SHARING WITH PERF C2C

This procedure describes how to detect false sharing using the **perf c2c** command.

Prerequisites

- The **perf** user space tool is installed. For more information, see [installing perf](#).
- A **perf.data** file recorded using the **perf c2c** command is available in the current directory. For more information, see [Detecting cache-line contention with perf c2c](#).

Procedure

1. Open the **perf.data** file for further analysis:

```
# perf c2c report --stdio
```

This opens the **perf.data** file in the terminal.

2. In the "Trace Event Information" table, locate the row containing the values for **LLC Misses to Remote Cache (HITM)**:

The percentage in the value column of the **LLC Misses to Remote Cache (HITM)** row represents the percentage of LLC misses that were occurring across NUMA nodes in modified cache-lines and is a key indicator false sharing has occurred.

```
=====
```

Trace Event Information

```

=====
Total records           : 329219
Locked Load/Store Operations : 14654
Load Operations         : 69679
Loads - uncacheable    : 0
Loads - IO              : 0
Loads - Miss            : 3972
Loads - no mapping     : 0
Load Fill Buffer Hit     : 11958
Load L1D hit            : 17235
Load L2D hit            : 21
Load LLC hit            : 14219
Load Local HITM         : 3402
Load Remote HITM        : 12757
Load Remote HIT         : 5295
Load Local DRAM         : 976
Load Remote DRAM        : 3246
Load MESI State Exclusive : 4222
Load MESI State Shared   : 0
Load LLC Misses         : 22274
LLC Misses to Local DRAM : 4.4%
LLC Misses to Remote DRAM : 14.6%
LLC Misses to Remote cache (HIT) : 23.8%
LLC Misses to Remote cache (HITM) : 57.3%
Store Operations        : 259539
Store - uncacheable     : 0
Store - no mapping      : 11
Store L1D Hit           : 256696
Store L1D Miss          : 2832
No Page Map Rejects     : 2376
Unable to parse data source : 1

```

3. Inspect the **Rmt** column of the **LLC Load Hitm** field of the **Shared Data Cache Line Table**

```

=====
Shared Data Cache Line Table
=====
#
#           Total   Rmt  ---- LLC Load Hitm ----  ---- Store Reference ----  ---
Load Dram ----   LLC   Total  ---- Core Load Hit ----  -- LLC Load Hit --
# Index      Cacheline records   Hitm   Total   Lcl   Rmt   Total   L1Hit   L1Miss
Lcl   Rmt   Ld Miss   Loads   FB   L1   L2   Llc   Rmt
# .....
#
0          0x602180 149904 77.09% 12103 2269 9834 109504 109036
468 727 2657 13747 40400 5355 16154 0 2875 529
1          0x602100 12128 22.20% 3951 1119 2832 0 0 0 65
200 3749 12128 5096 108 0 2056 652
2 0xffff883ffb6a7e80 260 0.09% 15 3 12 161 161 0 1
1 15 99 25 50 0 6 1
3 0xffffffff81aec000 157 0.07% 9 0 9 1 0 1 0 7
20 156 50 59 0 27 4
4 0xffffffff81e3f540 179 0.06% 9 1 8 117 97 20 0
10 25 62 11 1 0 24 7

```

-

This table is sorted in descending order by the amount of remote **Hitm** detected per cache line. A high number in the **Rmt** column of the **LLC Load Hitm** section indicates false sharing and requires further inspection of the cache line on which it occurred to debug the false sharing activity.

CHAPTER 25. GETTING STARTED WITH FLAMEGRAPHS

As a system administrator, you can use **flamegraphs** to create visualizations of system performance data recorded with the **perf** tool. As a software developer, you can use **flamegraphs** to create visualizations of application performance data recorded with the **perf** tool.

Sampling stack traces is a common technique for profiling CPU performance with the **perf** tool. Unfortunately, the results of profiling stack traces with **perf** can be extremely verbose and labor-intensive to analyze. **flamegraphs** are visualizations created from data recorded with **perf** to make identifying hot code-paths faster and easier.

25.1. INSTALLING FLAMEGRAPHS

To begin using **flamegraphs**, install the required package.

Procedure

- Install the **flamegraphs** package:

```
# dnf install js-d3-flame-graph
```

25.2. CREATING FLAMEGRAPHS OVER THE ENTIRE SYSTEM

This procedure describes how to visualize performance data recorded over an entire system using **flamegraphs**.

Prerequisites

- **flamegraphs** are installed as described in [installing flamegraphs](#).
- The **perf** tool is installed as described in [installing perf](#).

Procedure

- Record the data and create the visualization:

```
# perf script flamegraph -a -F 99 sleep 60
```

This command samples and records performance data over the entire system for 60 seconds, as stipulated by use of the **sleep** command, and then constructs the visualization which will be stored in the current active directory as **flamegraph.html**. The command samples call-graph data by default and takes the same arguments as the **perf** tool, in this particular case:

-a

Stipulates to record data over the entire system.

-F

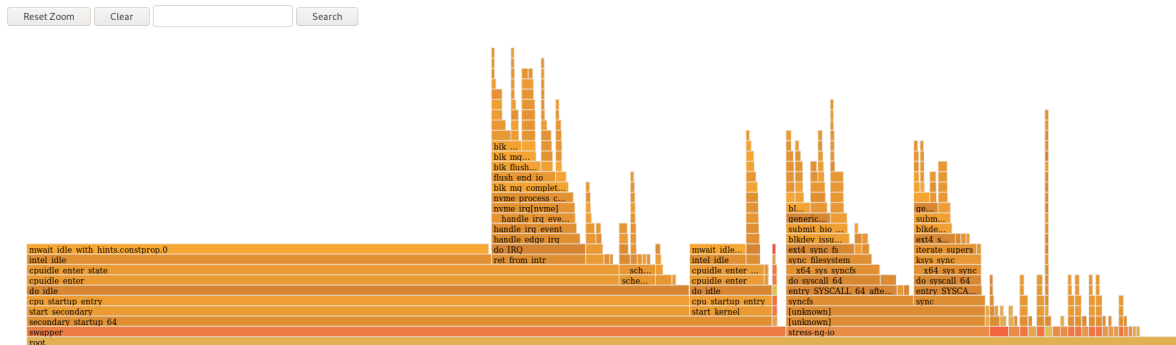
To set the sampling frequency per second.

Verification steps

- For analysis, view the generated visualization:

```
# xdg-open flamegraph.html
```

This command opens the visualization in the default browser:



25.3. CREATING FLAMEGRAPHS OVER SPECIFIC PROCESSES

You can use **flamegraphs** to visualize performance data recorded over specific running processes.

Prerequisites

- **flamegraphs** are installed as described in [installing flamegraphs](#).
- The **perf** tool is installed as described in [installing perf](#).

Procedure

- Record the data and create the visualization:

```
# perf script flamegraph -a -F 99 -p ID1,ID2 sleep 60
```

This command samples and records performance data of the processes with the process ID's **ID1** and **ID2** for 60 seconds, as stipulated by use of the **sleep** command, and then constructs the visualization which will be stored in the current active directory as **flamegraph.html**. The command samples call-graph data by default and takes the same arguments as the **perf** tool, in this particular case:

-a

Stipulates to record data over the entire system.

-F

To set the sampling frequency per second.

-p

To stipulate specific process ID's to sample and record data over.

Verification steps

- For analysis, view the generated visualization:

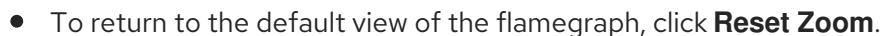
```
# xdg-open flamegraph.html
```

This command opens the visualization in the default browser:



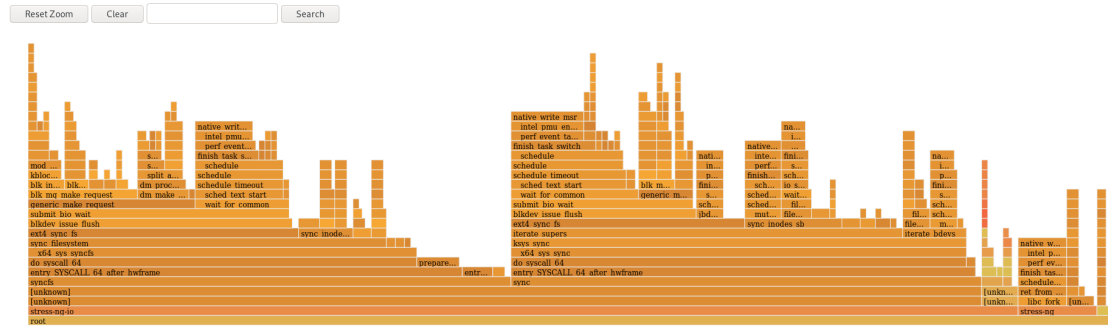
The children of a stack in a given row are displayed based on the number of samples taken of each respective function in descending order along the x-axis; the x-axis does not represent the passing of time. The wider an individual box is, the more frequent it was on-CPU or part of an on-CPU ancestry at the time the data was being sampled.

- To reveal the names of functions which may have not been displayed previously and further investigate the data click on a box within the flamegraph to zoom into the stack at that given location:



IMPORTANT

Boxes representing user-space functions may be labeled as **Unknown** in **flamegraphs** because the binary of the function is stripped. The **debuginfo** package of the executable must be installed or, if the executable is a locally developed application, the application must be compiled with debugging information. Use the **-g** option in GCC, to display the function names or symbols in such a situation.



Additional resources

- [Why perf displays some function names as raw functions addresses](#)
- [Enabling debugging with debugging information](#)

CHAPTER 26. MONITORING PROCESSES FOR PERFORMANCE BOTTLENECKS USING PERF CIRCULAR BUFFERS

You can create circular buffers that take event-specific snapshots of data with the **perf** tool in order to monitor performance bottlenecks in specific processes or parts of applications running on your system. In such cases, **perf** only writes data to a **perf.data** file for later analysis if a specified event is detected.

26.1. CIRCULAR BUFFERS AND EVENT-SPECIFIC SNAPSHOTS WITH PERF

When investigating performance issues in a process or application with **perf**, it may not be affordable or appropriate to record data for hours preceding a specific event of interest occurring. In such cases, you can use **perf record** to create custom circular buffers that take snapshots after specific events.

The **--overwrite** option makes **perf record** store all data in an overwriteable circular buffer. When the buffer gets full, **perf record** automatically overwrites the oldest records which, therefore, never get written to a **perf.data** file.

Using the **--overwrite** and **--switch-output-event** options together configures a circular buffer that records and dumps data continuously until it detects the **--switch-output-event** trigger event. The trigger event signals to **perf record** that something of interest to the user has occurred and to write the data in the circular buffer to a **perf.data** file. This collects specific data you are interested in while simultaneously reducing the overhead of the running **perf** process by not writing data you do not want to a **perf.data** file.

26.2. COLLECTING SPECIFIC DATA TO MONITOR FOR PERFORMANCE BOTTLENECKS USING PERF CIRCULAR BUFFERS

With the **perf** tool, you can create circular buffers that are triggered by events you specify in order to only collect data you are interested in. To create circular buffers that collect event-specific data, use the **--overwrite** and **--switch-output-event** options for **perf**.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).
- You have placed a uprobe in the process or application you are interested in monitoring at a location of interest within the process or application:

```
# perf probe -x /path/to/executable -a function
Added new event:
probe_executable:function (on function in /path/to/executable)
```

You can now use it in all **perf** tools, such as:

```
perf record -e probe_executable:function -aR sleep 1
```

Procedure

- Create the circular buffer with the uprobe as the trigger event:

```
# perf record --overwrite -e cycles --switch-output-event probe_executable:function
./executable
[ perf record: dump data: Woken up 1 times ]
[ perf record: Dump perf.data.2021021012231959 ]
[ perf record: dump data: Woken up 1 times ]
[ perf record: Dump perf.data.2021021012232008 ]
^C[ perf record: dump data: Woken up 1 times ]
[ perf record: Dump perf.data.2021021012232082 ]
[ perf record: Captured and wrote 5.621 MB perf.data.<timestamp> ]
```

This example initiates the executable and collects cpu cycles, specified after the **-e** option, until **perf** detects the uprobe, the trigger event specified after the **--switch-output-event** option. At that point, **perf** takes a snapshot of all the data in the circular buffer and stores it in a unique **perf.data** file identified by timestamp. This example produced a total of 2 snapshots, the last **perf.data** file was forced by pressing **Ctrl+c**.

CHAPTER 27. ADDING AND REMOVING TRACEPOINTS FROM A RUNNING PERF COLLECTOR WITHOUT STOPPING OR RESTARTING PERF

By using the control pipe interface to enable and disable different tracepoints in a running **perf** collector, you can dynamically adjust what data you are collecting without having to stop or restart **perf**. This ensures you do not lose performance data that would have otherwise been recorded during the stopping or restarting process.

27.1. ADDING TRACEPOINTS TO A RUNNING PERF COLLECTOR WITHOUT STOPPING OR RESTARTING PERF

Add tracepoints to a running **perf** collector using the control pipe interface to adjust the data you are recording without having to stop **perf** and losing performance data.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).

Procedure

1. Configure the control pipe interface:

```
# mkfifo control ack perf.pipe
```

2. Run **perf record** with the control file setup and events you are interested in enabling:

```
# perf record --control=fifo:control,ack -D -1 --no-buffering -e 'sched:*' -o - > perf.pipe
```

In this example, declaring '**sched:***' after the **-e** option starts **perf record** with scheduler events.

3. In a second terminal, start the read side of the control pipe:

```
# cat perf.pipe | perf --no-pager script -i -
```

Starting the read side of the control pipe triggers the following message in the first terminal:

```
Events disabled
```

4. In a third terminal, enable a tracepoint using the control file:

```
# echo 'enable sched:sched_process_fork' > control
```

This command triggers **perf** to scan the current event list in the control file for the declared event. If the event is present, the tracepoint is enabled and the following message appears in the first terminal:

```
event sched:sched_process_fork enabled
```

Once the tracepoint is enabled, the second terminal displays the output from **perf** detecting the tracepoint:

```
bash 33349 [034] 149587.674295: sched:sched_process_fork: comm=bash pid=33349
child_comm=bash child_pid=34056
```

27.2. REMOVING TRACEPOINTS FROM A RUNNING PERF COLLECTOR WITHOUT STOPPING OR RESTARTING PERF

Remove tracepoints from a running **perf** collector using the control pipe interface to reduce the scope of data you are collecting without having to stop **perf** and losing performance data.

Prerequisites

- You have the **perf** user space tool installed as described in [Installing perf](#).
- You have added tracepoints to a running **perf** collector via the control pipe interface. For more information, see [Adding tracepoints to a running perf collector without stopping or restarting perf](#).

Procedure

- Remove the tracepoint:

```
# echo 'disable sched:sched_process_fork' > control
```



NOTE

This example assumes you have previously loaded scheduler events into the control file and enabled the tracepoint **sched:sched_process_fork**.

This command triggers **perf** to scan the current event list in the control file for the declared event. If the event is present, the tracepoint is disabled and the following message appears in the terminal used to configure the control pipe:

```
event sched:sched_process_fork disabled
```

CHAPTER 28. PROFILING MEMORY ALLOCATION WITH NUMASTAT

With the **numastat** tool, you can display statistics over memory allocations in a system.

The **numastat** tool displays data for each NUMA node separately. You can use this information to investigate memory performance of your system or the effectiveness of different memory policies on your system.

28.1. DEFAULT NUMASTAT STATISTICS

By default, the **numastat** tool displays statistics over these categories of data for each NUMA node:

numa_hit

The number of pages that were successfully allocated to this node.

numa_miss

The number of pages that were allocated on this node because of low memory on the intended node. Each **numa_miss** event has a corresponding **numa_foreign** event on another node.

numa_foreign

The number of pages initially intended for this node that were allocated to another node instead. Each **numa_foreign** event has a corresponding **numa_miss** event on another node.

interleave_hit

The number of interleave policy pages successfully allocated to this node.

local_node

The number of pages successfully allocated on this node by a process on this node.

other_node

The number of pages allocated on this node by a process on another node.



NOTE

High **numa_hit** values and low **numa_miss** values (relative to each other) indicate optimal performance.

28.2. VIEWING MEMORY ALLOCATION WITH NUMASTAT

You can view the memory allocation of the system by using the **numastat** tool.

Prerequisites

- Install the **numactl** package:

```
# dnf install numactl
```

Procedure

- View the memory allocation of your system:

```
$ numastat
node0    node1
```

numa_hit	76557759	92126519
numa_miss	30772308	30827638
numa_foreign	30827638	30772308
interleave_hit	106507	103832
local_node	76502227	92086995
other_node	30827840	30867162

Additional resources

- **numastat(8)** man page

CHAPTER 29. CONFIGURING AN OPERATING SYSTEM TO OPTIMIZE CPU UTILIZATION

You can configure the operating system to optimize CPU utilization across their workloads.

29.1. TOOLS FOR MONITORING AND DIAGNOSING PROCESSOR ISSUES

The following are the tools available in Red Hat Enterprise Linux 9 to monitor and diagnose processor-related performance issues:

- **turbostat** tool prints counter results at specified intervals to help administrators identify unexpected behavior in servers, such as excessive power usage, failure to enter deep sleep states, or system management interrupts (SMIs) being created unnecessarily.
- **numactl** utility provides a number of options to manage processor and memory affinity. The **numactl** package includes the **libnuma** library which offers a simple programming interface to the NUMA policy supported by the kernel, and can be used for more fine-grained tuning than the **numactl** application.
- **numastat** tool displays per-NUMA node memory statistics for the operating system and its processes, and shows administrators whether the process memory is spread throughout a system or is centralized on specific nodes. This tool is provided by the **numactl** package.
- **numad** is an automatic NUMA affinity management daemon. It monitors NUMA topology and resource usage within a system in order to dynamically improve NUMA resource allocation and management.
- **/proc/interrupts** file displays the interrupt request (IRQ) number, the number of similar interrupt requests handled by each processor in the system, the type of interrupt sent, and a comma-separated list of devices that respond to the listed interrupt request.
- **pqos** utility is available in the **intel-cmt-cat** package. It monitors CPU cache and memory bandwidth on recent Intel processors. It monitors:
 - The instructions per cycle (IPC).
 - The count of last level cache MISSES.
 - The size in kilobytes that the program executing in a given CPU occupies in the LLC.
 - The bandwidth to local memory (MBL).
 - The bandwidth to remote memory (MBR).
- **x86_energy_perf_policy** tool allows administrators to define the relative importance of performance and energy efficiency. This information can then be used to influence processors that support this feature when they select options that trade off between performance and energy efficiency.
- **taskset** tool is provided by the **util-linux** package. It allows administrators to retrieve and set the processor affinity of a running process, or launch a process with a specified processor affinity.

Additional resources

- **turbostat(8)**, **numactl(8)**, **numastat(8)**, **numa(7)**, **numad(8)**, **pqos(8)**, **x86_energy_perf_policy(8)**, and **taskset(1)** man pages

29.2. TYPES OF SYSTEM TOPOLOGY

In modern computing, the idea of a CPU is a misleading one, as most modern systems have multiple processors. The topology of the system is the way these processors are connected to each other and to other system resources. This can affect system and application performance, and the tuning considerations for a system.

The following are the two primary types of topology used in modern computing:

Symmetric Multi-Processor (SMP) topology

SMP topology allows all processors to access memory in the same amount of time. However, because shared and equal memory access inherently forces serialized memory accesses from all the CPUs, SMP system scaling constraints are now generally viewed as unacceptable. For this reason, practically all modern server systems are NUMA machines.

Non-Uniform Memory Access (NUMA) topology

NUMA topology was developed more recently than SMP topology. In a NUMA system, multiple processors are physically grouped on a socket. Each socket has a dedicated area of memory and processors that have local access to that memory, these are referred to collectively as a node. Processors on the same node have high speed access to that node's memory bank, and slower access to memory banks not on their node.

Therefore, there is a performance penalty when accessing non-local memory. Thus, performance sensitive applications on a system with NUMA topology should access memory that is on the same node as the processor executing the application, and should avoid accessing remote memory wherever possible.

Multi-threaded applications that are sensitive to performance may benefit from being configured to execute on a specific NUMA node rather than a specific processor. Whether this is suitable depends on your system and the requirements of your application. If multiple application threads access the same cached data, then configuring those threads to execute on the same processor may be suitable. However, if multiple threads that access and cache different data execute on the same processor, each thread may evict cached data accessed by a previous thread. This means that each thread 'misses' the cache and wastes execution time fetching data from memory and replacing it in the cache. Use the **perf** tool to check for an excessive number of cache misses.

29.2.1. Displaying system topologies

There are a number of commands that help understand the topology of a system. This procedure describes how to determine the system topology.

Procedure

- To display an overview of your system topology:

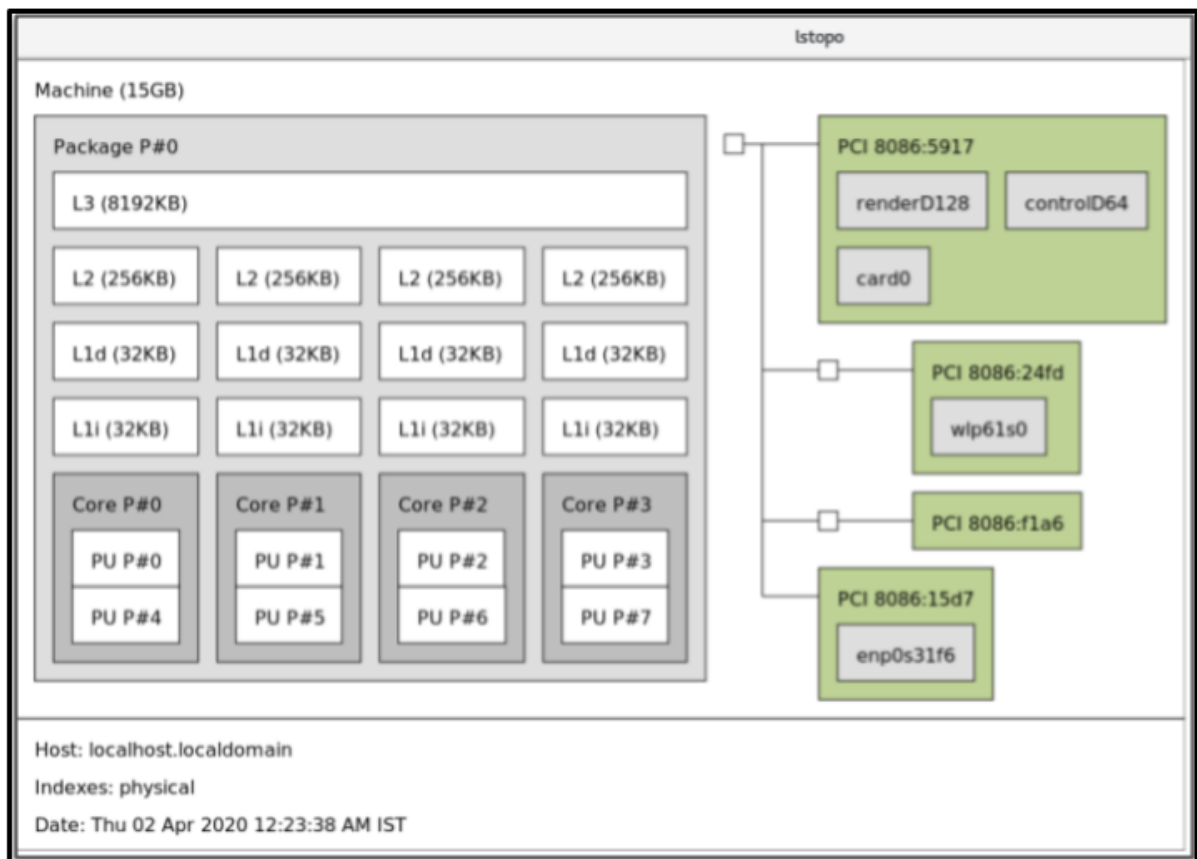
```
$ numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 4 8 12 16 20 24 28 32 36
node 0 size: 65415 MB
node 0 free: 43971 MB
[...]
```

- To gather the information about the CPU architecture, such as the number of CPUs, threads, cores, sockets, and NUMA nodes:

```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                40
On-line CPU(s) list:   0-39
Thread(s) per core:    1
Core(s) per socket:    10
Socket(s):             4
NUMA node(s):          4
Vendor ID:             GenuineIntel
CPU family:            6
Model:                47
Model name:            Intel(R) Xeon(R) CPU E7- 4870  @ 2.40GHz
Stepping:              2
CPU MHz:               2394.204
BogoMIPS:              4787.85
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              30720K
NUMA node0 CPU(s):    0,4,8,12,16,20,24,28,32,36
NUMA node1 CPU(s):    2,6,10,14,18,22,26,30,34,38
NUMA node2 CPU(s):    1,5,9,13,17,21,25,29,33,37
NUMA node3 CPU(s):    3,7,11,15,19,23,27,31,35,39
```

- To view a graphical representation of your system:

```
# dnf install hwloc-gui
# lstopo
```

Figure 29.1. The **lstopo** output

- To view the detailed textual output:

```
# dnf install hwloc
# lstopo-no-graphics
Machine (15GB)
Package L#0 + L3 L#0 (8192KB)
  L2 L#0 (256KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0
    PU L#0 (P#0)
    PU L#1 (P#4)
  HostBridge L#0
  PCI 8086:5917
    GPU L#0 "renderD128"
    GPU L#1 "controlD64"
    GPU L#2 "card0"
  PCIBridge
    PCI 8086:24fd
      Net L#3 "wlp61s0"
  PCIBridge
    PCI 8086:f1a6
  PCI 8086:15d7
    Net L#4 "enp0s31f6"
```

Additional resources

- **numactl(8)**, **lscpu(1)**, and **lstopo(1)** man pages

29.3. CONFIGURING KERNEL TICK TIME

By default, Red Hat Enterprise Linux 9 uses a tickless kernel, which does not interrupt idle CPUs in order to reduce power usage and allow new processors to take advantage of deep sleep states.

Red Hat Enterprise Linux 9 also offers a dynamic tickless option, which is useful for latency-sensitive workloads, such as high performance computing or realtime computing. By default, the dynamic tickless option is disabled. Red Hat recommends using the **cpu-partitioning TuneD** profile to enable the dynamic tickless option for cores specified as **isolated_cores**.

This procedure describes how to manually persistently enable dynamic tickless behavior.

Procedure

1. To enable dynamic tickless behavior in certain cores, specify those cores on the kernel command line with the **nohz_full** parameter. On a 16 core system, enable the **nohz_full=1-15** kernel option:

```
# grubby --update-kernel=ALL --args="nohz_full=1-15"
```

This enables dynamic tickless behavior on cores **1** through **15**, moving all timekeeping to the only unspecified core (core **0**).

2. When the system boots, manually move the **rcu** threads to the non-latency-sensitive core, in this case core **0**:

```
# for i in `pgrep rcu[^c]` ; do taskset -pc 0 $i ; done
```

3. Optional: Use the **isolcpus** parameter on the kernel command line to isolate certain cores from user-space tasks.
4. Optional: Set the CPU affinity for the kernel's **write-back bdi-flush** threads to the housekeeping core:

```
echo 1 > /sys/bus/workqueue/devices/writeback/cpumask
```

Verification steps

- Once the system is rebooted, verify if **dynticks** are enabled:

```
# journalctl -xe | grep dynticks
Mar 15 18:34:54 rhel-server kernel: NO_HZ: Full dynticks CPUs: 1-15.
```

- Verify that the dynamic tickless configuration is working correctly:

```
# perf stat -C 1 -e irq_vectors:local_timer_entry taskset -c 1 sleep 3
```

This command measures ticks on CPU 1 while telling CPU 1 to sleep for 3 seconds.

- The default kernel timer configuration shows around 3100 ticks on a regular CPU:

```
# perf stat -C 0 -e irq_vectors:local_timer_entry taskset -c 0 sleep 3
```

```
Performance counter stats for 'CPU(s) 0':
```

```
3,107    irq_vectors:local_timer_entry
```

```
3.001342790 seconds time elapsed
```

- With the dynamic tickless kernel configured, you should see around 4 ticks instead:

```
# perf stat -C 1 -e irq_vectors:local_timer_entry taskset -c 1 sleep 3
```

```
Performance counter stats for 'CPU(s) 1':
```

```
4    irq_vectors:local_timer_entry
```

```
3.001544078 seconds time elapsed
```

Additional resources

- **perf(1)** and **cpuset(7)** man pages
- [All about nohz_full kernel parameter Red Hat Knowledgebase article](#)
- [How to verify the list of "isolated" and "nohz_full" CPU information from sysfs? Red Hat Knowledgebase article](#)

29.4. OVERVIEW OF AN INTERRUPT REQUEST

An interrupt request or IRQ is a signal for immediate attention sent from a piece of hardware to a processor. Each device in a system is assigned one or more IRQ numbers which allow it to send unique interrupts. When interrupts are enabled, a processor that receives an interrupt request immediately pauses execution of the current application thread in order to address the interrupt request.

Because interrupt halts normal operation, high interrupt rates can severely degrade system performance. It is possible to reduce the amount of time taken by interrupts by configuring interrupt affinity or by sending a number of lower priority interrupts in a batch (coalescing a number of interrupts).

Interrupt requests have an associated affinity property, **smp_affinity**, which defines the processors that handle the interrupt request. To improve application performance, assign interrupt affinity and process affinity to the same processor, or processors on the same core. This allows the specified interrupt and application threads to share cache lines.

On systems that support interrupt steering, modifying the **smp_affinity** property of an interrupt request sets up the hardware so that the decision to service an interrupt with a particular processor is made at the hardware level with no intervention from the kernel.

29.4.1. Balancing interrupts manually

If your BIOS exports its NUMA topology, the **irqbalance** service can automatically serve interrupt requests on the node that is local to the hardware requesting service.

Procedure

1. Check which devices correspond to the interrupt requests that you want to configure.
2. Find the hardware specification for your platform. Check if the chipset on your system supports distributing interrupts.

- a. If it does, you can configure interrupt delivery as described in the following steps. Additionally, check which algorithm your chipset uses to balance interrupts. Some BIOSes have options to configure interrupt delivery.
 - b. If it does not, your chipset always routes all interrupts to a single, static CPU. You cannot configure which CPU is used.
3. Check which Advanced Programmable Interrupt Controller (APIC) mode is in use on your system:

```
$ journalctl --dmesg | grep APIC
```

Here,

- If your system uses a mode other than **flat**, you can see a line similar to **Setting APIC routing to physical flat**.
- If you can see no such message, your system uses **flat** mode. If your system uses **x2apic** mode, you can disable it by adding the **nox2apic** option to the kernel command line in the **bootloader** configuration.

Only non-physical flat mode (**flat**) supports distributing interrupts to multiple CPUs. This mode is available only for systems that have up to **8** CPUs.

4. Calculate the **smp_affinity mask**. For more information about how to calculate the **smp_affinity mask**, see [Setting the smp_affinity mask](#).

Additional resources

- **journalctl(1)** and **taskset(1)** man pages

29.4.2. Setting the smp_affinity mask

The **smp_affinity** value is stored as a hexadecimal bit mask representing all processors in the system. Each bit configures a different CPU. The least significant bit is CPU 0.

The default value of the mask is **f**, which means that an interrupt request can be handled on any processor in the system. Setting this value to 1 means that only processor 0 can handle the interrupt.

Procedure

1. In binary, use the value 1 for CPUs that handle the interrupts. For example, to set CPU 0 and CPU 7 to handle interrupts, use **0000000010000001** as the binary code:

Table 29.1. Binary Bits for CPUs

CPU	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Binary	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

2. Convert the binary code to hexadecimal:
For example, to convert the binary code using Python:

```
>>> hex(int('0000000010000001', 2))  
  
'0x81'
```

On systems with more than 32 processors, you must delimit the **smp_affinity** values for discrete 32 bit groups. For example, if you want only the first 32 processors of a 64 processor system to service an interrupt request, use **0xffffffff,00000000**.

3. The interrupt affinity value for a particular interrupt request is stored in the associated **/proc/irq/irq_number/smp_affinity** file. Set the **smp_affinity** mask in this file:

```
# echo mask > /proc/irq/irq_number/smp_affinity
```

Additional resources

- **journalctl(1)**, **irqbalance(1)**, and **taskset(1)** man pages

CHAPTER 30. TUNING SCHEDULING POLICY

In Red Hat Enterprise Linux, the smallest unit of process execution is called a thread. The system scheduler determines which processor runs a thread, and for how long the thread runs. However, because the scheduler's primary concern is to keep the system busy, it may not schedule threads optimally for application performance.

For example, say an application on a NUMA system is running on Node A when a processor on Node B becomes available. To keep the processor on Node B busy, the scheduler moves one of the application's threads to Node B. However, the application thread still requires access to memory on Node A. But, this memory will take longer to access because the thread is now running on Node B and Node A memory is no longer local to the thread. Thus, it may take longer for the thread to finish running on Node B than it would have taken to wait for a processor on Node A to become available, and then to execute the thread on the original node with local memory access.

30.1. CATEGORIES OF SCHEDULING POLICIES

Performance sensitive applications often benefit from the designer or administrator determining where threads are run. The Linux scheduler implements a number of scheduling policies which determine where and for how long a thread runs.

The following are the two major categories of scheduling policies:

Normal policies

Normal threads are used for tasks of normal priority.

Realtime policies

Realtime policies are used for time-sensitive tasks that must complete without interruptions. Realtime threads are not subject to time slicing. This means the thread runs until they block, exit, voluntarily yield, or are preempted by a higher priority thread.

The lowest priority realtime thread is scheduled before any thread with a normal policy. For more information, see [Static priority scheduling with SCHED_FIFO](#) and [Round robin priority scheduling with SCHED_RR](#).

Additional resources

- `sched(7)`, `sched_setaffinity(2)`, `sched_getaffinity(2)`, `sched_setscheduler(2)`, and `sched_getscheduler(2)` man pages

30.2. STATIC PRIORITY SCHEDULING WITH SCHED_FIFO

The **SCHED_FIFO**, also called static priority scheduling, is a realtime policy that defines a fixed priority for each thread. This policy allows administrators to improve event response time and reduce latency. It is recommended to not execute this policy for an extended period of time for time sensitive tasks.

When **SCHED_FIFO** is in use, the scheduler scans the list of all the **SCHED_FIFO** threads in order of priority and schedules the highest priority thread that is ready to run. The priority level of a **SCHED_FIFO** thread can be any integer from **1** to **99**, where **99** is treated as the highest priority. Red Hat recommends starting with a lower number and increasing priority only when you identify latency issues.



WARNING

Because realtime threads are not subject to time slicing, Red Hat does not recommend setting a priority as 99. This keeps your process at the same priority level as migration and watchdog threads; if your thread goes into a computational loop and these threads are blocked, they will not be able to run. Systems with a single processor will eventually hang in this situation.

Administrators can limit **SCHED_FIFO** bandwidth to prevent realtime application programmers from initiating realtime tasks that monopolize the processor.

The following are some of the parameters used in this policy:

/proc/sys/kernel/sched_rt_period_us

This parameter defines the time period, in microseconds, that is considered to be one hundred percent of the processor bandwidth. The default value is **1000000 µs**, or **1 second**.

/proc/sys/kernel/sched_rt_runtime_us

This parameter defines the time period, in microseconds, that is devoted to running real-time threads. The default value is **950000 µs**, or **0.95 seconds**.

30.3. ROUND ROBIN PRIORITY SCHEDULING WITH SCHED_RR

The **SCHED_RR** is a round-robin variant of the **SCHED_FIFO**. This policy is useful when multiple threads need to run at the same priority level.

Like **SCHED_FIFO**, **SCHED_RR** is a realtime policy that defines a fixed priority for each thread. The scheduler scans the list of all **SCHED_RR** threads in order of priority and schedules the highest priority thread that is ready to run. However, unlike **SCHED_FIFO**, threads that have the same priority are scheduled in a round-robin style within a certain time slice.

You can set the value of this time slice in milliseconds with the **sched_rr_timeslice_ms** kernel parameter in the **/proc/sys/kernel/sched_rr_timeslice_ms** file. The lowest value is **1 millisecond**.

30.4. NORMAL SCHEDULING WITH SCHED_OTHER

The **SCHED_OTHER** is the default scheduling policy in Red Hat Enterprise Linux 9. This policy uses the Completely Fair Scheduler (CFS) to allow fair processor access to all threads scheduled with this policy. This policy is most useful when there are a large number of threads or when data throughput is a priority, as it allows more efficient scheduling of threads over time.

When this policy is in use, the scheduler creates a dynamic priority list based partly on the niceness value of each process thread. Administrators can change the niceness value of a process, but cannot change the scheduler's dynamic priority list directly.

30.5. SETTING SCHEDULER POLICIES

Check and adjust scheduler policies and priorities by using the **chrt** command line tool. It can start new processes with the desired properties, or change the properties of a running process. It can also be used for setting the policy at runtime.

Procedure

1. View the process ID (PID) of the active processes:

```
# ps
```

Use the **--pid** or **-p** option with the **ps** command to view the details of the particular PID.

2. Check the scheduling policy, PID, and priority of a particular process:

```
# chrt -p 468
pid 468's current scheduling policy: SCHED_FIFO
pid 468's current scheduling priority: 85

# chrt -p 476
pid 476's current scheduling policy: SCHED_OTHER
pid 476's current scheduling priority: 0
```

Here, *468* and *476* are PID of a process.

3. Set the scheduling policy of a process:

- a. For example, to set the process with PID *1000* to *SCHED_FIFO*, with a priority of *50*:

```
# chrt -f -p 50 1000
```

- b. For example, to set the process with PID *1000* to *SCHED_OTHER*, with a priority of *0*:

```
# chrt -o -p 0 1000
```

- c. For example, to set the process with PID *1000* to *SCHED_RR*, with a priority of *10*:

```
# chrt -r -p 10 1000
```

- d. To start a new application with a particular policy and priority, specify the name of the application:

```
# chrt -f 36 /bin/my-app
```

Additional resources

- **chrt(1)** man page
- [Policy Options for the chrt command](#)
- [Changing the priority of services during the boot process](#)

30.6. POLICY OPTIONS FOR THE CHRT COMMAND

Using the **chrt** command, you can view and set the scheduling policy of a process.

The following table describes the appropriate policy options, which can be used to set the scheduling policy of a process.

Table 30.1. Policy Options for the `chrt` Command

Short option	Long option	Description
-f	--fifo	Set schedule to SCHED_FIFO
-o	--other	Set schedule to SCHED_OTHER
-r	--rr	Set schedule to SCHED_RR

30.7. CHANGING THE PRIORITY OF SERVICES DURING THE BOOT PROCESS

Using the **systemd** service, it is possible to set up real-time priorities for services launched during the boot process. The *unit configuration directives* are used to change the priority of a service during the boot process.

The boot process priority change is done by using the following directives in the service section:

CPUSchedulingPolicy=

Sets the CPU scheduling policy for executed processes. It is used to set **other**, **fifo**, and **rr** policies.

CPUSchedulingPriority=

Sets the CPU scheduling priority for executed processes. The available priority range depends on the selected CPU scheduling policy. For real-time scheduling policies, an integer between **1** (lowest priority) and **99** (highest priority) can be used.

The following procedure describes how to change the priority of a service, during the boot process, using the **mcelog** service.

Prerequisites

1. Install the TuneD package:

```
# dnf install tuned
```

2. Enable and start the TuneD service:

```
# systemctl enable --now tuned
```

Procedure

1. View the scheduling priorities of running threads:

```
# tuna --show_threads
          thread      ctxt_switches
pid SCHED_ rtpri affinity voluntary nonvoluntary      cmd
1  OTHER   0  0xff  3181      292      systemd
2  OTHER   0  0xff   254       0      kthreadd
3  OTHER   0  0xff    2       0      rcu_gp
4  OTHER   0  0xff    2       0      rcu_par_gp
```

```

6  OTHER  0    0    9      0 kworker/0:0H-kblockd
7  OTHER  0  0xff 1301    1 kworker/u16:0-events_unbound
8  OTHER  0  0xff  2      0 mm_percpu_wq
9  OTHER  0    0 266      0 ksoftirqd/0
[...]
```

2. Create a supplementary **mcelog** service configuration directory file and insert the policy name and priority in this file:

```

# cat << EOF > /etc/systemd/system/mcelog.system.d/priority.conf

[SERVICE]
CPUSchedulingPolicy=fifo
CPUSchedulingPriority=20
EOF
```

3. Reload the **systemd** scripts configuration:

```
# systemctl daemon-reload
```

4. Restart the **mcelog** service:

```
# systemctl restart mcelog
```

Verification steps

- Display the **mcelog** priority set by **systemd** issue:

```

# tuna -t mcelog -P
thread  ctxt_switches
pid SCHED_rtpri affinity voluntary nonvoluntary  cmd
826  FIFO  20 0,1,2,3    13      0    mcelog
```

Additional resources

- **systemd(1)** and **tuna(8)** man pages
- [Description of the priority range](#)

30.8. PRIORITY MAP

Priorities are defined in groups, with some groups dedicated to certain kernel functions. For real-time scheduling policies, an integer between **1** (lowest priority) and **99** (highest priority) can be used.

The following table describes the priority range, which can be used while setting the scheduling policy of a process.

Table 30.2. Description of the priority range

Priority	Threads	Description
----------	---------	-------------

Priority	Threads	Description
1	Low priority kernel threads	This priority is usually reserved for the tasks that need to be just above SCHED_OTHER .
2 - 49	Available for use	The range used for typical application priorities.
50	Default hard-IRQ value	
51 - 98	High priority threads	Use this range for threads that execute periodically and must have quick response times. Do not use this range for CPU-bound threads as you will starve interrupts.
99	Watchdogs and migration	System threads that must run at the highest priority.

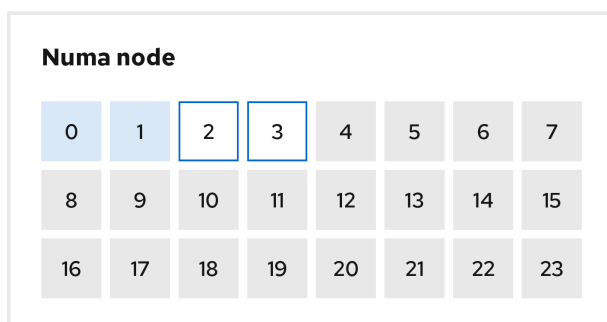
30.9. TUNED CPU-PARTITIONING PROFILE

For tuning Red Hat Enterprise Linux 9 for latency-sensitive workloads, Red Hat recommends to use the **cpu-partitioning** Tuned profile.

Prior to Red Hat Enterprise Linux 9, the low-latency Red Hat documentation described the numerous low-level steps needed to achieve low-latency tuning. In Red Hat Enterprise Linux 9, you can perform low-latency tuning more efficiently by using the **cpu-partitioning** Tuned profile. This profile is easily customizable according to the requirements for individual low-latency applications.

The following figure is an example to demonstrate how to use the **cpu-partitioning** profile. This example uses the CPU and node layout.

Figure 30.1. Figure cpu-partitioning



191_RHEL_1021

You can configure the cpu-partitioning profile in the `/etc/tuned/cpu-partitioning-variables.conf` file using the following configuration options:

Isolated CPUs with load balancing

In the `cpu-partitioning` figure, the blocks numbered from 4 to 23, are the default isolated CPUs. The kernel scheduler's process load balancing is enabled on these CPUs. It is designed for low-latency processes with multiple threads that need the kernel scheduler load balancing.

You can configure the `cpu-partitioning` profile in the `/etc/tuned/cpu-partitioning-variables.conf` file using the **`isolated_cores=cpu-list`** option, which lists CPUs to isolate that will use the kernel scheduler load balancing.

The list of isolated CPUs is comma-separated or you can specify a range using a dash, such as **`3-5`**. This option is mandatory. Any CPU missing from this list is automatically considered a housekeeping CPU.

Isolated CPUs without load balancing

In the `cpu-partitioning` figure, the blocks numbered 2 and 3, are the isolated CPUs that do not provide any additional kernel scheduler process load balancing.

You can configure the `cpu-partitioning` profile in the `/etc/tuned/cpu-partitioning-variables.conf` file using the **`no_balance_cores=cpu-list`** option, which lists CPUs to isolate that will not use the kernel scheduler load balancing.

Specifying the **`no_balance_cores`** option is optional, however any CPUs in this list must be a subset of the CPUs listed in the **`isolated_cores`** list.

Application threads using these CPUs need to be pinned individually to each CPU.

Housekeeping CPUs

Any CPU not isolated in the `cpu-partitioning-variables.conf` file is automatically considered a housekeeping CPU. On the housekeeping CPUs, all services, daemons, user processes, movable kernel threads, interrupt handlers, and kernel timers are permitted to execute.

Additional resources

- **`tuned-profiles-cpu-partitioning(7)`** man page

30.10. USING THE TUNED CPU-PARTITIONING PROFILE FOR LOW-LATENCY TUNING

This procedure describes how to tune a system for low-latency using the TuneD's **`cpu-partitioning`** profile. It uses the example of a low-latency application that can use **`cpu-partitioning`** and the CPU layout as mentioned in the [cpu-partitioning](#) figure.

The application in this case uses:

- One dedicated reader thread that reads data from the network will be pinned to CPU 2.
- A large number of threads that process this network data will be pinned to CPUs 4-23.
- A dedicated writer thread that writes the processed data to the network will be pinned to CPU 3.

Prerequisites

- You have installed the **`cpu-partitioning`** TuneD profile by using the **`dnf install tuned-profiles-cpu-partitioning`** command as root.

Procedure

1. Edit **/etc/tuned/cpu-partitioning-variables.conf** file and add the following information:

```
# Isolated CPUs with the kernel's scheduler load balancing:
isolated_cores=2-23
# Isolated CPUs without the kernel's scheduler load balancing:
no_balance_cores=2,3
```

2. Set the **cpu-partitioning** TuneD profile:

```
# tuned-adm profile cpu-partitioning
```

3. Reboot

After rebooting, the system is tuned for low-latency, according to the isolation in the **cpu-partitioning** figure. The application can use **taskset** to pin the reader and writer threads to CPUs 2 and 3, and the remaining application threads on CPUs 4-23.

Additional resources

- **tuned-profiles-cpu-partitioning(7)** man page

30.11. CUSTOMIZING THE CPU-PARTITIONING TUNED PROFILE

You can extend the TuneD profile to make additional tuning changes.

For example, the **cpu-partitioning** profile sets the CPUs to use **cstate=1**. In order to use the **cpu-partitioning** profile but to additionally change the CPU **cstate** from **cstate1** to **cstate0**, the following procedure describes a new TuneD profile named *my_profile*, which inherits the **cpu-partitioning** profile and then sets C state-0.

Procedure

1. Create the **/etc/tuned/my_profile** directory:

```
# mkdir /etc/tuned/my_profile
```

2. Create a **tuned.conf** file in this directory, and add the following content:

```
# vi /etc/tuned/my_profile/tuned.conf
[main]
summary=Customized tuning on top of cpu-partitioning
include=cpu-partitioning
[cpu]
force_latency=cstate.id:0|1
```

3. Use the new profile:

```
# tuned-adm profile my_profile
```

**NOTE**

In the shared example, a reboot is not required. However, if the changes in the *my_profile* profile require a reboot to take effect, then reboot your machine.

Additional resources

- **tuned-profiles-cpu-partitioning(7)** man page

CHAPTER 31. TUNING THE NETWORK PERFORMANCE

Tuning the network settings is a complex process with many factors to consider. For example, this includes the CPU-to-memory architecture, the amount of CPU cores, and more. Red Hat Enterprise Linux uses default settings that are optimized for most scenarios. However, in certain cases, it can be necessary to tune network settings to increase the throughput or latency or to solve problems, such as packet drops.

31.1. TUNING NETWORK ADAPTER SETTINGS

In high-speed networks with 40 Gbps and faster, certain default values of network adapter-related kernel settings can be a cause of packet drops and performance degradation. Tuning these settings can prevent such problems.

31.1.1. Increasing the ring buffers to reduce a high packet drop rate

Receive ring buffers are shared between the device driver and network interface controller (NIC). The card assigns a transmit (TX) and receive (RX) ring buffer. As the name implies, the ring buffer is a circular buffer where an overflow overwrites existing data. There are two ways to move data from the NIC to the kernel, hardware interrupts and software interrupts, also called SoftIRQs.

The kernel uses the RX ring buffer to store incoming packets until they can be processed by the device driver. The device driver drains the RX ring, typically using SoftIRQs, which puts the incoming packets into a kernel data structure called an **sk_buff** or **skb** to begin its journey through the kernel and up to the application which owns the relevant socket.

The kernel uses the TX ring buffer to hold outgoing packets which are destined for the wire. These ring buffers reside at the bottom of the stack and are a crucial point at which packet drop can occur, which in turn will adversely affect network performance.

Increase the size of an Ethernet device's ring buffers if the packet drop rate causes applications to report a loss of data, timeouts, or other issues.

Procedure

1. Display the packet drop statistics of the interface:

```
# ethtool -S enp1s0
...
rx_queue_0_drops: 97326
rx_queue_1_drops: 63783
...
```

Note that the output of the command depends on the network card and the driver.

High values in **discard** or **drop** counters indicate that the available buffer fills up faster than the kernel can process the packets. Increasing the ring buffers can help to avoid such loss.

2. Display the maximum ring buffer sizes:

```
# ethtool -g enp1s0
Ring parameters for enp1s0:
Pre-set maximums:
RX:          4096
RX Mini:     0
```

```

RX Jumbo:    16320
TX:          4096
Current hardware settings:
RX:          255
RX Mini:     0
RX Jumbo:    0
TX:          255

```

If the values in the **Pre-set maximums** section are higher than in the **Current hardware settings** section, you can change the settings in the next steps.

3. Identify the NetworkManager connection profile that uses the interface:

```

# nmcli connection show
NAME                UUID                                TYPE    DEVICE
Example-Connection a5eb6490-cc20-3668-81f8-0314a27f3f75 ethernet enp1s0

```

4. Update the connection profile, and increase the ring buffers:

- To increase the RX ring buffer, enter:

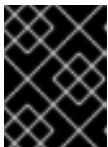
```
# nmcli connection modify Example-Connection ethtool.ring-rx 4096
```

- To increase the TX ring buffer, enter:

```
# nmcli connection modify Example-Connection ethtool.ring-tx 4096
```

5. Reload the NetworkManager connection:

```
# nmcli connection up Example-Connection
```



IMPORTANT

Depending on the driver your NIC uses, changing in the ring buffer can shortly interrupt the network connection.

Additional resources

- [ifconfig and ip commands report packet drops](#)
- [Should I be concerned about a 0.05% packet drop rate?](#)
- **ethtool(8)** man page

31.1.2. Tuning the network device backlog queue to avoid packet drops

When a network card receives packets and before the kernel protocol stack processes them, the kernel stores these packets in backlog queues. The kernel maintains a separate queue for each CPU core.

If the backlog queue for a core is full, the kernel drops all further incoming packets that the **netif_receive_skb()** kernel function assigns to this queue. If the server contains a 10 Gbps or faster network adapter or multiple 1 Gbps adapters, tune the backlog queue size to avoid this problem.

Prerequisites

- A 10 Gbps or faster or multiple 1 Gbps network adapters

Procedure

1. Determine whether tuning the backlog queue is needed, display the counters in the `/proc/net/softnet_stat` file:

```
# awk '{for (i=1; i<=NF; i++) printf strtonum("0x" $i) (i==NF?"\n":" ")}'
/proc/net/softnet_stat | column -t
221951548 0 0 0 0 0 0 0 0 0 0 0
192058677 18862 0 0 0 0 0 0 0 0 0 1
455324886 0 0 0 0 0 0 0 0 0 0 2
...
```

This **awk** command converts the values in `/proc/net/softnet_stat` from hexadecimal to decimal format and displays them in table format. Each line represents a CPU core starting with core 0.

The relevant columns are:

- First column: The total number of received frames
 - Second column: The number of dropped frames because of a full backlog queue
 - Last column: The CPU core number
2. If the values in the second column of the `/proc/net/softnet_stat` file increment over time, increase the size of the backlog queue:

- a. Display the current backlog queue size:

```
# sysctl net.core.netdev_max_backlog
net.core.netdev_max_backlog = 1000
```

- b. Create the `/etc/sysctl.d/10-netdev_max_backlog.conf` file with the following content:

```
net.core.netdev_max_backlog = 2000
```

Set the **net.core.netdev_max_backlog** parameter to a double of the current value.

- c. Load the settings from the `/etc/sysctl.d/10-netdev_max_backlog.conf` file:

```
# sysctl -p /etc/sysctl.d/10-netdev_max_backlog.conf
```

Verification

- Monitor the second column in the `/proc/net/softnet_stat` file:

```
# awk '{for (i=1; i<=NF; i++) printf strtonum("0x" $i) (i==NF?"\n":" ")}'
/proc/net/softnet_stat | column -t
```

If the values still increase, double the **net.core.netdev_max_backlog** value again. Repeat this process until the packet drop counters no longer increase.

31.1.3. Increasing the transmit queue length of a NIC to reduce the number of transmit errors

The kernel stores packets in a transmit queue before transmitting them. The default length (1000 packets) is typically sufficient for 10 Gbps and often also for 40 Gbps networks. However, in faster networks, or if you encounter an increasing number of transmit errors on an adapter, increase the queue length.

Procedure

1. Display the current transmit queue length:

```
# ip -s link show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
...
```

In this example, the transmit queue length (**qlen**) of the **enp1s0** interface is **1000**.

2. Monitor the dropped packets counter of a network interface's software transmit queue:

```
# tc -s qdisc show dev enp1s0
qdisc fq_codel 0: root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5ms interval
100ms memory_limit 32Mb ecn drop_batch 64
Sent 16889923 bytes 426862765 pkt (dropped 191980, overlimits 0 requeues 2)
...
```

3. If you encounter a high or increasing transmit error count, set a higher transmit queue length:
 - a. Identify the NetworkManager connection profile that uses this interface:

```
# nmcli connection show
NAME                UUID                                TYPE    DEVICE
Example-Connection a5eb6490-cc20-3668-81f8-0314a27f3f75 ethernet enp1s0
```

- b. Create the **/etc/NetworkManager/dispatcher.d/99-set-tx-queue-length-up** NetworkManager dispatcher script with the following content:

```
#!/bin/bash
# Set TX queue length on enp1s0 to 2000

if [ "$1" == "enp1s0" ] && [ "$2" == "up" ] ; then
    ip link set dev enp1s0 txqueuelen 2000
fi
```

- c. Set the executable bit on the **/etc/NetworkManager/dispatcher.d/99-set-tx-queue-length-up** file:

```
# chmod +x /etc/NetworkManager/dispatcher.d/99-set-tx-queue-length-up
```

- d. Reload the NetworkManager connection:

```
# nmcli connection up Example-Connection
```

Verification

1. Display the transmit queue length:

```
# ip -s link show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 2000
...
```

2. Monitor the dropped packets counter:

```
# tc -s qdisc show dev enp1s0
```

If the **dropped** counter still increases, double the transmit queue length again. Repeat this process until the counter no longer increases.

31.2. TUNING IRQ BALANCING

On multi-core hosts, you can increase the performance by ensuring that Red Hat Enterprise Linux balances interrupt queues (IRQs) to distribute the interrupts across CPU cores.

31.2.1. Interrupts and interrupt handlers

When a network interface controller (NIC) receives incoming data, it copies the data into kernel buffers by using Direct Memory Access (DMA). The NIC then notifies the kernel about this data by triggering a hard interrupt. These interrupts are processed by interrupt handlers which do minimal work, as they have already interrupted another task and the handlers cannot interrupt themselves. Hard interrupts can be costly in terms of CPU usage, especially if they use kernel locks.

The hard interrupt handler then leaves the majority of packet reception to a software interrupt request (SoftIRQ) process. The kernel can schedule these processes more fairly.

Example 31.1. Displaying hardware interrupts

The kernel stores the interrupt counters in the `/proc/interrupts` file. To display the counters for a specific NIC, such as `enp1s0`, enter:

```
# egrep "CPU|enp1s0" /proc/interrupts
    CPU0   CPU1   CPU2   CPU3   CPU4   CPU5
105: 141606      0      0      0      0      0 IR-PCI-MSI-edge enp1s0-rx-0
106:   0 141091      0      0      0      0 IR-PCI-MSI-edge enp1s0-rx-1
107:   2      0 163785      0      0      0 IR-PCI-MSI-edge enp1s0-rx-2
108:   3      0      0 194370      0      0 IR-PCI-MSI-edge enp1s0-rx-3
109:   0      0      0      0      0      0 IR-PCI-MSI-edge enp1s0-tx
```

Each queue has an interrupt vector in the first column assigned to it. The kernel initializes these vectors when the system boots or when a user loads the NIC driver module. Each receive (**RX**) and transmit (**TX**) queue is assigned a unique vector that informs the interrupt handler which NIC or queue the interrupt is coming from. The columns represent the number of incoming interrupts for every CPU core.

31.2.2. Software interrupt requests

Software interrupt requests (SoftIRQs) clear the receive ring buffers of network adapters. The kernel schedules SoftIRQ routines to run at a time when other tasks will not be interrupted. On Red Hat Enterprise Linux, processes named **ksoftirqd/cpu-number** run these routines and call driver-specific code functions.

To monitor the SoftIRQ counters for each CPU core, enter:

```
# watch -n1 'egrep "CPU|NET_RX|NET_TX" /proc/softirqs'
```

	CPU0	CPU1	CPU2	CPU3	CPU4	CPU5	CPU6	CPU7
NET_TX:	49672	52610	28175	97288	12633	19843	18746	220689
NET_RX:	96	1615	789	46	31	1735	1315	470798

The command dynamically updates the output. Press **Ctrl+C** to interrupt the output.

31.2.3. NAPI Polling

New API (NAPI) is an extension to the device driver packet processing framework to improve the efficiency of incoming network packets. Hard interrupts are expensive because they usually cause a context switch from the kernel space to the user space and back again, and cannot interrupt themselves. Even with interrupt coalescence, the interrupt handler monopolizes a CPU core completely. With NAPI, the driver can use a polling mode instead of being hard-interrupted by the kernel for every packet that is received.

Under normal operation, the kernel issues an initial hard interrupt, followed by a soft interrupt request (SoftIRQ) handler that polls the network card using NAPI routines. To prevent SoftIRQs from monopolizing a CPU core, the polling routine has a budget that determines the CPU time the SoftIRQ can consume. On completion of the SoftIRQ poll routine, the kernel exits the routine and schedules it to run again at a later time to repeat the process of receiving packets from the network card.

31.2.4. The irqbalance service

On systems both with and without Non-Uniform Memory Access (NUMA) architecture, the **irqbalance** service balances interrupts effectively across CPU cores, based on system conditions. The **irqbalance** service runs in the background and monitors the CPU load every 10 seconds. The service moves interrupts to other CPU cores when a CPU's load is too high. As a result, the system performs well and handles load more efficiently.

If **irqbalance** is not running, usually the CPU core 0 handles most of the interrupts. Even at moderate load, this CPU core can become busy trying to handle the workload of all the hardware in the system. As a consequence, interrupts or interrupt-based work can be missed or delayed. This can result in low network and storage performance, packet loss, and potentially other issues.



IMPORTANT

Disabling **irqbalance** can negatively impact the network throughput.

On systems with only a single CPU core, the **irqbalance** service provides no benefit and exits on its own.

By default, the **irqbalance** service is enabled and running on Red Hat Enterprise Linux. To re-enable the service if you disabled it, enter:

```
# systemctl enable --now irqbalance
```

Additional resources

- [Do we need irqbalance?](#) solution
- [How should I configure network interface IRQ channels?](#) solution

31.2.5. Increasing the time SoftIRQs can run on the CPU

If SoftIRQs do not run long enough, the rate of incoming data could exceed the kernel's capability to drain the buffer fast enough. As a result, the network interface controller (NIC) buffers overflow and packets are lost.

If **softirqd** processes could not retrieve all packets from interfaces in one NAPI polling cycle, it is an indicator that the SoftIRQs do not have enough CPU time. This could be the case on hosts with fast NICs, such as 10 Gbps and faster. If you increase the values of the **net.core.netdev_budget** and **net.core.netdev_budget_usecs** kernel parameters, you can control the time and number of packets **softirqd** can process in a polling cycle.

Procedure

1. To determine whether tuning the **net.core.netdev_budget** parameter is needed, display the counters in the **/proc/net/softnet_stat** file:

```
# awk '{for (i=1; i<=NF; i++) printf strtonum("0x" $i) (i==NF?"\n":" ")}'
/proc/net/softnet_stat | column -t
221951548 0 0 0 0 0 0 0 0 0 0 0 0 0
192058677 0 20380 0 0 0 0 0 0 0 0 0 0 1
455324886 0 0 0 0 0 0 0 0 0 0 0 0 2
...
```

This **awk** command converts the values in **/proc/net/softnet_stat** from hexadecimal to decimal format and displays them in the table format. Each line represents a CPU core starting with core 0.

The relevant columns are:

- First column: The total number of received frames.
 - Third column: The number times **softirqd** processes that could not retrieve all packets from interfaces in one NAPI polling cycle.
 - Last column: The CPU core number.
2. If the counters in the third column of the **/proc/net/softnet_stat** file increment over time, tune the system:
 - a. Display the current values of the **net.core.netdev_budget_usecs** and **net.core.netdev_budget** parameters:

```
# sysctl net.core.netdev_budget_usecs net.core.netdev_budget
net.core.netdev_budget_usecs = 2000
net.core.netdev_budget = 300
```

With these settings, **softirqd** processes have up to 2000 microseconds to process up to 300 messages from the NIC in one polling cycle. Polling ends based on which condition is met first.

- b. Create the `/etc/sysctl.d/10-netdev_budget.conf` file with the following content:

```
net.core.netdev_budget = 600
net.core.netdev_budget_usecs = 4000
```

Set the parameters to a double of their current values.

- c. Load the settings from the `/etc/sysctl.d/10-netdev_budget.conf` file:

```
# sysctl -p /etc/sysctl.d/10-netdev_budget.conf
```

Verification

- Monitor the third column in the `/proc/net/softnet_stat` file:

```
# awk '{for (i=1; i<=NF; i++) printf strtonum("0x" $i) (i==NF?"\n":" ")}'
/proc/net/softnet_stat | column -t
```

If the values still increase, set `net.core.netdev_budget_usecs` and `net.core.netdev_budget` to higher values. Repeat this process until the counters no longer increase.

31.3. IMPROVING THE NETWORK LATENCY

CPU power management features can cause unwanted delays in time-sensitive application processing. You can disable some or all of these power management features to improve the network latency.

For example, if the latency is higher when the server is idle than under heavy load, CPU power management settings could influence the latency.



IMPORTANT

Disabling CPU power management features can cause a higher power consumption and heat loss.

31.3.1. How the CPU power states influence the network latency

The consumption state (C-states) of CPUs optimize and reduce the power consumption of computers. The C-states are numbered, starting at C0. In C0, the processor is fully powered and executing. In C1, the processor is fully powered but not executing. The higher the number of the C-state, the more components the CPU turns off.

Whenever a CPU core is idle, the built-in power saving logic steps in and attempts to move the core from the current C-state to a higher one by turning off various processor components. If the CPU core must process data, Red Hat Enterprise Linux (RHEL) sends an interrupt to the processor to wake up the core and set its C-state back to C0.

Moving out of deep C-states back to C0 takes time due to turning power back on to various components of the processor. On multi-core systems, it can also happen that many of the cores are simultaneously idle and, therefore, in deeper C-states. If RHEL tries to wake them up at the same time, the kernel can generate a large number of Inter-Processor Interrupts (IPIs) while all cores return from deep C-states. Due to locking that is required while processing interrupts, the system can then stall for some time while handling all the interrupts. This can result in large delays in the application response to events.

Example 31.2. Displaying times in C-state per core

The **Idle Stats** page in the PowerTOP application displays how much time the CPU cores spend in each C-state:

Pkg(HW)	Core(HW)	CPU(OS) 0	CPU(OS) 4
	C0 active	2.5%	2.2%
	POLL	0.0%	0.0 ms 0.0%
	C1	0.1%	0.2 ms 0.0%
C2 (pc2) 63.7%			
C3 (pc3) 0.0%	C3 (cc3) 0.1%	C3	0.1% 0.1 ms 0.1% 0.1 ms
C6 (pc6) 0.0%	C6 (cc6) 8.3%	C6	5.2% 0.6 ms 6.0% 0.6 ms
C7 (pc7) 0.0%	C7 (cc7) 76.6%	C7s	0.0% 0.0 ms 0.0% 0.0 ms
C8 (pc8) 0.0%	C8	6.3%	0.9 ms 5.8% 0.8 ms
C9 (pc9) 0.0%	C9	0.4%	3.7 ms 2.2% 2.2 ms
C10 (pc10) 0.0%			
	C10	80.8%	3.7 ms 79.4% 4.4 ms
	C1E	0.1%	0.1 ms 0.1% 0.1 ms
...			

Additional resources

- [Managing power consumption with PowerTOP](#)

31.3.2. C-state settings in the EFI firmware

In most systems with an EFI firmware, you can enable and disable the individual consumption states (C-states). However, on Red Hat Enterprise Linux (RHEL), the idle driver determines whether the kernel uses the settings from the firmware:

- **intel_idle**: This is the default driver on hosts with an Intel CPU and ignores the C-state settings from the EFI firmware.
- **acpi_idle**: RHEL uses this driver on hosts with CPUs from vendors other than Intel and if **intel_idle** is disabled. By default, the **acpi_idle** driver uses the C-state settings from the EFI firmware.

Additional resources

- `/usr/share/doc/kernel-doc-<version>/Documentation/admin-guide/pm/cpuidle.rst` provided by the **kernel-doc** package

31.3.3. Disabling C-states by using a custom Tuned profile

The Tuned service uses the Power Management Quality of Service (**PMQOS**) interface of the kernel to set consumption states (C-states) locking. The kernel idle driver can communicate with this interface to dynamically limit the C-states. This prevents that administrators must hard code a maximum C-state value by using kernel command line parameters.

Prerequisites

- The **tuned** package is installed.

- The **tuned** service is enabled and running.

Procedure

1. Display the active profile:

```
# tuned-adm active
Current active profile: network-latency
```

2. Create a directory for the custom TuneD profile:

```
# mkdir /etc/tuned/network-latency-custom
```

3. Create the `/etc/tuned/network-latency-custom/tuned.conf` file with the following content:

```
[main]
include=network-latency

[cpu]
force_latency=cstate.id:1|2
```

This custom profile inherits all settings from the **network-latency** profile. The **force_latency** TuneD parameter specifies the latency in microseconds (μ s). If the C-state latency is higher than the specified value, the idle driver in Red Hat Enterprise Linux prevents the CPU from moving to a higher C-state. With **force_latency=cstate.id:1|2**, TuneD first checks if the `/sys/devices/system/cpu/cpu_<number>/_cpuidle/state_<cstate.id>/_` directory exists. In this case, TuneD reads the latency value from the **latency** file in this directory. If the directory does not exist, TuneD uses 2 microseconds as a fallback value.

4. Activate the ***network-latency-custom*** profile:

```
# tuned-adm profile network-latency-custom
```

Additional resources

- [Getting started with TuneD](#)
- [Customizing TuneD profiles](#)

31.3.4. Disabling C-states by using a kernel command line option

The **processor.max_cstate** and **intel_idle.max_cstat** kernel command line parameters configure the maximum consumption states (C-state) CPU cores can use. For example, setting the parameters to **1** ensures that the CPU will never request a C-state below C1.

Use this method to test whether the latency of applications on a host are being affected by C-states. To not hard code a specific state, consider using a more dynamic solution. See [Disabling C-states by using a custom TuneD profile](#).

Prerequisites

- The **tuned** service is not running or configured to not update C-state settings.

Procedure

1. Display the idle driver the system uses:

```
# cat /sys/devices/system/cpu/cpuidle/current_driver
intel_idle
```

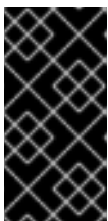
2. If the host uses the **intel_idle** driver, set the **intel_idle.max_cstate** kernel parameter to define the highest C-state that CPU cores should be able to use:

```
# grubby --update-kernel=ALL --args="intel_idle.max_cstate=0"
```

Setting **intel_idle.max_cstate=0** disables the **intel_idle** driver. Consequently, the kernel uses the **acpi_idle** driver that uses the C-state values set in the EFI firmware. For this reason, also set **processor.max_cstate** to override these C-state settings.

3. On every host, independent from the CPU vendor, set the highest C-state that CPU cores should be able to use:

```
# grubby --update-kernel=ALL --args="processor.max_cstate=0"
```



IMPORTANT

If you set **processor.max_cstate=0** in addition to **intel_idle.max_cstate=0**, the **acpi_idle** driver overrides the value of **processor.max_cstate** and sets it to **1**. As a result, with **processor.max_cstate=0** **intel_idle.max_cstate=0**, the highest C-state the kernel will use is C1, not C0.

4. Restart the host for the changes to take effect:

```
# reboot
```

Verification

1. Display the maximum C-state:

```
# cat /sys/module/processor/parameters/max_cstate
1
```

2. If the host uses the **intel_idle** driver, display the maximum C-state:

```
# cat /sys/module/intel_idle/parameters/max_cstate
0
```

Additional resources

- [What are CPU "C-states" and how to disable them if needed?](#)
- `/usr/share/doc/kernel-doc-<version>/Documentation/admin-guide/pm/cpuidle.rst` provided by the **kernel-doc** package

31.4. IMPROVING THE THROUGHPUT OF LARGE AMOUNTS OF CONTIGUOUS DATA STREAMS

According to the IEEE 802.3 standard, a default Ethernet frame without Virtual Local Area Network (VLAN) tag has a maximum size of 1518 bytes. Each of these frames includes an 18 bytes header, leaving 1500 bytes for payload. Consequently, for every 1500 bytes of data the server transmits over the network, 18 bytes (1.2%) Ethernet frame header are overhead and transmitted as well. Headers from layer 3 and 4 protocols increase the overhead per packet further.

Consider employing jumbo frames to save overhead if hosts on your network often send numerous contiguous data streams, such as backup servers or file servers hosting numerous huge files. Jumbo frames are non-standardized frames that have a larger Maximum Transmission Unit (MTU) than the standard Ethernet payload size of 1500 bytes. For example, if you configure jumbo frames with the maximum allowed MTU of 9000 bytes payload, the overhead of each frame reduces to 0.2%.

Depending on the network and services, it can be beneficial to enable jumbo frames only in specific parts of a network, such as the storage backend of a cluster. This avoids packet fragmentation.

31.4.1. Considerations before configuring jumbo frames

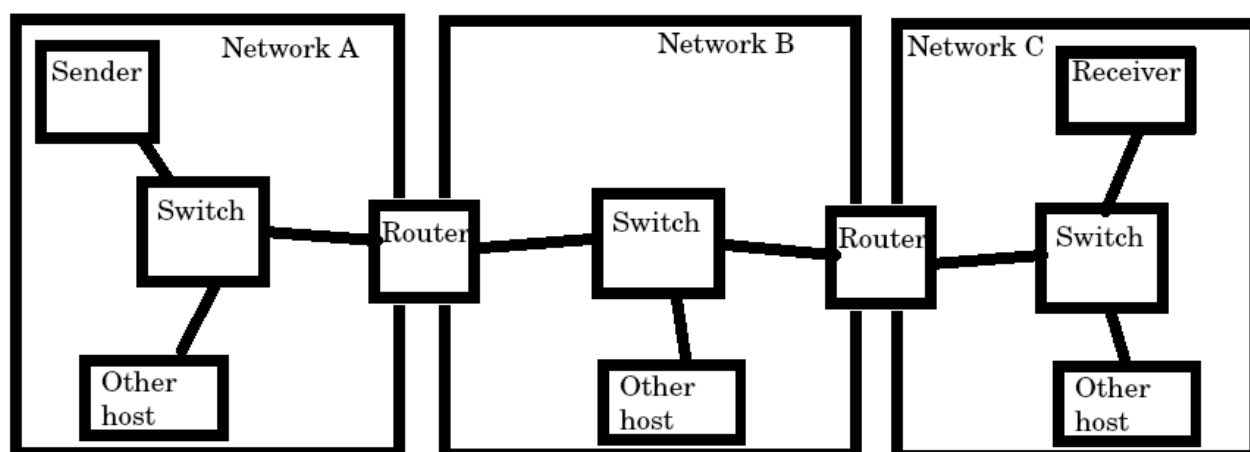
Depending on your hardware, applications, and services in your network, jumbo frames can have different impacts. Decide carefully whether enabling jumbo frames provides a benefit in your scenario.

Prerequisites

All network devices on the transmission path must support jumbo frames and use the same Maximum Transmission Unit (MTU) size. In the opposite case, you can face the following problems:

- Dropped packets.
- Higher latency due to fragmented packets.
- Increased risk of packet loss caused by fragmentation. For example, if a router fragments a single 9000-bytes frame into six 1500-bytes frames, and any of those 1500-byte frames are lost, the whole frame is lost because it cannot be reassembled.

In the following diagram, all hosts in the three subnets must use the same MTU if a host from network A sends a packet to a host in network C:



Benefits of jumbo frames

- Higher throughput: Each frame contains more user data while the protocol overhead is fixed.
- Lower CPU utilization: Jumbo frames cause fewer interrupts and, therefore, save CPU cycles.

Drawbacks of jumbo frames

- Higher latency: Larger frames delay packets that follow.
- Increased memory buffer usage: Larger frames can fill buffer queue memory more quickly.

31.4.2. Configuring the MTU in an existing NetworkManager connection profile

If your network requires a different Maximum Transmission Unit (MTU) than the default, you can configure this setting in the corresponding NetworkManager connection profile.

Jumbo frames are network packets with a payload of between 1500 and 9000 bytes. All devices in the same broadcast domain have to support those frames.

Prerequisites

- All devices in the broadcast domain use the same MTU.
- You know the MTU of the network.
- You already configured a connection profile for the network with the divergent MTU.

Procedure

1. Optional: Display the current MTU:

```
# ip link show
...
3: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 52:54:00:74:79:56 brd ff:ff:ff:ff:ff:ff
...
```

2. Optional: Display the NetworkManager connection profiles:

```
# nmcli connection show
NAME      UUID                                  TYPE      DEVICE
Example   f2f33f29-bb5c-3a07-9069-be72eaec3ecf ethernet  enp1s0
...
```

3. Set the MTU in the profile that manages the connection to the network with the divergent MTU:

```
# nmcli connection modify Example mtu 9000
```

4. Reactivate the connection:

```
# nmcli connection up Example
```

Verification

1. Display the MTU setting:

```
# ip link show
...
3: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 52:54:00:74:79:56 brd ff:ff:ff:ff:ff:ff
...
```

2. Verify that no host on the transmission paths fragments the packets:

- On the receiver side, display the IP reassembly statistics of the kernel:

```
# nstat -az IpReasm*
#kernel
IpReasmTimeout 0 0.0
IpReasmReqds 0 0.0
IpReasmOKs 0 0.0
IpReasmFails 0 0.0
```

If the counters return **0**, packets were not reassembled.

- On the sender side, transmit an ICMP request with the prohibit-fragmentation-bit:

```
# ping -c1 -Mdo -s 8972 destination_host
```

If the command succeeds, the packet was not fragmented.

Calculate the value for the **-s** packet size option as follows: MTU size - 8 bytes ICMP header - 20 bytes IPv4 header = packet size

31.5. TUNING TCP CONNECTIONS FOR HIGH THROUGHPUT

Tune TCP-related settings on Red Hat Enterprise Linux to increase the throughput, reduce the latency, or prevent problems, such as packet loss.

31.5.1. Testing the TCP throughput using iperf3

The **iperf3** utility provides a server and client mode to perform network throughput tests between two hosts.



NOTE

The throughput of applications depends on many factors, such as the buffer sizes that the application uses. Therefore, the results measured with testing utilities, such as **iperf3**, can be significantly different from those of applications on a server under production workload.

Prerequisites

- The **iperf3** package is installed on both the client and server.
- No other services on either host cause network traffic that substantially affects the test result.

- For 40 Gbps and faster connections, the network card supports Accelerated Receive Flow Steering (ARFS) and the feature is enabled on the interface.

Procedure

1. Optional: Display the maximum network speed of the network interface controller (NIC) on both the server and client:

```
# ethtool enp1s0 | grep "Speed"
Speed: 100000Mb/s
```

2. On the server:

- a. Temporarily open the default **iperf3** TCP port 5201 in the **firewalld** service:

```
# firewall-cmd --add-port=5201/tcp
# firewall-cmd --reload
```

- b. Start **iperf3** in server mode:

```
# iperf3 --server
```

The service now is waiting for incoming client connections.

3. On the client:

- a. Start measuring the throughput:

```
# iperf3 --time 60 --zerocopy --client 192.0.2.1
```

- **--time <seconds>**: Defines the time in seconds when the client stops the transmission. Set this parameter to a value that you expect to work and increase it in later measurements. If the server sends packets at a faster rate than the devices on the transmit path or the client can process, packets can be dropped.
- **--zerocopy**: Enables a zero copy method instead of using the **write()** system call. You require this option only if you want to simulate a zero-copy-capable application or to reach 40 Gbps and more on a single stream.
- **--client <server>**: Enables the client mode and sets the IP address or name of the server that runs the **iperf3** server.

4. Wait until **iperf3** completes the test. Both the server and the client display statistics every second and a summary at the end. For example, the following is a summary displayed on a client:

```
[ ID] Interval      Transfer  Bitrate    Retr
[  5] 0.00-60.00 sec 101 GBytes 14.4 Gbits/sec 0  sender
[  5] 0.00-60.04 sec 101 GBytes 14.4 Gbits/sec    receiver
```

In this example, the average bitrate was 14.4 Gbps.

5. On the server:

- a. Press **Ctrl+C** to stop the **iperf3** server.

- b. Close the TCP port 5201 in **firewalld**:

```
# firewall-cmd --remove-port=5201/tcp
# firewall-cmd --reload
```

Additional resources

- **iperf3(1)** man page

31.5.2. The system-wide TCP socket buffer settings

Socket buffers temporarily store data that the kernel has received or should send:

- The read socket buffer holds packets that the kernel has received but which the application has not read yet.
- The write socket buffer holds packets that an application has written to the buffer but which the kernel has not passed to the IP stack and network driver yet.

If a TCP packet is too large and exceeds the buffer size or packets are sent or received at a too fast rate, the kernel drops any new incoming TCP packet until the data is removed from the buffer. In this case, increasing the socket buffers can prevent packet loss.

Both the **net.ipv4.tcp_rmem** (read) and **net.ipv4.tcp_wmem** (write) socket buffer kernel settings contain three values:

```
net.ipv4.tcp_rmem = 4096 131072 6291456
net.ipv4.tcp_wmem = 4096 16384 4194304
```

The displayed values are in bytes and Red Hat Enterprise Linux uses them in the following way:

- The first value is the minimum buffer size. New sockets cannot have a smaller size.
- The second value is the default buffer size. If an application sets no buffer size, this is the default value.
- The third value is the maximum size of automatically tuned buffers. Using the **setsockopt()** function with the **SO_SNDBUF** socket option in an application disables this maximum buffer size.

Note that the **net.ipv4.tcp_rmem** and **net.ipv4.tcp_wmem** parameters set the socket sizes for both the IPv4 and IPv6 protocols.

31.5.3. Increasing the system-wide TCP socket buffers

The system-wide TCP socket buffers temporarily store data that the kernel has received or should send. Both **net.ipv4.tcp_rmem** (read) and **net.ipv4.tcp_wmem** (write) socket buffer kernel settings each contain three settings: A minimum, default, and maximum value.



IMPORTANT

Setting too large buffer sizes wastes memory. Each socket can be set to the size that the application requests, and the kernel doubles this value. For example, if an application requests a 256 KiB socket buffer size and opens 1 million sockets, the system can use up to 512 GB RAM (512 KiB x 1 million) only for the potential socket buffer space.

Additionally, a too large value for the maximum buffer size can increase the latency.

Prerequisites

- You encountered a significant rate of dropped TCP packets.

Procedure

1. Determine the latency of the connection. For example, ping from the client to server to measure the average Round Trip Time (RTT):

```
# ping -c 10 server.example.com
...
--- server.example.com ping statistics ---
 10 packets transmitted, 10 received, 0% packet loss, time 9014ms
 rtt min/avg/max/mdev = 117.208/117.056/119.333/0.616 ms
```

In this example, the latency is 117 ms.

2. Use the following formula to calculate the Bandwidth Delay Product (BDP) for the traffic you want to tune:

```
connection speed in bytes * latency in ms = BDP in bytes
```

For example, to calculate the BDP for a 10 Gbps connection that has a 117 ms latency:

```
(10 * 1000 * 1000 * 1000 / 8) * 117 = 10683760 bytes
```

3. Create the `/etc/sysctl.d/10-tcp-socket-buffers.conf` file and either set the maximum read or write buffer size, or both, based on your requirements:

```
net.ipv4.tcp_rmem = 4096 262144 21367520
net.ipv4.tcp_wmem = 4096 24576 21367520
```

Specify the values in bytes. Use the following rule of thumb when you try to identify optimized values for your environment:

- Default buffer size (second value): Increase this value only slightly or set it to **524288** (512 KiB) at most. A too high default buffer size can cause buffer collapsing and, consequently, latency spikes.
 - Maximum buffer size (third value): A value double to triple of the BDP is often sufficient.
4. Load the settings from the `/etc/sysctl.d/10-tcp-socket-buffers.conf` file:

```
# sysctl -p /etc/sysctl.d/10-tcp-socket-buffers.conf
```

5. Configure your applications to use a larger socket buffer size. The third value in the **net.ipv4.tcp_rmem** and **net.ipv4.tcp_wmem** parameters defines the maximum buffer size that the **setsockopt()** function in an application can request.
For further details, see the documentation of the programming language of your application. If you are not the developer of the application, contact the developer.
6. If you have changed the second value in the **net.ipv4.tcp_rmem** or **net.ipv4.tcp_wmem** parameter, restart the applications to use the new TCP buffer sizes.
If you have changed only the third value, you do not need to restart the application because auto-tuning applies these settings dynamically.

Verification

1. Optional: [Test the TCP throughput using iperf3](#).
2. Monitor the packet drop statistics using the same method that you used when you encountered the packet drops.
If packet drops still occur but at a lower rate, increase the buffer sizes further.

Additional resources

- [What are the implications of changing socket buffer sizes?](#) solution
- **tcp(7)** man page
- **socket(7)** man page

31.5.4. TCP Window Scaling

The TCP Window Scaling feature, which is enabled by default in Red Hat Enterprise Linux, is an extension of the TCP protocol that significantly improves the throughput.

For example, on a 1 Gbps connection with 1.5 ms Round Trip Time (RTT):

- With TCP Window Scaling enabled, approximately 630 Mbps are realistic.
- With TCP Window Scaling disabled, the throughput goes down to 380 Mbps.

One of the features TCP provides is flow control. With flow control, a sender can send as much data as the receiver can receive, but no more. To achieve this, the receiver advertises a **window** value, which is the amount of data a sender can send.

TCP originally supported window sizes up to 64 KiB, but at high Bandwidth Delay Products (BDP), this value becomes a restriction because the sender cannot send more than 64 KiB at a time. High-speed connections can transfer much more than 64 KiB of data at a given time. For example, a 10 Gbps link with 1 ms of latency between systems can have more than 1 MiB of data in transit at a given time. It would be inefficient if a host sends only 64 KiB, then pauses until the other host receives that 64 KiB.

To remove this bottleneck, the TCP Window Scaling extension allows the TCP window value to be arithmetically shifted left to increase the window size beyond 64 KiB. For example, the largest window value of **65535** shifted 7 places to the left, resulting in a window size of almost 8 MiB. This enables transferring much more data at a given time.

TCP Window Scaling is negotiated during the three-way TCP handshake that opens every TCP connection. Both sender and receiver must support TCP Window Scaling for the feature to work. If either or both participants do not advertise window scaling ability in their handshake, the connection

reverts to using the original 16-bit TCP window size.

By default, TCP Window Scaling is enabled in Red Hat Enterprise Linux:

```
# sysctl net.ipv4.tcp_window_scaling
net.ipv4.tcp_window_scaling = 1
```

If TCP Window Scaling is disabled (**0**) on your server, revert the setting in the same way as you set it.

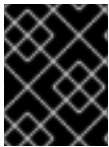
Additional resources

- [RFC 1323: TCP Extensions for High Performance](#)
- [Configuring kernel parameters at runtime](#)

31.5.5. How TCP SACK reduces the packet drop rate

The TCP Selective Acknowledgment (TCP SACK) feature, which is enabled by default in Red Hat Enterprise Linux (RHEL), is an enhancement of the TCP protocol and increases the efficiency of TCP connections.

In TCP transmissions, the receiver sends an ACK packet to the sender for every packet it receives. For example, a client sends the TCP packets 1-10 to the server but the packets number 5 and 6 get lost. Without TCP SACK, the server drops packets 7-10, and the client must retransmit all packets from the point of loss, which is inefficient. With TCP SACK enabled on both hosts, the client must re-transmit only the lost packets 5 and 6.



IMPORTANT

Disabling TCP SACK decreases the performance and causes a higher packet drop rate on the receiver side in a TCP connection.

By default, TCP SACK is enabled in RHEL. To verify:

```
# sysctl net.ipv4.tcp_sack
1
```

If TCP SACK is disabled (**0**) on your server, revert the setting in the same way as you set it.

Additional resources

- [RFC 2018: TCP Selective Acknowledgment Options](#)
- [Should I be concerned about a 0.05% packet drop rate?](#) solution
- [Configuring kernel parameters at runtime](#)

31.6. TUNING UDP CONNECTIONS

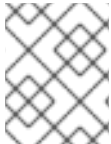
Before you start tuning Red Hat Enterprise Linux to improve the throughput of UDP traffic, it is important to have the realistic expectations. UDP is a simple protocol. Compared to TCP, UDP does not contain features, such as flow control, congestion control, and data reliability. This makes it difficult to

reach reliable communication over UDP with a throughput rate that is close to the maximum speed of the network interface controller (NIC).

31.6.1. Detecting packet drops

There are multiple levels in the network stack in which the kernel can drop packets. Red Hat Enterprise Linux provides different utilities to display statistics of these levels. Use them to identify potential problems.

Note that you can ignore a very small rate of dropped packets. However, if you encounter a significant rate, consider tuning measures.



NOTE

The kernel drops network packets if the networking stack cannot handle the incoming traffic.

Procedure

1. Identify if the network interface controller (NIC) drops packets:

- a. Display the NIC and driver-specific statistics:

```
# ethtool -S enp1s0
NIC statistics:
...
rx_queue_0_drops: 17657
...
```

The naming of the statistics and if they are available depend on the NIC and the driver.

- b. Display the interface statistics:

```
# ip -s link show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
UP mode DEFAULT group default qlen 1000
link/ether 52:54:00:74:79:56 brd ff:ff:ff:ff:ff:ff
RX:  bytes packets errors dropped missed mcast_
84697611107 56866482 0 10904 0 0
TX:  bytes packets errors dropped carrier collsns_
5540028184 3722234 0 0 0 0
```

RX represents the statistics of received packets and **TX** of transmitted packets.

2. Identify UDP protocol-specific packet drops due to too small socket buffers or slow application processing:

```
# nstat -az UdpSndbufErrors UdpRcvbufErrors
#kernel
UdpSndbufErrors      4 0.0
UdpRcvbufErrors    45716659 0.0
```

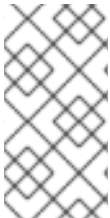
The second column in the output lists the counters.

Additional resources

- [RHEL network interface dropping packets](#) solution
- [Should I be concerned about a 0.05% packet drop rate?](#) solution

31.6.2. Testing the UDP throughput using iperf3

The **iperf3** utility provides a server and client mode to perform network throughput tests between two hosts.



NOTE

The throughput of applications depend on many factors, such as the buffer sizes that the application uses. Therefore, the results measured with testing utilities, such as **iperf3**, can significantly be different from those of applications on a server under production workload.

Prerequisites

- The **iperf3** package is installed on both the client and server.
- No other services on both hosts cause network traffic that substantially affects the test result.
- Optional: You increased the maximum UDP socket sizes on both the server and the client. For details, see [Increasing the system-wide UDP socket buffers](#).

Procedure

1. Optional: Display the maximum network speed of the network interface controller (NIC) on both the server and client:

```
# ethtool enp1s0 | grep "Speed"
Speed: 10000Mb/s
```

2. On the server:

- a. Display the maximum UDP socket read buffer size, and note the value:

```
# sysctl net.core.rmem_max
net.core.rmem_max = 16777216
```

The displayed value is in bytes.

- b. Temporarily open the default **iperf3** port 5201 in the **firewalld** service:

```
# firewall-cmd --add-port=5201/tcp --add-port=5201/udp
# firewall-cmd --reload
```

Note that, **iperf3** opens only a TCP socket on the server. If a clients wants to use UDP, it first connects to this TCP port, and then the server opens a UDP socket on the same port number for performing the UDP traffic throughput test. For this reason, you must open port 5201 for both the TCP and UDP protocol in the local firewall.

- c. Start **iperf3** in server mode:

iperf3 --server

The service now waits for incoming client connections.

3. On the client:

- a. Display the Maximum Transmission Unit (MTU) of the interface that the client will use for the connection to the server, and note the value:

```
# ip link show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
UP mode DEFAULT group default qlen 1000
...
```

- b. Display the maximum UDP socket write buffer size, and note the value:

```
# sysctl net.core.wmem_max
net.core.wmem_max = 16777216
```

The displayed value is in bytes.

- c. Start measuring the throughput:

```
# iperf3 --udp --time 60 --window 16777216 --length 1472 --bitrate 2G --client
192.0.2.1
```

- **--udp**: Use the UDP protocol for the test.
- **--time <seconds>**: Defines the time in seconds when the client stops the transmission.
- **--window <size>**: Sets the UDP socket buffer size. Ideally, the sizes are the same on both the client and server. In case that they are different, set this parameter to the value that is smaller: **net.core.wmem_max** on the client or **net.core.rmem_max** on the server.
- **--length <size>**: Sets the length of the buffer to read and write. Set this option to the largest unfragmented payload. Calculate the ideal value as follows: MTU - IP header (20 bytes for IPv4 and 40 bytes for IPv6) - 8 bytes UDP header.
- **--bitrate <rate>**: Limits the bit rate to the specified value in bits per second. You can specify units, such as **2G** for 2 Gbps.
Set this parameter to a value that you expect to work and increase it in later measurements. If the server sends packets at a faster rate than the devices on the transmit path or the client can process them, packets can be dropped.
- **--client <server>**: Enables the client mode and sets the IP address or name of the server that runs the **iperf3** server.

4. Wait until **iperf3** completes the test. Both the server and the client display statistics every second and a summary at the end. For example, the following is a summary displayed on a client:

```
[ ID] Interval      Transfer    Bitrate      Jitter  Lost/Total Datagrams
[ 5] 0.00-60.00 sec  14.0 GBytes  2.00 Gbits/sec  0.000 ms  0/10190216 (0%) sender
[ 5] 0.00-60.04 sec  14.0 GBytes  2.00 Gbits/sec  0.002 ms  0/10190216 (0%) receiver
```

In this example, the average bit rate was 2 Gbps, and no packets were lost.

5. On the server:
 - a. Press **Ctrl+C** to stop the **iperf3** server.
 - b. Close port 5201 in **firewalld**:

```
# firewall-cmd --remove-port=5201/tcp --remove-port=5201/udp
# firewall-cmd --reload
```

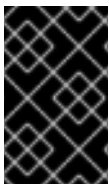
Additional resources

- **iperf3(1)** man page

31.6.3. Impact of the MTU size on UDP traffic throughput

If your application uses a large UDP message size, using jumbo frames can improve the throughput. According to the IEEE 802.3 standard, a default Ethernet frame without Virtual Local Area Network (VLAN) tag has a maximum size of 1518 bytes. Each of these frames includes an 18 bytes header, leaving 1500 bytes for payload. Consequently, for every 1500 bytes of data the server transmits over the network, 18 bytes (1.2%) are overhead.

Jumbo frames are non-standardized frames that have a larger Maximum Transmission Unit (MTU) than the standard Ethernet payload size of 1500 bytes. For example, if you configure jumbo Frames with the maximum allowed MTU of 9000 bytes payload, the overhead of each frame reduces to 0.2%.



IMPORTANT

All network devices on the transmission path and the involved broadcast domains must support jumbo frames and use the same MTU. Packet fragmentation and reassembly due to inconsistent MTU settings on the transmission path reduces the network throughput.

Different connection types have certain MTU limitations:

- Ethernet: the MTU is limited to 9000 bytes.
- IP over InfiniBand (IPoIB) in datagram mode: The MTU is limited to 4 bytes less than the InfiniBand MTU.
- In-memory networking commonly supports larger MTUs. For details, see the respective documentation.

31.6.4. Impact of the CPU speed on UDP traffic throughput

In bulk transfers, the UDP protocol is much less efficient than TCP, mainly due to the missing packet aggregation in UDP. By default, the Generic Receive Offload (GRO) and Transmit Segmentation Offload (TSO) features are not enabled. Consequently, the CPU frequency can limit the UDP throughput for bulk transfer on high speed links.

For example, on a tuned host with a high Maximum Transmission Unit (MTU) and large socket buffers, a 3 GHz CPU can process the traffic of a 10 GBit NIC that sends or receives UDP traffic at full speed. However, you can expect about 1-2 Gbps speed loss for every 100 MHz CPU speed under 3 GHz when you transmit UDP traffic. Also, if a CPU speed of 3 GHz can closely achieve 10 Gbps, the same CPU restricts UDP traffic on a 40 GBit NIC to roughly 20-25 Gbps.

31.6.5. Increasing the system-wide UDP socket buffers

Socket buffers temporarily store data that the kernel has received or should send:

- The read socket buffer holds packets that the kernel has received but which the application has not read yet.
- The write socket buffer holds packets that an application has written to the buffer but which the kernel has not passed to the IP stack and network driver yet.

If a UDP packet is too large and exceeds the buffer size or packets are sent or received at a too fast rate, the kernel drops any new incoming UDP packet until the data is removed from the buffer. In this case, increasing the socket buffers can prevent packet loss.



IMPORTANT

Setting too large buffer sizes wastes memory. Each socket can be set to the size that the application requests, and the kernel doubles this value. For example, if an application requests a 256 KiB socket buffer size and opens 1 million sockets, the system requires 512 GB RAM (512 KiB x 1 million) only for the potential socket buffer space.

Prerequisites

- You encountered a significant rate of dropped UDP packets.

Procedure

1. Create the **/etc/sysctl.d/10-udp-socket-buffers.conf** file and either set the maximum read or write buffer size, or both, based on your requirements:

```
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
```

Specify the values in bytes. The values in this example set the maximum size of buffers to 16 MiB. The default values of both parameters are **212992** bytes (208 KiB).

2. Load the settings from the **/etc/sysctl.d/10-udp-socket-buffers.conf** file:

```
# sysctl -p /etc/sysctl.d/10-udp-socket-buffers.conf
```

3. Configure your applications to use the larger socket buffer sizes.
The **net.core.rmem_max** and **net.core.wmem_max** parameters define the maximum buffer size that the **setsockopt()** function in an application can request. Note that, if you configure your application to not use the **setsockopt()** function, the kernel uses the values from the **rmem_default** and **wmem_default** parameters.

For further details, see the documentation of the programming language of your application. If you are not the developer of the application, contact the developer.

4. Restart the applications to use the new UDP buffer sizes.

Verification

- Monitor the packet drop statistics using the same method as you used when you encountered the packet drops.

If packet drops still occur but at a lower rate, increase the buffer sizes further.

Additional resources

- [What are the implications of changing socket buffer sizes?](#) solution
- **udp(7)** man page
- **socket(7)** man page

31.7. IDENTIFYING APPLICATION READ SOCKET BUFFER BOTTLENECKS

If TCP applications do not clear the read socket buffers frequently enough, performance can suffer and packets can be lost. Red Hat Enterprise Linux provides different utilities to identify such problems.

31.7.1. Identifying receive buffer collapsing and pruning

When the data in the receive queue exceeds the receive buffer size, the TCP stack tries to free some space by removing unnecessary metadata from the socket buffer. This step is known as collapsing.

If collapsing fails to free sufficient space for additional traffic, the kernel prunes new data that arrives. This means that the kernel removes the data from the memory and the packet is lost.

To avoid collapsing and pruning operations, monitor whether TCP buffer collapsing and pruning happens on your server and, in this case, tune the TCP buffers.

Procedure

1. Use the **nstat** utility to query the **TcpExtTCPRcvCollapsed** and **TcpExtRcvPruned** counters:

```
# nstat -az TcpExtTCPRcvCollapsed TcpExtRcvPruned
#kernel
TcpExtRcvPruned      0      0.0
TcpExtTCPRcvCollapsed 612859 0.0
```

2. Wait some time and re-run the **nstat** command:

```
# nstat -az TcpExtTCPRcvCollapsed TcpExtRcvPruned
#kernel
TcpExtRcvPruned      0      0.0
TcpExtTCPRcvCollapsed 620358 0.0
```

3. If the values of the counters have increased compared to the first run, tuning is required:
 - If the application uses the **setsockopt(SO_RCVBUF)** call, consider removing it. With this call, the application only uses the receive buffer size specified in the call and turns off the socket's ability to auto-tune its size.
 - If the application does not use the **setsockopt(SO_RCVBUF)** call, tune the default and maximum values of the TCP read socket buffer.
4. Display the receive backlog queue (**Recv-Q**):

```
# ss -nti
State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
ESTAB 0      0      192.0.2.1:443      192.0.2.125:41574
      :7,7 ... lastrcv:543 ...
ESTAB 78     0      192.0.2.1:443      192.0.2.56:42612
      :7,7 ... lastrcv:658 ...
ESTAB 88     0      192.0.2.1:443      192.0.2.97:40313
      :7,7 ... lastrcv:5764 ...
...
```

5. Run the **ss -nt** command multiple times with a few seconds waiting time between each run. If the output lists only one case of a high value in the **Recv-Q** column, the application was between two receive operations. However, if the values in **Recv-Q** stays constant while **lastrcv** continually grows, or **Recv-Q** continually increases over time, one of the following problems can be the cause:

- The application does not check its socket buffers often enough. Contact the application vendor for details about how you can solve this problem.
- The application does not get enough CPU time. To further debug this problem:
 - i. Display on which CPU cores the application runs:

```
# ps -eo pid,tid,psr,pcpu,stat,wchan:20,comm
PID    TID PSR %CPU STAT WCHAN      COMMAND
...
44594  44594  5  0.0 Ss  do_select      httpd
44595  44595  3  0.0 S   skb_wait_for_more_pa httpd
44596  44596  5  0.0 SI  pipe_read      httpd
44597  44597  5  0.0 SI  pipe_read      httpd
44602  44602  5  0.0 SI  pipe_read      httpd
...
```

The **PSR** column displays the CPU cores the process is currently assigned to.

- ii. Identify other processes running on the same cores and consider assigning them to other cores.

Additional resources

- [Increasing the system-wide TCP socket buffers](#)

31.8. TUNING APPLICATIONS WITH A LARGE NUMBER OF INCOMING REQUESTS

If you run an application that handles a large number of incoming requests, such as web servers, it can be necessary to tune Red Hat Enterprise Linux to optimize the performance.

31.8.1. Tuning the TCP listen backlog to process a high number of TCP connection attempts

When an application opens a TCP socket in **LISTEN** state, the kernel limits the number of accepted client connections this socket can handle. If clients try to establish more connections than the application can process, the new connections get lost or the kernel sends SYN cookies to the client.

If the system is under normal workload and too many connections from legitimate clients cause the kernel to send SYN cookies, tune Red Hat Enterprise Linux (RHEL) to avoid them.

Prerequisites

- RHEL logs **possible SYN flooding on port <ip_address>:<port_number>** error messages in the Systemd journal.
- The high number of connection attempts are from valid sources and not caused by an attack.

Procedure

1. To verify whether tuning is required, display the statistics for the affected port:

```
# ss -ntl '( sport = :443)'
State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
LISTEN 650 500 192.0.2.1:443 0.0.0.0:*
```

If the current number of connections in the backlog (**Recv-Q**) is larger than the socket backlog (**Send-Q**), the listen backlog is still not large enough and tuning is required.

2. Optional: Display the current TCP listen backlog limit:

```
# sysctl net.core.somaxconn
net.core.somaxconn = 4096
```

3. Create the `/etc/sysctl.d/10-socket-backlog-limit.conf` file, and set a larger listen backlog limit:

```
net.core.somaxconn = 8192
```

Note that applications can request a larger listen backlog than specified in the **net.core.somaxconn** kernel parameter but the kernel limits the application to the number you set in this parameter.

4. Load the setting from the `/etc/sysctl.d/10-socket-backlog-limit.conf` file:

```
# sysctl -p /etc/sysctl.d/10-socket-backlog-limit.conf
```

5. Reconfigure the application to use the new listen backlog limit:

- If the application provides a config option for the limit, update it. For example, the Apache HTTP Server provides the **ListenBacklog** configuration option to set the listen backlog limit for this service.
- If you cannot configure the limit, recompile the application.

6. Restart the application.

Verification

1. Monitor the Systemd journal for further occurrences of **possible SYN flooding on port <port_number>** error messages.
2. Monitor the current number of connections in the backlog and compare it with the socket backlog:

```
# ss -ntl '( sport = :443)'
```

State	Recv-Q	Send-Q	Local Address:Port	Peer Address:Port	Process
LISTEN	0	500	192.0.2.1:443	0.0.0.0:*	

If the current number of connections in the backlog (**Recv-Q**) is larger than the socket backlog (**Send-Q**), the listen backlog is not large enough and further tuning is required.

Additional resources

- [kernel: Possible SYN flooding on port #. Sending cookies](#) solution
- [Listening TCP server ignores SYN or ACK for new connection handshake](#) solution
- [listen\(2\)](#) man page

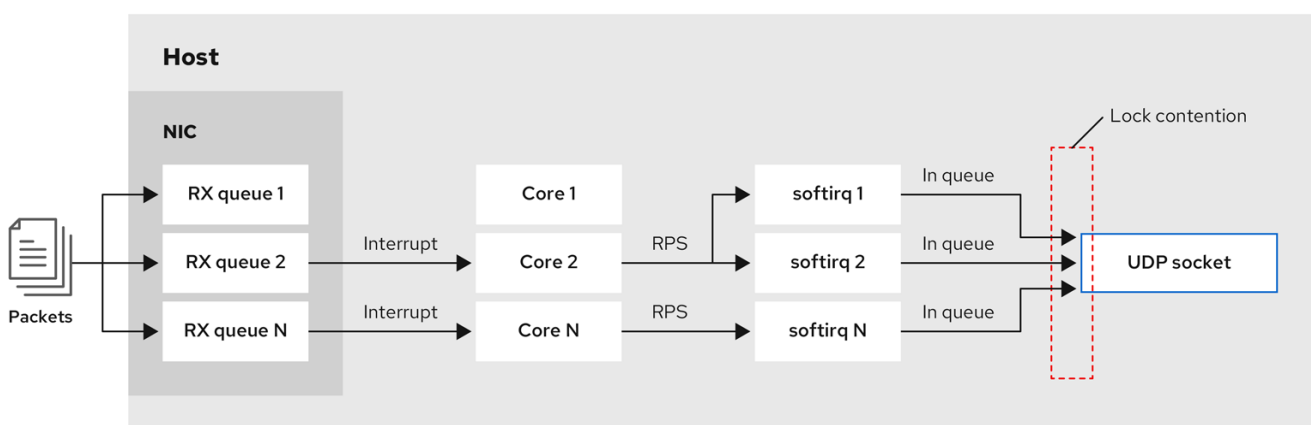
31.9. AVOIDING LISTEN QUEUE LOCK CONTENTION

Queue lock contention can cause packet drops and higher CPU usage and, consequently, a higher latency. You can avoid queue lock contention on the receive (RX) and transmit (TX) queue by tuning your application and using transmit packet steering.

31.9.1. Avoiding RX queue lock contention: The `SO_REUSEPORT` and `SO_REUSEPORT_BPF` socket options

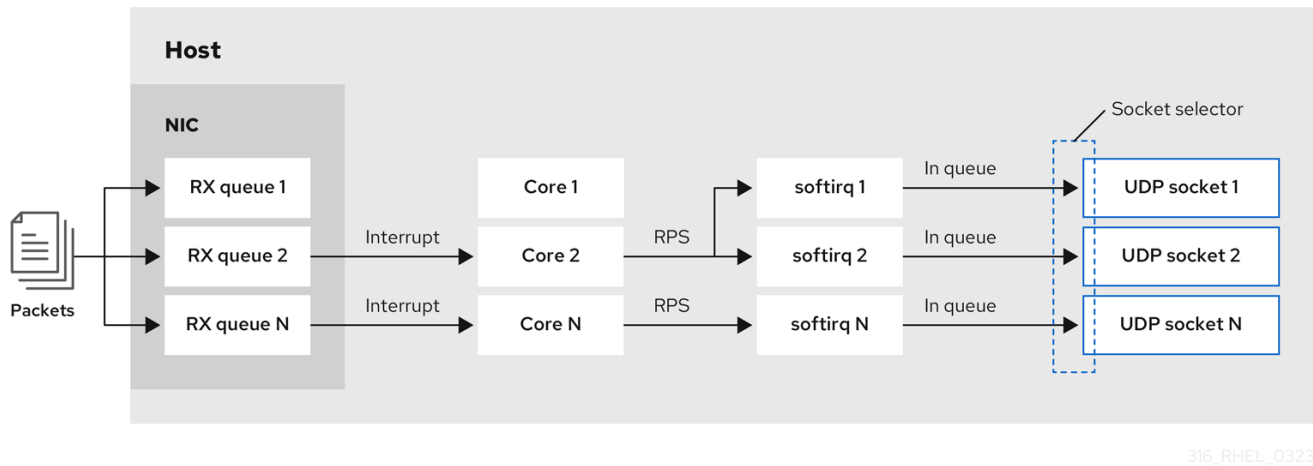
On a multi-core system, you can improve the performance of multi-threaded network server applications if the application opens the port by using the `SO_REUSEPORT` or `SO_REUSEPORT_BPF` socket option. If the application does not use one of these socket options, all threads are forced to share a single socket to receive the incoming traffic. Using a single socket causes:

- Significant contention on the receive buffer, which can cause packet drops and higher CPU usage.
- A significant increase of CPU usage
- Possibly packet drops



316_RHEL_0323

With the `SO_REUSEPORT` or `SO_REUSEPORT_BPF` socket option, multiple sockets on one host can bind to the same port:



316_RHEL_0323

Red Hat Enterprise Linux provides a code example of how to use the **SO_REUSEPORT** socket options in the kernel sources. To access the code example:

1. Enable the **rhel-9-for-x86_64-baseos-debug-rpms** repository:

```
# subscription-manager repos --enable rhel-9-for-x86_64-baseos-debug-rpms
```

2. Install the **kernel-debuginfo-common-x86_64** package:

```
# dnf install kernel-debuginfo-common-x86_64
```

3. The code example is now available in the **/usr/src/debug/kernel-*<version>*/linux-*<version>*/tools/testing/selftests/net/reuseport_bpf_cpu.c** file.

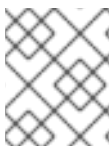
Additional resources

- **socket(7)** man page
- **/usr/src/debug/kernel-*<version>*/linux-*<version>*/tools/testing/selftests/net/reuseport_bpf_cpu.c**

31.9.2. Avoiding TX queue lock contention: Transmit packet steering

In hosts with a network interface controller (NIC) that supports multiple queues, transmit packet steering (XPS) distributes the processing of outgoing network packets among several queues. This enables multiple CPUs to process the outgoing network traffic and to avoid transmit queue lock contention and, consequently, packet drops.

Certain drivers, such as **ixgbe**, **i40e**, and **mlx5** automatically configure XPS. To identify if the driver supports this capability, consult the documentation of your NIC driver. Consult your NIC driver's documentation to identify if the driver supports this capability. If the driver does not support XPS auto-tuning, you can manually assign CPU cores to the transmit queues.



NOTE

Red Hat Enterprise Linux does not provide an option to permanently assign transmit queues to CPU cores. Use the commands in a script and run it when the system boots.

Prerequisites

- The NIC supports multiple queues.
- The **numactl** package is installed.

Procedure

1. Display the count of available queues:

```
# ethtool -l enp1s0
Channel parameters for enp1s0:
Pre-set maximums:
RX: 0
TX: 0
Other: 0
Combined: 4
Current hardware settings:
RX: 0
TX: 0
Other: 0
Combined: 1
```

The **Pre-set maximums** section shows the total number of queues and **Current hardware settings** the number of queues that are currently assigned to the receive, transmit, other, or combined queues.

2. Optional: If you require queues on specific channels, assign them accordingly. For example, to assign the 4 queues to the **Combined** channel, enter:

```
# ethtool -L enp1s0 combined 4
```

3. Display to which Non-Uniform Memory Access (NUMA) node the NIC is assigned:

```
# cat /sys/class/net/enp1s0/device/numa_node
0
```

If the file is not found or the command returns **-1**, the host is not a NUMA system.

4. If the host is a NUMA system, display which CPUs are assigned to which NUMA node:

```
# lscpu | grep NUMA
NUMA node(s): 2
NUMA node0 CPU(s): 0-3
NUMA node1 CPU(s): 4-7
```

5. In the example above, the NIC has 4 queues and the NIC is assigned to NUMA node 0. This node uses the CPU cores 0-3. Consequently, map each transmit queue to one of the CPU cores from 0-3:

```
# echo 1 > /sys/class/net/enp1s0/queues/tx-0/xps_cpus
# echo 2 > /sys/class/net/enp1s0/queues/tx-1/xps_cpus
# echo 4 > /sys/class/net/enp1s0/queues/tx-2/xps_cpus
# echo 8 > /sys/class/net/enp1s0/queues/tx-3/xps_cpus
```

If the number of CPU cores and transmit (TX) queues is the same, use a 1 to 1 mapping to avoid

any kind of contention on the TX queue. Otherwise, if you map multiple CPUs on the same TX queue, transmit operations on different CPUs will cause TX queue lock contention and negatively impacts the transmit throughput.

Note that you must pass the bitmap, containing the CPU's core numbers, to the queues. Use the following command to calculate the bitmap:

```
# printf %x $((1 << <core_number> ))
```

Verification

1. Identify the process IDs (PIDs) of services that send traffic:

```
# pidof <process_name>
12345 98765
```

2. Pin the PIDs to cores that use XPS:

```
# numactl -C 0-3 12345 98765
```

3. Monitor the **requeues** counter while the process send traffic:

```
# tc -s qdisc
qdisc fq_codel 0: dev enp10s0u1 root refcnt 2 limit 10240p flows 1024 quantum 1514 target
5ms interval 100ms memory_limit 32Mb ecn drop_batch 64
Sent 125728849 bytes 1067587 pkt (dropped 0, overlimits 0 requeues 30)
backlog 0b 0p requeues 30
...
```

If the **requeues** counter no longer increases at a significant rate, TX queue lock contention no longer happens.

Additional resources

- [/usr/share/doc/kernel-doc-_**<version>**/Documentation/networking/scaling.rst](/usr/share/doc/kernel-doc-_<version>/Documentation/networking/scaling.rst)

31.9.3. Disabling the Generic Receive Offload feature on servers with high UDP traffic

Applications that use high-speed UDP bulk transfer should enable and use UDP Generic Receive Offload (GRO) on the UDP socket. However, you can disable GRO to increase the throughput if the following conditions apply:

- The application does not support GRO and the feature cannot be added.
- TCP throughput is not relevant.

**WARNING**

Disabling GRO significantly reduces the receive throughput of TCP traffic. Therefore, do not disable GRO on hosts where TCP performance is relevant.

Prerequisites

- The host mainly processes UDP traffic.
- The application does not use GRO.
- The host does not use UDP tunnel protocols, such as VXLAN.
- The host does not run virtual machines (VMs) or containers.

Procedure

1. Optional: Display the NetworkManager connection profiles:

```
# nmcli connection show
NAME      UUID                                  TYPE      DEVICE
example   f2f33f29-bb5c-3a07-9069-be72eaec3ecf ethernet enp1s0
```

2. Disable GRO support in the connection profile:

```
# nmcli connection modify example ethtool.feature-gro off
```

3. Reactivate the connection profile:

```
# nmcli connection up example
```

Verification

1. Verify that GRO is disabled:

```
# ethtool -k enp1s0 | grep generic-receive-offload
generic-receive-offload: off
```

2. Monitor the throughput on the server. Re-enable GRO in the NetworkManager profile if the setting has negative side effects to other applications on the host.

31.10. TUNING THE DEVICE DRIVER AND NIC

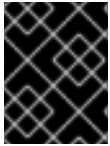
In RHEL, kernel modules provide drivers for network interface controllers (NICs). These modules support parameters to tune and optimize the device driver and the NIC. For example, if the driver supports delaying the generation of receive interrupts, you can reduce the value of the corresponding parameter to avoid running out of receive descriptors.

**NOTE**

Not all modules support custom parameters, and the features depend on the hardware, as well as the driver and firmware version.

31.10.1. Configuring custom NIC driver parameters

Many kernel modules support setting parameters to tune the driver and the network interface controller (NIC). You can customize the settings according to the hardware and the driver.

**IMPORTANT**

If you set parameters on a kernel module, RHEL applies these settings to all devices that use this driver.

Prerequisites

- A NIC is installed in the host.
- The kernel module that provides the driver for the NIC supports the required tuning feature.
- You are logged in locally or using a network interface that is different from the one that uses the driver for which you want to change the parameters.

Procedure

1. Identify the driver:

```
# ethtool -i enp0s31f6
driver: e1000e
version: ...
firmware-version: ...
...
```

Note that certain features can require a specific driver and firmware version.

2. Display the available parameters of the kernel module:

```
# modinfo -p e1000e
...
SmartPowerDownEnable:Enable PHY smart power down (array of int)
parm:RxIntDelay:Receive Interrupt Delay (array of int)
```

For further details on the parameters, see the kernel module's documentation. For modules in RHEL, see the documentation in the `/usr/share/doc/kernel-doc-<version>/Documentation/networking/device_drivers/` directory that is provided by the `kernel-doc` package.

3. Create the `/etc/modprobe.d/nic-parameters.conf` file and specify the parameters for the module:

```
options <module_name> <parameter1>=<value> <parameter2>=<value>
```

For example, to enable the port power saving mechanism and set the generation of receive interrupts to 4 units, enter:

```
options e1000e SmartPowerDownEnable=1 RxIntDelay=4
```

4. Unload the module:

```
# modprobe -r e1000e
```



WARNING

Unloading a module that an active network interface uses, immediately terminates the connection and you can lock yourself out of the server.

5. Load the module:

```
# modprobe e1000e
```

6. Reactivate the network connections:

```
# nmcli connection up <profile_name>
```

Verification

1. Display the kernel messages:

```
# dmesg
...
[35309.225765] e1000e 0000:00:1f.6: Transmit Interrupt Delay set to 16
[35309.225769] e1000e 0000:00:1f.6: PHY Smart Power Down Enabled
...
```

Note that not all modules log parameter settings to the kernel ring buffer.

2. Certain kernel modules create files for each module parameter in the `/sys/module/<driver>/parameters/` directory. Each of these files contain the current value of this parameter. You can display these files to verify a setting:

```
# cat /sys/module/<driver_name>/parameters/<parameter_name>
```

31.11. CONFIGURING NETWORK ADAPTER OFFLOAD SETTINGS

To reduce CPU load, certain network adapters use offloading features which move the network processing load to the network interface controller (NIC). For example, with Encapsulating Security Payload (ESP) offload, the NIC performs ESP operations to accelerate IPsec connections and reduce CPU load.

By default, most offloading features in Red Hat Enterprise Linux are enabled. Only disable them in the following cases:

- Temporarily disable offload features for troubleshooting purposes.
- Permanently disable offload features when a specific feature negatively impacts your host.

If a performance-related offload feature is not enabled by default in a network driver, you can enable it manually.

31.11.1. Temporarily setting an offload feature

If you expect that an offload feature causes problems or reduces the performance of your host, you can attempt to narrow down the cause by temporarily enabling or disabling it, depending on its current state.

If you temporarily enable or disable an offload feature, it returns to its previous value on the next reboot.

Prerequisites

- The network card supports offload features.

Procedure

1. Display the interface's available offload features and their current state:

```
# ethtool -k enp1s0
...
esp-hw-offload: on
ntuple-filters: off
rx-vlan-filter: off [fixed]
...
```

The output depends on the capabilities of the hardware and its driver. Note that you cannot change the state of features that are flagged with **[fixed]**.

2. Temporarily disable an offload feature:

```
# ethtool -K <interface> <feature> [on/off]
```

- For example, to temporarily disable IPsec Encapsulating Security Payload (ESP) offload on the **enp10s0u1** interface, enter:

```
# ethtool -K enp10s0u1 esp-hw-offload off
```

- For example, to temporarily enable accelerated Receive Flow Steering (aRFS) filtering on the **enp10s0u1** interface, enter:

```
# ethtool -K enp10s0u1 ntuple-filters on
```

Verification

1. Display the states of the offload features:

```
# ethtool -k enp1s0
...
esp-hw-offload: off
```

```
ntuple-filters: on
```

```
...
```

2. Test whether the problem you encountered before changing the offload feature still exists.
 - If the problem no longer exists after changing a specific offload feature:
 - i. Contact [Red Hat Support](#) and report the problem.
 - ii. Consider [permanently setting the offload feature](#) until a fix is available.
 - If the problem still exists after disabling a specific offload feature:
 - i. Reset the setting to its previous state by using the **ethtool -K <interface> <feature> [on/off]** command.
 - ii. Enable or disable a different offload feature to narrow down the problem.

Additional resources

- **ethtool(8)** man page

31.11.2. Permanently setting an offload feature

If you have identified a specific offload feature that limits the performance on your host, you can permanently enable or disable it, depending on its current state.

If you permanently enable or disable an offload feature, NetworkManager ensures that the feature still has this state after a reboot.

Prerequisites

- You identified a specific offload feature to limit the performance on your host.

Procedure

1. Identify the connection profile that uses the network interface on which you want to change the state of the offload feature:

```
# nmcli connection show
NAME    UUID                                  TYPE    DEVICE
Example a5eb6490-cc20-3668-81f8-0314a27f3f75 ethernet enp1ss0
...
```

2. Permanently change the state of the offload feature:

```
# nmcli connection modify <connection_name> <feature> [on/off]
```

- For example, to permanently disable IPsec Encapsulating Security Payload (ESP) offload in the **Example** connection profile, enter:

```
# nmcli connection modify Example ethtool.feature-esp-hw-offload off
```

- For example, to permanently enable accelerated Receive Flow Steering (aRFS) filtering in the **Example** connection profile, enter:

```
# nmcli connection modify Example ethtool.feature-ntuple on
```

3. Reactivate the connection profile:

```
# nmcli connection up Example
```

Verification

- Display the output states of the offload features:

```
# ethtool -k enp1s0
...
esp-hw-offload: off
ntuple-filters: on
...
```

Additional resources

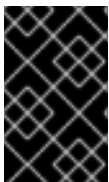
- [Offload features supported by NetworkManager](#)

31.12. TUNING INTERRUPT COALESCENCE SETTINGS

Interrupt coalescence is a mechanism for reducing the number of interrupts generated by a network card. Generally, fewer interrupts can enhance the latency and overall performance of your network.

Tuning the interrupt coalescence settings involves adjusting the parameters that control:

- The number of packets that are combined into a single interrupt.
- The delay before generating an interrupt.



IMPORTANT

The optimal coalescence settings depend on the specific network conditions and hardware in use. Therefore, it might take several attempts to find the settings that work best for your environment and needs.

31.12.1. Optimizing RHEL for latency or throughput-sensitive services

The goal of coalesce tuning is to minimize the number of interrupts required for a given workload. In high-throughput situations, the goal is to have as few interrupts as possible while maintaining a high data rate. In low-latency situations, more interrupts can be used to handle traffic quickly.

You can adjust the settings on your network card to increase or decrease the number of packets that are combined into a single interrupt. As a result, you can achieve improved throughput or latency for your traffic.

Procedure

1. Identify the network interface that is experiencing the bottleneck:

```
# ethtool -S enp1s0
NIC statistics:
```

```

rx_packets: 1234
tx_packets: 5678
rx_bytes: 12345678
tx_bytes: 87654321
rx_errors: 0
tx_errors: 0
rx_missed: 0
tx_dropped: 0
coalesced_pkts: 0
coalesced_events: 0
coalesced_aborts: 0

```

Identify the packet counters containing "drop", "discard", or "error" in their name. These particular statistics measure the actual packet loss at the network interface card (NIC) packet buffer, which can be caused by NIC coalescence.

2. Monitor values of packet counters you identified in the previous step. Compare them to the expected values for your network to determine whether any particular interface experiences a bottleneck. Some common signs of a network bottleneck include, but are not limited to:

- Many errors on a network interface
- High packet loss
- Heavy usage of the network interface



NOTE

Other important factors are for example CPU usage, memory usage, and disk I/O when identifying a network bottleneck.

3. View the current coalescence settings:

```

# ethtool enp1s0
Settings for enp1s0:
    Supported ports: [ TP ]
    Supported link modes:  10baseT/Half 10baseT/Full
                        100baseT/Half 100baseT/Full
                        1000baseT/Full
    Supported pause frame use: No
    Supports auto-negotiation: Yes
    Advertised link modes: 10baseT/Half 10baseT/Full
                        100baseT/Half 100baseT/Full
                        1000baseT/Full
    Advertised pause frame use: No
    Advertised auto-negotiation: Yes
    Speed: 1000Mb/s
    Duplex: Full
    Port: Twisted Pair
    PHYAD: 0
    Transceiver: internal
    Auto-negotiation: on
    MDI-X: Unknown
    Supports Wake-on: g

```

```
Wake-on: g
Current message level: 0x00000033 (51)
                    drv probe link
Link detected: yes
```

In this output, monitor the **Speed** and **Duplex** fields. These fields display information about the network interface operation and whether it is running at its expected values.

4. Check the current interrupt coalescence settings:

```
# ethtool -c enp1s0
Coalesce parameters for enp1s0:
  Adaptive RX: off
  Adaptive TX: off
  RX usecs: 100
  RX frames: 8
  RX usecs irq: 100
  RX frames irq: 8
  TX usecs: 100
  TX frames: 8
  TX usecs irq: 100
  TX frames irq: 8
```

- The **usecs** values refer to the number of microseconds that the receiver or transmitter waits before generating an interrupt.
- The **frames** values refer to the number of frames that the receiver or transmitter waits before generating an interrupt.
- The **irq** values are used to configure the interrupt moderation when the network interface is already handling an interrupt.



NOTE

Not all network interface cards support reporting and changing all values from the example output.

- The **Adaptive RX/TX** value represents the adaptive interrupt coalescence mechanism, which adjusts the interrupt coalescence settings dynamically. Based on the packet conditions, the NIC driver auto-calculates coalesce values when **Adaptive RX/TX** are enabled (the algorithm differs for every NIC driver).

5. Modify the coalescence settings as needed. For example:

- While **ethtool.coalesce-adaptive-rx** is disabled, configure **ethtool.coalesce-rx-usecs** to set the delay before generating an interrupt to 100 microseconds for the RX packets:

```
# nmcli connection modify enp1s0 ethtool.coalesce-rx-usecs 100
```

- Enable **ethtool.coalesce-adaptive-rx** while **ethtool.coalesce-rx-usecs** is set to its default value:

```
# nmcli connection modify enp1s0 ethtool.coalesce-adaptive-rx on
```

Red Hat recommends that modifying the Adaptive-RX setting as follows:

- Users concerned with low latency (sub-50us) should not enable **Adaptive-RX**.
- Users concerned with throughput can probably enable **Adaptive-RX** with no harm. If they do not want to use the adaptive interrupt coalescence mechanism, they can try setting large values like 100us, or 250us to **ethtool.coalesce-rx-usecs**.
- Users unsure about their needs should not modify this setting until an issue occurs.

6. Re-activate the connection:

```
# nmcli connection up enp1s0
```

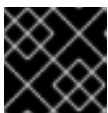
Verification steps

- Monitor the network performance and check for dropped packets:

```
# ethtool -S enp1s0
NIC statistics:
  rx_packets: 1234
  tx_packets: 5678
  rx_bytes: 12345678
  tx_bytes: 87654321
  rx_errors: 0
  tx_errors: 0
  rx_missed: 0
  tx_dropped: 0
  coalesced_pkts: 12
  coalesced_events: 34
  coalesced_aborts: 56
...
```

The value of the **rx_errors**, **rx_dropped**, **tx_errors**, and **tx_dropped** fields should be 0 or close to it (up to few hundreds, depending on the network traffic and system resources). A high value in these fields indicates a network problem. Your counters can have different names. Closely monitor packet counters containing "drop", "discard", or "error" in their name.

The value of the **rx_packets**, **tx_packets**, **rx_bytes**, and **tx_bytes** should increase over time. If the values do not increase, there might be a network problem. The packet counters can have different names, depending on your NIC driver.



IMPORTANT

The **ethtool** command output can vary depending on the NIC and driver in use.

Users with focus on extremely low latency can use application-level metrics or the kernel packet time-stamping API for their monitoring purposes.

Additional resources

- [Initial investigation for any performance issue](#)
- [What are the kernel parameters available for network tuning?](#)
- [How to make NIC ethtool settings persistent \(apply automatically at boot\)](#)

- [Timestamping](#)

31.13. BENEFITS OF TCP TIMESTAMPS

TCP Timestamps are optional information in the TCP header and an extension of the TCP protocol. By default, TCP Timestamps are enabled in Red Hat Enterprise Linux, and the kernel uses TCP Timestamps to better estimate the round trip time (RTT) in TCP connections. This results in more accurate TCP window and buffer calculations.

Additionally, TCP Timestamps provide an alternative method to determine the age and order of a segment, and protect against wrapped sequence numbers. TCP packet headers record the sequence number in a 32-bit field. On a 10 Gbps connection, the value of this field can wrap after 1.7 seconds. Without TCP Timestamps, the receiver could not determine whether a segment with a wrapped sequence number is a new segment or an old duplicate. With TCP Timestamps, however, the receiver can make the correct choice to receive or discard the segment. Therefore, enabling TCP Timestamps on systems with fast network interfaces is essential.

The **net.ipv4.tcp_timestamps** kernel parameter can have one of the following values:

- **0**: TCP Timestamps are disabled.
- **1**: TCP Timestamps are enabled (default).
- **2**: TCP Timestamps are enabled but without random offsets.



IMPORTANT

Without random offsets for each connection, it is possible to approximately determine the host's uptime and fingerprint and use this information in attacks.

By default, TCP Timestamps are enabled in Red Hat Enterprise Linux and use random offsets for each connection instead of only storing the current time:

```
# sysctl net.ipv4.tcp_timestamps
net.ipv4.tcp_timestamps = 1
```

If the **net.ipv4.tcp_timestamps** parameter has a different value than the default (**1**), revert the setting in the same way as you set it.

Additional resources

- [RFC 1323: TCP Extensions for High Performance](#)

31.14. FLOW CONTROL FOR ETHERNET NETWORKS

On an Ethernet link, continuous data transmission between a network interface and a switch port can lead to full buffer capacity. Full buffer capacity results in network congestion. In this case, when the sender transmits data at a higher rate than the processing capacity of the receiver, packet loss can occur due to the lower data processing capacity of a network interface on the other end of the link which is a switch port.

The flow control mechanism manages data transmission across the Ethernet link where each sender and receiver has different sending and receiving capacities. To avoid packet loss, the Ethernet flow control mechanism temporarily suspends the packet transmission to manage a higher transmission rate from a

switch port. Note that routers do not forward pause frames beyond a switch port.

When receive (RX) buffers become full, a receiver sends pause frames to the transmitter. The transmitter then stops data transmission for a short sub-second time frame, while continuing to buffer incoming data during this pause period. This duration provides enough time for the receiver to empty its interface buffers and prevent buffer overflow.



NOTE

Either end of the Ethernet link can send pause frames to another end. If the receive buffers of a network interface are full, the network interface will send pause frames to the switch port. Similarly, when the receive buffers of a switch port are full, the switch port sends pause frames to the network interface.

By default, most of the network drivers in Red Hat Enterprise Linux have pause frame support enabled. To display the current settings of a network interface, enter:

```
# ethtool --show-pause enp1s0
Pause parameters for enp1s0:
...
RX:   on
TX:   on
...
```

Verify with your switch vendor to confirm if your switch supports pause frames.

Additional resources

- **ethtool(8)** man page
- [What is network link flow control and how does it work in Red Hat Enterprise Linux?](#)

CHAPTER 32. FACTORS AFFECTING I/O AND FILE SYSTEM PERFORMANCE

The appropriate settings for storage and file system performance are highly dependent on the storage purpose.

I/O and file system performance can be affected by any of the following factors:

- Data write or read patterns
- Sequential or random
- Buffered or Direct IO
- Data alignment with underlying geometry
- Block size
- File system size
- Journal size and location
- Recording access times
- Ensuring data reliability
- Pre-fetching data
- Pre-allocating disk space
- File fragmentation
- Resource contention

32.1. TOOLS FOR MONITORING AND DIAGNOSING I/O AND FILE SYSTEM ISSUES

The following tools are available in Red Hat Enterprise Linux 9 for monitoring system performance and diagnosing performance problems related to I/O, file systems, and their configuration:

- **vmstat** tool reports on processes, memory, paging, block I/O, interrupts, and CPU activity across the entire system. It can help administrators determine whether the I/O subsystem is responsible for any performance issues. If analysis with **vmstat** shows that the I/O subsystem is responsible for reduced performance, administrators can use the **iostat** tool to determine the responsible I/O device.
- **iostat** reports on I/O device load in your system. It is provided by the **sysstat** package.
- **blktrace** provides detailed information about how time is spent in the I/O subsystem. The companion utility **blkparse** reads the raw output from **blktrace** and produces a human readable summary of input and output operations recorded by **blktrace**.
- **btt** analyzes **blktrace** output and displays the amount of time that data spends in each area of the I/O stack, making it easier to spot bottlenecks in the I/O subsystem. This utility is provided as part of the **blktrace** package. Some of the important events tracked by the **blktrace** mechanism and analyzed by **btt** are:

- Queuing of the I/O event (**Q**)
- Dispatch of the I/O to the driver event (**D**)
- Completion of I/O event (**C**)
- **iowatcher** can use the **blktrace** output to graph I/O over time. It focuses on the Logical Block Address (LBA) of disk I/O, throughput in megabytes per second, the number of seeks per second, and I/O operations per second. This can help to identify when you are hitting the operations-per-second limit of a device.
- BPF Compiler Collection (BCC) is a library, which facilitates the creation of the extended Berkeley Packet Filter (**eBPF**) programs. The **eBPF** programs are triggered on events, such as disk I/O, TCP connections, and process creations. The BCC tools are installed in the **/usr/share/bcc/tools/** directory. The following **bcc-tools** helps to analyze performance:
 - **biolatency** summarizes the latency in block device I/O (disk I/O) in histogram. This allows the distribution to be studied, including two modes for device cache hits and for cache misses, and latency outliers.
 - **biosnoop** is a basic block I/O tracing tool for displaying each I/O event along with the issuing process ID, and the I/O latency. Using this tool, you can investigate disk I/O performance issues.
 - **biotop** is used for block i/o operations in the kernel.
 - **filelife** tool traces the **stat()** syscalls.
 - **fileslower** traces slow synchronous file reads and writes.
 - **filetop** displays file reads and writes by process.
 - **ext4slower**, **nfsslower**, and **xfsslower** are tools that show file system operations slower than a certain threshold, which defaults to **10ms**.
For more information, see the [Analyzing system performance with BPF Compiler Collection](#) .
- **bpftool** is a tracing language for **eBPF** used for analyzing performance issues. It also provides trace utilities like BCC for system observation, which is useful for investigating I/O performance issues.
- The following **SystemTap** scripts may be useful in diagnosing storage or file system performance problems:
 - **disktop.stp**: Checks the status of reading or writing disk every 5 seconds and outputs the top ten entries during that period.
 - **iotime.stp**: Prints the amount of time spent on read and write operations, and the number of bytes read and written.
 - **traceio.stp**: Prints the top ten executable based on cumulative I/O traffic observed, every second.
 - **traceio2.stp**: Prints the executable name and process identifier as reads and writes to the specified device occur.
 - **Inodewatch.stp**: Prints the executable name and process identifier each time a read or write occurs to the specified inode on the specified major or minor device.

- **inodewatch2.stp**: Prints the executable name, process identifier, and attributes each time the attributes are changed on the specified inode on the specified major or minor device.

Additional resources

- **vmstat(8)**, **iostat(1)**, **blktrace(8)**, **blkparse(1)**, **btt(1)**, **bpfttrace**, and **iowatcher(1)** man pages
- [Analyzing system performance with BPF Compiler Collection](#)

32.2. AVAILABLE TUNING OPTIONS FOR FORMATTING A FILE SYSTEM

Some file system configuration decisions cannot be changed after the device is formatted.

The following are the options available before formatting a storage device:

Size

Create an appropriately-sized file system for your workload. Smaller file systems require less time and memory for file system checks. However, if a file system is too small, its performance suffers from high fragmentation.

Block size

The block is the unit of work for the file system. The block size determines how much data can be stored in a single block, and therefore the smallest amount of data that is written or read at one time. The default block size is appropriate for most use cases. However, your file system performs better and stores data more efficiently if the block size or the size of multiple blocks is the same as or slightly larger than the amount of data that is typically read or written at one time. A small file still uses an entire block. Files can be spread across multiple blocks, but this can create additional runtime overhead.

Additionally, some file systems are limited to a certain number of blocks, which in turn limits the maximum size of the file system. Block size is specified as part of the file system options when formatting a device with the **mkfs** command. The parameter that specifies the block size varies with the file system.

Geometry

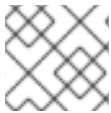
File system geometry is concerned with the distribution of data across a file system. If your system uses striped storage, like RAID, you can improve performance by aligning data and metadata with the underlying storage geometry when you format the device.

Many devices export recommended geometry, which is then set automatically when the devices are formatted with a particular file system. If your device does not export these recommendations, or you want to change the recommended settings, you must specify geometry manually when you format the device with the **mkfs** command.

The parameters that specify file system geometry vary with the file system.

External journals

Journaling file systems document the changes that will be made during a write operation in a journal file prior to the operation being executed. This reduces the likelihood that a storage device will become corrupted in the event of a system crash or power failure, and speeds up the recovery process.

**NOTE**

Red Hat does not recommend using the external journals option.

Metadata-intensive workloads involve very frequent updates to the journal. A larger journal uses more memory, but reduces the frequency of write operations. Additionally, you can improve the seek time of a device with a metadata-intensive workload by placing its journal on dedicated storage that is as fast as, or faster than, the primary storage.

**WARNING**

Ensure that external journals are reliable. Losing an external journal device causes file system corruption. External journals must be created at format time, with journal devices being specified at mount time.

Additional resources

- **mkfs(8)** and **mount(8)** man pages
- [Overview of available file systems](#)

32.3. AVAILABLE TUNING OPTIONS FOR MOUNTING A FILE SYSTEM

The following are the options available to most file systems and can be specified as the device is mounted:

Access Time

Every time a file is read, its metadata is updated with the time at which access occurred (**atime**). This involves additional write I/O. The **relatime** is the default **atime** setting for most file systems. However, if updating this metadata is time consuming, and if accurate access time data is not required, you can mount the file system with the **noatime** mount option. This disables updates to metadata when a file is read. It also enables **nodiratime** behavior, which disables updates to metadata when a directory is read.

**NOTE**

Disabling **atime** updates by using the **noatime mount** option can break applications that rely on them, for example, backup programs.

Read-ahead

Read-ahead behavior speeds up file access by pre-fetching data that is likely to be needed soon and loading it into the page cache, where it can be retrieved more quickly than if it were on disk. The higher the read-ahead value, the further ahead the system pre-fetches data.

Red Hat Enterprise Linux attempts to set an appropriate read-ahead value based on what it detects about your file system. However, accurate detection is not always possible. For example, if a storage array presents itself to the system as a single LUN, the system detects the single LUN, and does not set the appropriate read-ahead value for an array.

Workloads that involve heavy streaming of sequential I/O often benefit from high read-ahead values. The storage-related tuned profiles provided with Red Hat Enterprise Linux raise the read-ahead value, as does using LVM striping, but these adjustments are not always sufficient for all workloads.

Additional resources

- **mount(8)**, **xfs(5)**, and **ext4(5)** man pages

32.4. TYPES OF DISCARDING UNUSED BLOCKS

Regularly discarding blocks that are not in use by the file system is a recommended practice for both solid-state disks and thinly-provisioned storage.

The following are the two methods of discarding unused blocks:

Batch discard

This type of discard is part of the **fstrim** command. It discards all unused blocks in a file system that match criteria specified by the administrator. Red Hat Enterprise Linux 9 supports batch discard on XFS and ext4 formatted devices that support physical discard operations.

Online discard

This type of discard operation is configured at mount time with the discard option, and runs in real time without user intervention. However, it only discards blocks that are transitioning from used to free. Red Hat Enterprise Linux 9 supports online discard on XFS and ext4 formatted devices.

Red Hat recommends batch discard, except where online discard is required to maintain performance, or where batch discard is not feasible for the system's workload.

Pre-allocation marks disk space as being allocated to a file without writing any data into that space. This can be useful in limiting data fragmentation and poor read performance. Red Hat Enterprise Linux 9 supports pre-allocating space on XFS, ext4, and GFS2 file systems. Applications can also benefit from pre-allocating space by using the **fallocate(2) glibc** call.

Additional resources

- **mount(8)** and **fallocate(2)** man pages

32.5. SOLID-STATE DISKS TUNING CONSIDERATIONS

Solid-state disks (SSD) use NAND flash chips rather than rotating magnetic platters to store persistent data. SSD provides a constant access time for data across their full Logical Block Address range, and does not incur measurable seek costs like their rotating counterparts. They are more expensive per gigabyte of storage space and have a lesser storage density, but they also have lower latency and greater throughput than HDDs.

Performance generally degrades as the used blocks on an SSD approach the capacity of the disk. The degree of degradation varies by vendor, but all devices experience degradation in this circumstance. Enabling discard behavior can help to alleviate this degradation. For more information, see [Types of discarding unused blocks](#).

The default I/O scheduler and virtual memory options are suitable for use with SSDs. Consider the following factors when configuring settings that can affect SSD performance:

I/O Scheduler

Any I/O scheduler is expected to perform well with most SSDs. However, as with any other storage type, Red Hat recommends benchmarking to determine the optimal configuration for a given workload. When using SSDs, Red Hat advises changing the I/O scheduler only for benchmarking particular workloads. For instructions on how to switch between I/O schedulers, see the </usr/share/doc/kernel-version/Documentation/block/switching-sched.txt> file.

For single queue HBA, the default I/O scheduler is **deadline**. For multiple queue HBA, the default I/O scheduler is **none**. For information about how to set the I/O scheduler, see [Setting the disk scheduler](#).

Virtual Memory

Like the I/O scheduler, virtual memory (VM) subsystem requires no special tuning. Given the fast nature of I/O on SSD, try turning down the **vm_dirty_background_ratio** and **vm_dirty_ratio** settings, as increased write-out activity does not usually have a negative impact on the latency of other operations on the disk. However, this tuning can generate more overall I/O, and is therefore not generally recommended without workload-specific testing.

Swap

An SSD can also be used as a swap device, and is likely to produce good page-out and page-in performance.

32.6. GENERIC BLOCK DEVICE TUNING PARAMETERS

The generic tuning parameters listed here are available in the **/sys/block/sdX/queue/** directory.

The following listed tuning parameters are separate from I/O scheduler tuning, and are applicable to all I/O schedulers:

add_random

Some I/O events contribute to the entropy pool for the **/dev/random**. This parameter can be set to **0** if the overhead of these contributions become measurable.

iostats

By default, **iostats** is enabled and the default value is **1**. Setting **iostats** value to **0** disables the gathering of I/O statistics for the device, which removes a small amount of overhead with the I/O path. Setting **iostats** to **0** might slightly improve performance for very high performance devices, such as certain NVMe solid-state storage devices. It is recommended to leave **iostats** enabled unless otherwise specified for the given storage model by the vendor.

If you disable **iostats**, the I/O statistics for the device are no longer present within the **/proc/diskstats** file. The content of **/sys/diskstats** file is the source of I/O information for monitoring I/O tools, such as **sar** or **iotop**. Therefore, if you disable the **iostats** parameter for a device, the device is no longer present in the output of I/O monitoring tools.

max_sectors_kb

Specifies the maximum size of an I/O request in kilobytes. The default value is **512** KB. The minimum value for this parameter is determined by the logical block size of the storage device. The maximum value for this parameter is determined by the value of the **max_hw_sectors_kb**.

Red Hat recommends **max_sectors_kb** to always be a multiple of the optimal I/O size and the internal erase block size. Use a value of **logical_block_size** for either parameter if they are zero or not specified by the storage device.

nomerges

Most workloads benefit from request merging. However, disabling merges can be useful for debugging purposes. By default, the **nomerges** parameter is set to **0**, which enables merging. To disable simple one-hit merging, set **nomerges** to **1**. To disable all types of merging, set **nomerges**

to **2**.

nr_requests

It is the maximum allowed number of the queued I/O. If the current I/O scheduler is **none**, this number can only be reduced; otherwise the number can be increased or reduced.

optimal_io_size

Some storage devices report an optimal I/O size through this parameter. If this value is reported, Red Hat recommends that applications issue I/O aligned to and in multiples of the optimal I/O size wherever possible.

read_ahead_kb

Defines the maximum number of kilobytes that the operating system may read ahead during a sequential read operation. As a result, the necessary information is already present within the kernel page cache for the next sequential read, which improves read I/O performance.

Device mappers often benefit from a high **read_ahead_kb** value. **128** KB for each device to be mapped is a good starting point, but increasing the **read_ahead_kb** value up to request queue's **max_sectors_kb** of the disk might improve performance in application environments where sequential reading of large files takes place.

rotational

Some solid-state disks do not correctly advertise their solid-state status, and are mounted as traditional rotational disks. Manually set the **rotational** value to **0** to disable unnecessary seek-reducing logic in the scheduler.

rq_affinity

The default value of the **rq_affinity** is **1**. It completes the I/O operations on one CPU core, which is in the same CPU group of the issued CPU core. To perform completions only on the processor that issued the I/O request, set the **rq_affinity** to **2**. To disable the mentioned two abilities, set it to **0**.

scheduler

To set the scheduler or scheduler preference order for a particular storage device, edit the **/sys/block/devname/queue/scheduler** file, where *devname* is the name of the device you want to configure.

CHAPTER 33. USING SYSTEMD TO MANAGE RESOURCES USED BY APPLICATIONS

RHEL 9 moves the resource management settings from the process level to the application level by binding the system of **cgroup** hierarchies with the **systemd** unit tree. Therefore, you can manage the system resources with the **systemctl** command, or by modifying the **systemd** unit files.

To achieve this, **systemd** takes various configuration options from the unit files or directly via the **systemctl** command. Then **systemd** applies those options to specific process groups by utilizing the Linux kernel system calls and features like **cgroups** and **namespaces**.



NOTE

You can review the full set of configuration options for **systemd** in the following manual pages:

- **systemd.resource-control(5)**
- **systemd.exec(5)**

33.1. ALLOCATING SYSTEM RESOURCES USING SYSTEMD

To modify the distribution of system resources, you can apply one or more of the following distribution models:

Weights

You can distribute the resource by adding up the weights of all sub-groups and giving each sub-group the fraction matching its ratio against the sum.

For example, if you have 10 cgroups, each with weight of value 100, the sum is 1000. Each cgroup receives one tenth of the resource.

Weight is usually used to distribute stateless resources. For example the *CPUWeight=* option is an implementation of this resource distribution model.

Limits

A cgroup can consume up to the configured amount of the resource. The sum of sub-group limits can exceed the limit of the parent cgroup. Therefore it is possible to overcommit resources in this model.

For example the *MemoryMax=* option is an implementation of this resource distribution model.

Protections

You can set up a protected amount of a resource for a cgroup. If the resource usage is below the protection boundary, the kernel will try not to penalize this cgroup in favor of other cgroups that compete for the same resource. An overcommit is also possible.

For example the *MemoryLow=* option is an implementation of this resource distribution model.

Allocations

Exclusive allocations of an absolute amount of a finite resource. An overcommit is not possible. An example of this resource type in Linux is the real-time budget.

unit file option

A setting for resource control configuration.

For example, you can configure CPU resource with options like *CPUAccounting=*, or *CPUQuota=*. Similarly, you can configure memory or I/O resources with options like *AllowedMemoryNodes=* and *IOAccounting=*.

Procedure

To change the required value of the unit file option of your service, you can adjust the value in the unit file, or use **systemctl** command:

1. Check the assigned values for the service of your choice.

```
# systemctl show --property <unit file option> <service name>
```

2. Set the required value of the CPU time allocation policy option:

```
# systemctl set-property <service name> <unit file option>=<value>
```

Verification steps

- Check the newly assigned values for the service of your choice.

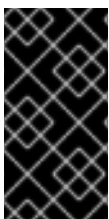
```
# systemctl show --property <unit file option> <service name>
```

Additional resources

- **systemd.resource-control(5)**, **systemd.exec(5)** manual pages

33.2. ROLE OF SYSTEMD IN RESOURCE MANAGEMENT

The core function of **systemd** is service management and supervision. The **systemd** system and service manager ensures that managed services start at the right time and in the correct order during the boot process. The services have to run smoothly to use the underlying hardware platform optimally. Therefore, **systemd** also provides capabilities to define resource management policies, and to tune various options, which can improve the performance of the service.



IMPORTANT

In general, Red Hat recommends you use **systemd** for controlling the usage of system resources. You should manually configure the **cgroups** virtual file system only in special cases. For example, when you need to use **cgroup-v1** controllers that have no equivalents in **cgroup-v2** hierarchy.

33.3. OVERVIEW OF SYSTEMD HIERARCHY FOR CGROUPS

On the backend, the **systemd** system and service manager makes use of the **slice**, the **scope** and the **service** units to organize and structure processes in the control groups. You can further modify this hierarchy by creating custom unit files or using the **systemctl** command. Also, **systemd** automatically mounts hierarchies for important kernel resource controllers at the **/sys/fs/cgroup/** directory.

Three **systemd** unit types are used for resource control:

- **Service** - A process or a group of processes, which **systemd** started according to a unit configuration file. Services encapsulate the specified processes so that they can be started and stopped as one set. Services are named in the following way:

```
<name>.service
```

- **Scope** - A group of externally created processes. Scopes encapsulate processes that are started and stopped by the arbitrary processes through the **fork()** function and then registered by **systemd** at runtime. For example, user sessions, containers, and virtual machines are treated as scopes. Scopes are named as follows:

```
<name>.scope
```

- **Slice** - A group of hierarchically organized units. Slices organize a hierarchy in which scopes and services are placed. The actual processes are contained in scopes or in services. Every name of a slice unit corresponds to the path to a location in the hierarchy. The dash ("-") character acts as a separator of the path components to a slice from the **-.slice** root slice. In the following example:

```
<parent-name>.slice
```

parent-name.slice is a sub-slice of **parent.slice**, which is a sub-slice of the **-.slice** root slice. **parent-name.slice** can have its own sub-slice named **parent-name-name2.slice**, and so on.

The **service**, the **scope**, and the **slice** units directly map to objects in the control group hierarchy. When these units are activated, they map directly to control group paths built from the unit names.

The following is an abbreviated example of a control group hierarchy:

```
Control group /:
-.slice
├── user.slice
│   ├── user-42.slice
│   │   ├── session-c1.scope
│   │   │   ├── 967 gdm-session-worker [pam/gdm-launch-environment]
│   │   │   ├── 1035 /usr/libexec/gdm-x-session gnome-session --autostart
│   │   │   └── /usr/share/gdm/greeter/autostart
│   │   │       ├── 1054 /usr/libexec/Xorg vt1 -displayfd 3 -auth /run/user/42/gdm/Xauthority -background none
│   │   │       ├── -noreset -keeptty -verbose 3
│   │   │       ├── 1212 /usr/libexec/gnome-session-binary --autostart /usr/share/gdm/greeter/autostart
│   │   │       ├── 1369 /usr/bin/gnome-shell
│   │   │       ├── 1732 ibus-daemon --xim --panel disable
│   │   │       ├── 1752 /usr/libexec/ibus-dconf
│   │   │       ├── 1762 /usr/libexec/ibus-x11 --kill-daemon
│   │   │       ├── 1912 /usr/libexec/gsd-xsettings
│   │   │       ├── 1917 /usr/libexec/gsd-a11y-settings
│   │   │       └── 1920 /usr/libexec/gsd-clipboard
│   │   └── ...
│   └── init.scope
│       ├── 1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
│       └── system.slice
│           ├── rngd.service
│           │   ├── 800 /sbin/rngd -f
│           ├── systemd-udevd.service
│           │   ├── 659 /usr/lib/systemd/systemd-udevd
```

```

├─chronyd.service
│   └─823 /usr/sbin/chronyd
├─auditd.service
│   ├──761 /sbin/auditd
│   └─763 /usr/sbin/sedispatch
├─accounts-daemon.service
│   └─876 /usr/libexec/accounts-daemon
├─example.service
│   ├── 929 /bin/bash /home/jdoe/example.sh
│   └─4902 sleep 1
...

```

The example above shows that services and scopes contain processes and are placed in slices that do not contain processes of their own.

Additional resources

- [Configuring basic system settings](#) in Red Hat Enterprise Linux
- [What are kernel resource controllers](#)
- **systemd.resource-control(5), systemd.exec(5), cgroups(7), fork(), fork(2)** manual pages
- [Understanding cgroups](#)

33.4. LISTING SYSTEMD UNITS

Use the **systemd** system and service manager to list its units.

Procedure

- List all active units on the system with the **# systemctl** command. The terminal will return an output similar to the following example:

```

# systemctl
UNIT                                LOAD    ACTIVE SUB    DESCRIPTION
...
init.scope                         loaded active running  System and Service Manager
session-2.scope                    loaded active running  Session 2 of user jdoe
abrt-ccpp.service                  loaded active exited  Install ABRT coredump hook
abrt-oops.service                  loaded active running  ABRT kernel log watcher
abrt-vmcore.service                loaded active exited  Harvest vmcores for ABRT
abrt-xorg.service                  loaded active running  ABRT Xorg log watcher
...
-.slice                            loaded active active   Root Slice
machine.slice                     loaded active active   Virtual Machine and Container
Slice system-getty.slice           loaded active active
system-getty.slice
system-lvm2\x2dpvscan.slice        loaded active active   system-
lvm2\x2dpvscan.slice
system-sshd\x2dkeygen.slice        loaded active active   system-
sshd\x2dkeygen.slice
system-systemd\x2dhibernate\x2dresume.slice  loaded active active   system-
systemd\x2dhibernate\x2dresume>
system-user\x2druntime\x2ddir.slice  loaded active active   system-

```

```

user\x2druntime\x2ddir.slice
system.slice          loaded active active  System Slice
user-1000.slice        loaded active active  User Slice of UID 1000
user-42.slice          loaded active active  User Slice of UID 42
user.slice             loaded active active  User and Session Slice
...

```

- **UNIT** – a name of a unit that also reflects the unit position in a control group hierarchy. The units relevant for resource control are a *slice*, a *scope*, and a *service*.
 - **LOAD** – indicates whether the unit configuration file was properly loaded. If the unit file failed to load, the field contains the state *error* instead of *loaded*. Other unit load states are: *stub*, *merged*, and *masked*.
 - **ACTIVE** – the high-level unit activation state, which is a generalization of **SUB**.
 - **SUB** – the low-level unit activation state. The range of possible values depends on the unit type.
 - **DESCRIPTION** – the description of the unit content and functionality.
- List inactive units.

```
# systemctl --all
```

- Limit the amount of information in the output.

```
# systemctl --type service,masked
```

The **--type** option requires a comma-separated list of unit types such as a *service* and a *slice*, or unit load states such as *loaded* and *masked*.

Additional resources

- [Configuring basic system settings](#) in RHEL
- The **systemd.resource-control(5)**, **systemd.exec(5)** manual pages

33.5. VIEWING SYSTEMD CONTROL GROUP HIERARCHY

Display control groups (**cgroups**) hierarchy and processes running in specific **cgroups**.

Procedure

- Display the whole **cgroups** hierarchy on your system with the **systemd-cgls** command.

```

# systemd-cgls
Control group /:
-.slice
  └─user.slice
      └─user-42.slice
          └─session-c1.scope
              └─965 gdm-session-worker [pam/gdm-launch-environment]
              └─1040 /usr/libexec/gdm-x-session gnome-session --autostart
                  /usr/share/gdm/greeter/autostart

```

```

...
└─init.scope
   └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
      └─system.slice
         ...
         └─example.service
            └─6882 /bin/bash /home/jdoe/example.sh
               └─6902 sleep 1
         └─systemd-journald.service
            └─629 /usr/lib/systemd/systemd-journald
         ...

```

The example output returns the entire **cgroups** hierarchy, where the highest level is formed by *slices*.

- Display the **cgroups** hierarchy filtered by a resource controller with the **systemd-cgls** *<resource_controller>* command.

```

# systemd-cgls memory
Controller memory; Control group /:
└─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
   └─user.slice
      └─user-42.slice
         └─session-c1.scope
            └─965 gdm-session-worker [pam/gdm-launch-environment]
      ...
   └─system.slice
      |
      ...
      └─chronyd.service
         └─844 /usr/sbin/chronyd
      └─example.service
         └─8914 /bin/bash /home/jdoe/example.sh
            └─8916 sleep 1
      ...

```

The example output of the above command lists the services that interact with the selected controller.

- Display detailed information about a certain unit and its part of the **cgroups** hierarchy with the **systemctl status** *<system_unit>* command.

```

# systemctl status example.service
● example.service - My example service
   Loaded: loaded (/usr/lib/systemd/system/example.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2019-04-16 12:12:39 CEST; 3s ago
 Main PID: 17737 (bash)
    Tasks: 2 (limit: 11522)
   Memory: 496.0K (limit: 1.5M)
    CGroup: /system.slice/example.service
            └─17737 /bin/bash /home/jdoe/example.sh
               └─17743 sleep 1

Apr 16 12:12:39 redhat systemd[1]: Started My example service.
Apr 16 12:12:39 redhat bash[17737]: The current time is Tue Apr 16 12:12:39 CEST 2019
Apr 16 12:12:40 redhat bash[17737]: The current time is Tue Apr 16 12:12:40 CEST 2019

```

-

Additional resources

- [What are kernel resource controllers](#)
- The **systemd.resource-control(5)**, **cgroups(7)** manual pages

33.6. VIEWING CGROUPS OF PROCESSES

The following procedure describes how to learn which *control group* (**cgroup**) a process belongs to. Then you can check the **cgroup** to learn which controllers and controller-specific configurations it uses.

Procedure

1. To view which **cgroup** a process belongs to, run the **# cat proc/<PID>/cgroup** command:

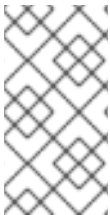
```
# cat /proc/2467/cgroup
0::/system.slice/example.service
```

The example output relates to a process of interest. In this case, it is a process identified by **PID 2467**, which belongs to the **example.service** unit. You can determine whether the process was placed in a correct control group as defined by the **systemd** unit file specifications.

2. To display what controllers the **cgroup** utilizes and the respective configuration files, check the **cgroup** directory:

```
# cat /sys/fs/cgroup/system.slice/example.service/cgroup.controllers
memory pids

# ls /sys/fs/cgroup/system.slice/example.service/
cgroup.controllers
cgroup.events
...
cpu.pressure
cpu.stat
io.pressure
memory.current
memory.events
...
pids.current
pids.events
pids.max
```



NOTE

The version 1 hierarchy of **cgroups** uses a per-controller model. Therefore the output from the **/proc/PID/cgroup** file shows, which **cgroups** under each controller the PID belongs to. You can find the respective **cgroups** under the controller directories at **/sys/fs/cgroup/<controller_name>/**.

Additional resources

- **cgroups(7)** manual page

- [What are kernel resource controllers](#)
- Documentation in the `/usr/share/doc/kernel-doc-<kernel_version>/Documentation/admin-guide/cgroup-v2.rst` file (after installing the **kernel-doc** package)

33.7. MONITORING RESOURCE CONSUMPTION

View a list of currently running control groups (**cgroups**) and their resource consumption in real-time.

Procedure

1. Display a dynamic account of currently running **cgroups** with the **systemd-cgtop** command.

```
# systemd-cgtop
Control Group          Tasks %CPU  Memory Input/s Output/s
/                    607 29.8  1.5G   -      -
/system.slice         125  -    428.7M   -      -
/system.slice/ModemManager.service      3  -    8.6M   -      -
/system.slice/NetworkManager.service    3  -   12.8M   -      -
/system.slice/accounts-daemon.service    3  -    1.8M   -      -
/system.slice/boot.mount                 -  -    48.0K   -      -
/system.slice/chronyd.service            1  -    2.0M   -      -
/system.slice/cockpit.socket             -  -    1.3M   -      -
/system.slice/colord.service             3  -    3.5M   -      -
/system.slice/crond.service              1  -    1.8M   -      -
/system.slice/cups.service               1  -    3.1M   -      -
/system.slice/dev-hugepages.mount        -  -   244.0K   -      -
/system.slice/dev-mapper-rhelx2dswap.swap -  -   912.0K   -      -
/system.slice/dev-mqueue.mount           -  -    48.0K   -      -
/system.slice/example.service            2  -    2.0M   -      -
/system.slice/firewalld.service          2  -   28.8M   -      -
...
```

The example output displays currently running **cgroups** ordered by their resource usage (CPU, memory, disk I/O load). The list refreshes every 1 second by default. Therefore, it offers a dynamic insight into the actual resource usage of each control group.

Additional resources

- The **systemd-cgtop(1)** manual page

33.8. USING SYSTEMD UNIT FILES TO SET LIMITS FOR APPLICATIONS

Each existing or running unit is supervised by the **systemd**, which also creates control groups for them. The units have configuration files in the `/usr/lib/systemd/system/` directory. You can manually modify the unit files to set limits, prioritize, or control access to hardware resources for groups of processes.

Prerequisites

- You have the **root** privileges.

Procedure

1. Modify the `/usr/lib/systemd/system/example.service` file to limit the memory usage of a service:

```
...
[Service]
MemoryMax=1500K
...
```

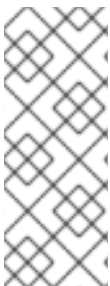
The configuration above places a maximum memory limit, which the processes in a control group cannot exceed. The **example.service** service is part of such a control group which has imposed limitations. You can use suffixes K, M, G, or T to identify Kilobyte, Megabyte, Gigabyte, or Terabyte as a unit of measurement.

2. Reload all unit configuration files:

```
# systemctl daemon-reload
```

3. Restart the service:

```
# systemctl restart example.service
```



NOTE

You can review the full set of configuration options for **systemd** in the following manual pages:

- **systemd.resource-control(5)**
- **systemd.exec(5)**

Verification

1. Check that the changes took effect:

```
# cat /sys/fs/cgroup/system.slice/example.service/memory.max
1536000
```

The example output shows that the memory consumption was limited at around 1,500 KB.

Additional resources

- [Understanding cgroups](#)
- [Configuring basic system settings](#) in Red Hat Enterprise Linux
- **systemd.resource-control(5)**, **systemd.exec(5)**, **cgroups(7)** manual pages

33.9. USING SYSTEMCTL COMMAND TO SET LIMITS TO APPLICATIONS

CPU affinity settings help you restrict the access of a particular process to some CPUs. Effectively, the CPU scheduler never schedules the process to run on the CPU that is not in the affinity mask of the process.

The default CPU affinity mask applies to all services managed by **systemd**.

To configure CPU affinity mask for a particular **systemd** service, **systemd** provides **CPUAffinity=** both as a unit file option and a manager configuration option in the **/etc/systemd/system.conf** file.

The **CPUAffinity= unit file option** sets a list of CPUs or CPU ranges that are merged and used as the affinity mask.

After configuring CPU affinity mask for a particular **systemd** service, you must restart the service to apply the changes.

Procedure

To set CPU affinity mask for a particular **systemd** service using the **CPUAffinity unit file** option:

1. Check the values of the **CPUAffinity** unit file option in the service of your choice:

```
$ systemctl show --property <CPU affinity configuration option> <service name>
```

2. As a root, set the required value of the **CPUAffinity** unit file option for the CPU ranges used as the affinity mask:

```
# systemctl set-property <service name> CPUAffinity=<value>
```

3. Restart the service to apply the changes.

```
# systemctl restart <service name>
```



NOTE

You can review the full set of configuration options for **systemd** in the following manual pages:

- **systemd.resource-control(5)**
- **systemd.exec(5)**

33.10. SETTING GLOBAL DEFAULT CPU AFFINITY THROUGH MANAGER CONFIGURATION

The **CPUAffinity** option in the **/etc/systemd/system.conf** file defines an affinity mask for the process identification number (PID) 1 and all processes forked off of PID1. You can then override the **CPUAffinity** on a per-service basis.

To set default CPU affinity mask for all systemd services using the **manager configuration** option:

1. Set the CPU numbers for the **CPUAffinity=** option in the **/etc/systemd/system.conf** file.
2. Save the edited file and reload the **systemd** service:

```
# systemctl daemon-reload
```

3. Reboot the server to apply the changes.

**NOTE**

You can review the full set of configuration options for **systemd** in the following manual pages:

- **systemd.resource-control(5)**
- **systemd.exec(5)**

33.11. CONFIGURING NUMA POLICIES USING SYSTEMD

Non-uniform memory access (NUMA) is a computer memory subsystem design, in which the memory access time depends on the physical memory location relative to the processor.

Memory close to the CPU has lower latency (local memory) than memory that is local for a different CPU (foreign memory) or is shared between a set of CPUs.

In terms of the Linux kernel, NUMA policy governs where (for example, on which NUMA nodes) the kernel allocates physical memory pages for the process.

systemd provides unit file options **NUMAPolicy** and **NUMAMask** to control memory allocation policies for services.

Procedure

To set the NUMA memory policy through the **NUMAPolicy** unit file option:

1. Check the values of the **NUMAPolicy** unit file option in the service of your choice:

```
$ systemctl show --property <NUMA policy configuration option> <service name>
```

2. As a root, set the required policy type of the **NUMAPolicy** unit file option:

```
# systemctl set-property <service name> NUMAPolicy=<value>
```

3. Restart the service to apply the changes.

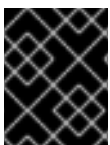
```
# systemctl restart <service name>
```

To set a global **NUMAPolicy** setting through the **manager configuration** option:

1. Search in the **/etc/systemd/system.conf** file for the **NUMAPolicy** option.
2. Edit the policy type and save the file.
3. Reload the **systemd** configuration:

```
# systemd daemon-reload
```

4. Reboot the server.

**IMPORTANT**

When you configure a strict NUMA policy, for example **bind**, make sure that you also appropriately set the **CPUAffinity=** unit file option.

Additional resources

- [Using systemctl command to set limits to applications](#)
- The **systemd.resource-control(5)**, **systemd.exec(5)**, **set_mempolicy(2)** manual pages.

33.12. NUMA POLICY CONFIGURATION OPTIONS FOR SYSTEMD

Systemd provides the following options to configure the NUMA policy:

NUMAPolicy

Controls the NUMA memory policy of the executed processes. The following policy types are possible:

- default
- preferred
- bind
- interleave
- local

NUMAMask

Controls the NUMA node list which is associated with the selected NUMA policy.

Note that the **NUMAMask** option is not required to be specified for the following policies:

- default
- local

For the preferred policy, the list specifies only a single NUMA node.

Additional resources

- **systemd.resource-control(5)**, **systemd.exec(5)**, and **set_mempolicy(2)** manual pages

33.13. CREATING TRANSIENT CGROUPS USING SYSTEMD-RUN COMMAND

The transient **cgroups** set limits on resources consumed by a unit (service or scope) during its runtime.

Procedure

- To create a transient control group, use the **systemd-run** command in the following format:

```
# systemd-run --unit=<name> --slice=<name>.slice <command>
```

This command creates and starts a transient service or a scope unit and runs a custom command in such a unit.

- The **--unit=<name>** option gives a name to the unit. If **--unit** is not specified, the name is generated automatically.
- The **--slice=<name>.slice** option makes your service or scope unit a member of a specified slice. Replace **<name>.slice** with the name of an existing slice (as shown in the output of **systemctl -t slice**), or create a new slice by passing a unique name. By default, services and scopes are created as members of the **system.slice**.
- Replace **<command>** with the command you wish to execute in the service or the scope unit.
The following message is displayed to confirm that you created and started the service or the scope successfully:

```
# Running as unit <name>.service
```

- Optionally, keep the unit running after its processes finished to collect run-time information:

```
# systemd-run --unit=<name> --slice=<name>.slice --remain-after-exit <command>
```

The command creates and starts a transient service unit and runs a custom command in such a unit. The **--remain-after-exit** option ensures that the service keeps running after its processes have finished.

Additional resources

- [What are control groups](#)
- [Configuring basic system settings](#) in RHEL
- the **systemd-run(1)** manual page

33.14. REMOVING TRANSIENT CONTROL GROUPS

You can use the **systemd** system and service manager to remove transient control groups (**cgroups**) if you no longer need to limit, prioritize, or control access to hardware resources for groups of processes.

Transient **cgroups** are automatically released once all the processes that a service or a scope unit contains, finish.

Procedure

- To stop the service unit with all its processes, execute:

```
# systemctl stop name.service
```

- To terminate one or more of the unit processes, execute:

```
# systemctl kill name.service --kill-who=PID,... --signal=<signal>
```

The command above uses the **--kill-who** option to select process(es) from the control group you wish to terminate. To kill multiple processes at the same time, pass a comma-separated list of PIDs. The **--signal** option determines the type of POSIX signal to be sent to the specified processes. The default signal is **SIGTERM**.

Additional resources

- [What are control groups](#)
- [What are kernel resource controllers](#)
- **systemd.resource-control(5), cgroups(7)** manual pages
- [Role of systemd in control groups](#)
- [Configuring basic system settings](#) in RHEL

CHAPTER 34. UNDERSTANDING CONTROL GROUPS

Using the control groups (**cgroups**) kernel functionality, you can control resource usage of applications to use them more efficiently.

You can use **cgroups** for the following tasks:

- Setting limits for system resource allocation.
- Prioritizing the allocation of hardware resources to specific processes.
- Isolating certain processes from obtaining hardware resources.

34.1. INTRODUCING CONTROL GROUPS

Using the *control groups* Linux kernel feature, you can organize processes into hierarchically ordered groups – **cgroups**. You define the hierarchy (control groups tree) by providing structure to **cgroups** virtual file system, mounted by default on the **/sys/fs/cgroup/** directory.

The **systemd** service manager uses **cgroups** to organize all units and services that it governs. Manually, you can manage the hierarchies of **cgroups** by creating and removing sub-directories in the **/sys/fs/cgroup/** directory.

The resource controllers in the kernel then modify the behavior of processes in **cgroups** by limiting, prioritizing or allocating system resources, of those processes. These resources include the following:

- CPU time
- Memory
- Network bandwidth
- Combinations of these resources

The primary use case of **cgroups** is aggregating system processes and dividing hardware resources among applications and users. This makes it possible to increase the efficiency, stability, and security of your environment.

Control groups version 1

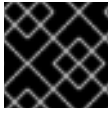
Control groups version 1 (**cgroups-v1**) provide a per-resource controller hierarchy. This means that each resource (such as CPU, memory, or I/O) has its own control group hierarchy. You can combine different control group hierarchies in a way that one controller can coordinate with another in managing their respective resources. However, when the two controllers belong to different process hierarchies, proper coordination is limited.

The **cgroups-v1** controllers were developed across a large time span and as a result, the behavior and naming of their control files is not uniform.

Control groups version 2

Control groups version 2 (**cgroups-v2**) provide a single control group hierarchy against which all resource controllers are mounted.

The control file behavior and naming is consistent among different controllers.



IMPORTANT

RHEL 9, by default, mounts and uses **cgroups-v2**.

Additional resources

- [Introducing kernel resource controllers](#)
- The **cgroups(7)** manual page
- [cgroups-v1](#)
- [cgroups-v2](#)

34.2. INTRODUCING KERNEL RESOURCE CONTROLLERS

Kernel resource controllers enable the functionality of control groups. RHEL 9 supports various controllers for *control groups version 1* (**cgroups-v1**) and *control groups version 2* (**cgroups-v2**).

A resource controller, also called a control group subsystem, is a kernel subsystem that represents a single resource, such as CPU time, memory, network bandwidth or disk I/O. The Linux kernel provides a range of resource controllers that are mounted automatically by the **systemd** service manager. You can find a list of the currently mounted resource controllers in the **/proc/cgroups** file.

Table 34.1. Controllers available for cgroups-v1:

blkio	Sets limits on input/output access to and from block devices.
cpu	Adjusts the parameters of the Completely Fair Scheduler (CFS) for a control group's tasks. The cpu controller is mounted together with the cpuacct controller on the same mount.
cpuacct	Creates automatic reports on CPU resources used by tasks in a control group. The cpuacct controller is mounted together with the cpu controller on the same mount.
cpuset	Restricts control group tasks to run only on a specified subset of CPUs and to direct the tasks to use memory only on specified memory nodes.
devices	Controls access to devices for tasks in a control group.
freezer	Suspends or resumes tasks in a control group.
memory	Sets limits on memory use by tasks in a control group and generates automatic reports on memory resources used by those tasks.
net_cls	Tags network packets with a class identifier (classid) that enables the Linux traffic controller (the tc command) to identify packets that originate from a particular control group task. A subsystem of net_cls , the net_filter (iptables), can also use this tag to perform actions on such packets. The net_filter tags network sockets with a firewall identifier (fwid) that allows the Linux firewall to identify packets that originate from a particular control group task (by using the iptables command).

net_prio	Sets the priority of network traffic.
pids	Sets limits for a number of processes and their children in a control group.
perf_event	Groups tasks for monitoring by the perf performance monitoring and reporting utility.
rdma	Sets limits on Remote Direct Memory Access/InfiniBand specific resources in a control group.
hugetlb	can be used to limit the usage of large size virtual memory pages by tasks in a control group.

Table 34.2. Controllers available **forcgroups-v2**:

io	Sets limits on input/output access to and from block devices.
memory	Sets limits on memory use by tasks in a control group and generates automatic reports on memory resources used by those tasks.
pids	Sets limits for a number of processes and their children in a control group.
rdma	Sets limits on Remote Direct Memory Access/InfiniBand specific resources in a control group.
cpu	Adjusts the parameters of the Completely Fair Scheduler (CFS) for a control group's tasks and creates automatic reports on CPU resources used by tasks in a control group.
cpuset	Restricts control group tasks to run only on a specified subset of CPUs and to direct the tasks to use memory only on specified memory nodes. Supports only the core functionality (cpus{,.effective} , mems{,.effective}) with a new partition feature.
perf_event	Groups tasks for monitoring by the perf performance monitoring and reporting utility. perf_event is enabled automatically on the v2 hierarchy.

**IMPORTANT**

A resource controller can be used either in a **cgroups-v1** hierarchy or a **cgroups-v2** hierarchy, not simultaneously in both.

Additional resources

- The **cgroups(7)** manual page
- Documentation in **/usr/share/doc/kernel-doc-<kernel_version>/Documentation/cgroups-v1/** directory (after installing the **kernel-doc** package).

34.3. INTRODUCING NAMESPACES

Namespaces are one of the most important methods for organizing and identifying software objects.

A namespace wraps a global system resource (for example, a mount point, a network device, or a hostname) in an abstraction that makes it appear to processes within the namespace that they have their own isolated instance of the global resource. One of the most common technologies that use namespaces are containers.

Changes to a particular global resource are visible only to processes in that namespace and do not affect the rest of the system or other namespaces.

To inspect which namespaces a process is a member of, you can check the symbolic links in the **/proc/<PID>/ns/** directory.

Table 34.3. Supported namespaces and resources which they isolate:

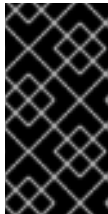
Namespace	Isolates
Mount	Mount points
UTS	Hostname and NIS domain name
IPC	System V IPC, POSIX message queues
PID	Process IDs
Network	Network devices, stacks, ports, etc
User	User and group IDs
Control groups	Control group root directory

Additional resources

- The **namespaces(7)** and **cgroup_namespaces(7)** manual pages
- [Introducing control groups](#)

CHAPTER 35. USING CGROUPFS TO MANUALLY MANAGE CGROUPS

You can manage **cgroup** hierarchies on your system by creating directories on the **cgroupfs** virtual file system. The file system is mounted by default on the **/sys/fs/cgroup/** directory and you can specify desired configurations in dedicated control files.



IMPORTANT

In general, Red Hat recommends you use **systemd** for controlling the usage of system resources. You should manually configure the **cgroups** virtual file system only in special cases. For example, when you need to use **cgroup-v1** controllers that have no equivalents in **cgroup-v2** hierarchy.

35.1. CREATING CGROUPS AND ENABLING CONTROLLERS IN CGROUPS-V2 FILE SYSTEM

You can manage the *control groups* (**cgroups**) by creating or removing directories and by writing to files in the **cgroups** virtual file system. The file system is by default mounted on the **/sys/fs/cgroup/** directory. To use settings from the **cgroups** controllers, you also need to enable the desired controllers for child **cgroups**. The root **cgroup** has, by default, enabled the **memory** and **pids** controllers for its child **cgroups**. Therefore, Red Hat recommends to create at least two levels of child **cgroups** inside the **/sys/fs/cgroup/** root **cgroup**. This way you optionally remove the **memory** and **pids** controllers from the child **cgroups** and maintain better organizational clarity of **cgroup** files.

Prerequisites

- You have root permissions.

Procedure

1. Create the **/sys/fs/cgroup/Example/** directory:

```
# mkdir /sys/fs/cgroup/Example/
```

The **/sys/fs/cgroup/Example/** directory defines a child group. When you create the **/sys/fs/cgroup/Example/** directory, some **cgroups-v2** interface files are automatically created in the directory. The **/sys/fs/cgroup/Example/** directory contains also controller-specific files for the **memory** and **pids** controllers.

2. Optionally, inspect the newly created child control group:

```
# ll /sys/fs/cgroup/Example/
-r--r--r--. 1 root root 0 Jun  1 10:33 cgroup.controllers
-r--r--r--. 1 root root 0 Jun  1 10:33 cgroup.events
-rw-r--r--. 1 root root 0 Jun  1 10:33 cgroup.freeze
-rw-r--r--. 1 root root 0 Jun  1 10:33 cgroup.procs
...
-rw-r--r--. 1 root root 0 Jun  1 10:33 cgroup.subtree_control
-r--r--r--. 1 root root 0 Jun  1 10:33 memory.events.local
-rw-r--r--. 1 root root 0 Jun  1 10:33 memory.high
-rw-r--r--. 1 root root 0 Jun  1 10:33 memory.low
...
```

```
-r—r—r--. 1 root root 0 Jun  1 10:33 pids.current
-r—r—r--. 1 root root 0 Jun  1 10:33 pids.events
-rw-r—r--. 1 root root 0 Jun  1 10:33 pids.max
```

The example output shows general **cgroup** control interface files such as **cgroup.procs** or **cgroup.controllers**. These files are common to all control groups, regardless of enabled controllers.

The files such as **memory.high** and **pids.max** relate to the **memory** and **pids** controllers, which are in the root control group (**/sys/fs/cgroup/**), and are enabled by default by **systemd**.

By default, the newly created child group inherits all settings from the parent **cgroup**. In this case, there are no limits from the root **cgroup**.

3. Verify that the desired controllers are available in the **/sys/fs/cgroup/cgroup.controllers** file:

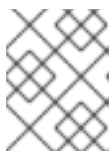
```
# cat /sys/fs/cgroup/cgroup.controllers
cpuset cpu io memory hugetlb pids rdma
```

4. Enable the desired controllers. In this example it is **cpu** and **cpuset** controllers:

```
# echo "+cpu" >> /sys/fs/cgroup/cgroup.subtree_control
# echo "+cpuset" >> /sys/fs/cgroup/cgroup.subtree_control
```

These commands enable the **cpu** and **cpuset** controllers for the immediate child groups of the **/sys/fs/cgroup/** root control group. Including the newly created **Example** control group. A *child group* is where you can specify processes and apply control checks to each of the processes based on your criteria.

Users can read the contents of the **cgroup.subtree_control** file at any level to get an idea of what controllers are going to be available for enablement in the immediate child group.



NOTE

By default, the **/sys/fs/cgroup/cgroup.subtree_control** file in the root control group contains **memory** and **pids** controllers.

5. Enable the desired controllers for child **cgroups** of the **Example** control group:

```
# echo "+cpu +cpuset" >> /sys/fs/cgroup/Example/cgroup.subtree_control
```

This command ensures that the immediate child control group will *only* have controllers relevant to regulate the CPU time distribution – not to **memory** or **pids** controllers.

6. Create the **/sys/fs/cgroup/Example/tasks/** directory:

```
# mkdir /sys/fs/cgroup/Example/tasks/
```

The **/sys/fs/cgroup/Example/tasks/** directory defines a child group with files that relate purely to **cpu** and **cpuset** controllers. You can now assign processes to this control group and utilize **cpu** and **cpuset** controller options for your processes.

7. Optionally, inspect the child control group:

ll /sys/fs/cgroup/Example/tasks

```

-r--r--r--. 1 root root 0 Jun 1 11:45 cgroup.controllers
-r--r--r--. 1 root root 0 Jun 1 11:45 cgroup.events
-rw-r--r--. 1 root root 0 Jun 1 11:45 cgroup.freeze
-rw-r--r--. 1 root root 0 Jun 1 11:45 cgroup.max.depth
-rw-r--r--. 1 root root 0 Jun 1 11:45 cgroup.max.descendants
-rw-r--r--. 1 root root 0 Jun 1 11:45 cgroup.procs
-r--r--r--. 1 root root 0 Jun 1 11:45 cgroup.stat
-rw-r--r--. 1 root root 0 Jun 1 11:45 cgroup.subtree_control
-rw-r--r--. 1 root root 0 Jun 1 11:45 cgroup.threads
-rw-r--r--. 1 root root 0 Jun 1 11:45 cgroup.type
-rw-r--r--. 1 root root 0 Jun 1 11:45 cpu.max
-rw-r--r--. 1 root root 0 Jun 1 11:45 cpu.pressure
-rw-r--r--. 1 root root 0 Jun 1 11:45 cpuset.cpus
-r--r--r--. 1 root root 0 Jun 1 11:45 cpuset.cpus.effective
-rw-r--r--. 1 root root 0 Jun 1 11:45 cpuset.cpus.partition
-rw-r--r--. 1 root root 0 Jun 1 11:45 cpuset.mems
-r--r--r--. 1 root root 0 Jun 1 11:45 cpuset.mems.effective
-r--r--r--. 1 root root 0 Jun 1 11:45 cpu.stat
-rw-r--r--. 1 root root 0 Jun 1 11:45 cpu.weight
-rw-r--r--. 1 root root 0 Jun 1 11:45 cpu.weight.nice
-rw-r--r--. 1 root root 0 Jun 1 11:45 io.pressure
-rw-r--r--. 1 root root 0 Jun 1 11:45 memory.pressure

```

**IMPORTANT**

The **cpu** controller is only activated if the relevant child control group has at least 2 processes which compete for time on a single CPU.

Verification steps

- Optional: confirm that you have created a new **cgroup** with only the desired controllers active:

```

# cat /sys/fs/cgroup/Example/tasks/cgroup.controllers
cpuset cpu

```

Additional resources

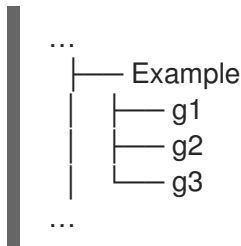
- [Understanding control groups](#)
- [What are kernel resource controllers](#)
- [Mounting cgroups-v1](#)
- **cgroups(7)**, **sysfs(5)** manual pages

35.2. CONTROLLING DISTRIBUTION OF CPU TIME FOR APPLICATIONS BY ADJUSTING CPU WEIGHT

You need to assign values to the relevant files of the **cpu** controller to regulate distribution of the CPU time to applications under the specific cgroup tree.

Prerequisites

- You have root permissions.
- You have applications for which you want to control distribution of CPU time.
- You created a two level hierarchy of *child control groups* inside the **/sys/fs/cgroup/** root control group as in the following example:



- You enabled the **cpu** controller in the parent control group and in child control groups similarly as described in [Creating cgroups and enabling controllers in cgroups-v2 file system](#) .

Procedure

1. Configure desired CPU weights to achieve resource restrictions within the control groups:

```
# echo "150" > /sys/fs/cgroup/Example/g1/cpu.weight
# echo "100" > /sys/fs/cgroup/Example/g2/cpu.weight
# echo "50" > /sys/fs/cgroup/Example/g3/cpu.weight
```

2. Add the applications' PIDs to the **g1**, **g2**, and **g3** child groups:

```
# echo "33373" > /sys/fs/cgroup/Example/g1/cgroup.procs
# echo "33374" > /sys/fs/cgroup/Example/g2/cgroup.procs
# echo "33377" > /sys/fs/cgroup/Example/g3/cgroup.procs
```

The example commands ensure that desired applications become members of the **Example/g*** child cgroups and will get their CPU time distributed as per the configuration of those cgroups.

The weights of the children cgroups (**g1**, **g2**, **g3**) that have running processes are summed up at the level of the parent cgroup (**Example**). The CPU resource is then distributed proportionally based on the respective weights.

As a result, when all processes run at the same time, the kernel allocates to each of them the proportionate CPU time based on their respective cgroup's **cpu.weight** file:

Child cgroup	cpu.weight file	CPU time allocation
g1	150	~50% (150/300)
g2	100	~33% (100/300)
g3	50	~16% (50/300)

The value of the **cpu.weight** controller file is not a percentage.

If one process stopped running, leaving cgroup **g2** with no running processes, the calculation would omit the cgroup **g2** and only account weights of cgroups **g1** and **g3**:

Child cgroup	cpu.weight file	CPU time allocation
g1	150	~75% (150/200)
g3	50	~25% (50/200)



IMPORTANT

If a child cgroup had multiple running processes, the CPU time allocated to the respective cgroup would be distributed equally to the member processes of that cgroup.

Verification

1. Verify that the applications run in the specified control groups:

```
# cat /proc/33373/cgroup /proc/33374/cgroup /proc/33377/cgroup
0::/Example/g1
0::/Example/g2
0::/Example/g3
```

The command output shows the processes of the specified applications that run in the **Example/g*** child cgroups.

2. Inspect the current CPU consumption of the throttled applications:

```
# top
top - 05:17:18 up 1 day, 18:25, 1 user, load average: 3.03, 3.03, 3.00
Tasks: 95 total, 4 running, 91 sleeping, 0 stopped, 0 zombie
%Cpu(s): 18.1 us, 81.6 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.3 hi, 0.0 si, 0.0 st
MiB Mem : 3737.0 total, 3233.7 free, 132.8 used, 370.5 buff/cache
MiB Swap: 4060.0 total, 4060.0 free, 0.0 used. 3373.1 avail Mem

  PID USER   PR NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 33373 root    20  0 18720 1748 1460 R  49.5  0.0 415:05.87 sha1sum
 33374 root    20  0 18720 1756 1464 R  32.9  0.0 412:58.33 sha1sum
 33377 root    20  0 18720 1860 1568 R  16.3  0.0 411:03.12 sha1sum
   760 root    20  0 416620 28540 15296 S  0.3  0.7  0:10.23 tuned
     1 root    20  0 186328 14108 9484 S  0.0  0.4  0:02.00 systemd
     2 root    20  0   0   0   0 S  0.0  0.0  0:00.01 kthread
...
```



NOTE

We forced all the example processes to run on a single CPU for clearer illustration. The CPU weight applies the same principles also when used on multiple CPUs.

Notice that the CPU resource for the **PID 33373**, **PID 33374**, and **PID 33377** was allocated based on the weights, 150, 100, 50, you assigned to the respective child cgroups. The weights correspond to around 50%, 33%, and 16% allocation of CPU time for each application.

Additional resources

- [Understanding control groups](#)
- [What are kernel resource controllers](#)
- [Creating cgroups and enabling controllers in cgroups-v2 file system](#)
- [Resource Distribution Models](#)
- **cgroups(7)**, **sysfs(5)** manual pages

35.3. MOUNTING CGROUPS-V1

During the boot process, RHEL 9 mounts the **cgroup-v2** virtual filesystem by default. To utilize **cgroup-v1** functionality in limiting resources for your applications, manually configure the system.



NOTE

Both **cgroup-v1** and **cgroup-v2** are fully enabled in the kernel. There is no default control group version from the kernel point of view, and is decided by **systemd** to mount at startup.

Prerequisites

- You have root permissions.

Procedure

1. Configure the system to mount **cgroups-v1** by default during system boot by the **systemd** system and service manager:

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --
args="systemd.unified_cgroup_hierarchy=0
systemd.legacy_systemd_cgroup_controller"
```

This adds the necessary kernel command-line parameters to the current boot entry.

To add the same parameters to all kernel boot entries:

```
# grubby --update-kernel=ALL --args="systemd.unified_cgroup_hierarchy=0
systemd.legacy_systemd_cgroup_controller"
```

2. Reboot the system for the changes to take effect.

Verification

1. Optionally, verify that the **cgroups-v1** filesystem was mounted:

```
# mount -l | grep cgroup
```

```

tmpfs on /sys/fs/cgroup type tmpfs
(ro,nosuid,nodev,noexec,seclabel,size=4096k,nr_inodes=1024,mode=755,inode64)
cgroup on /sys/fs/cgroup/systemd type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,xattr,release_agent=/usr/lib/systemd/systemd-
cgroups-agent,name=systemd)
cgroup on /sys/fs/cgroup/perf_event type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,perf_event)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,cpu,cpuacct)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,seclabel,pids)
cgroup on /sys/fs/cgroup/cpuset type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,cpuset)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,net_cls,net_prio)
cgroup on /sys/fs/cgroup/hugetlb type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,hugetlb)
cgroup on /sys/fs/cgroup/memory type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,memory)
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,seclabel,blkio)
cgroup on /sys/fs/cgroup/devices type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,devices)
cgroup on /sys/fs/cgroup/misc type cgroup (rw,nosuid,nodev,noexec,relatime,seclabel,misc)
cgroup on /sys/fs/cgroup/freezer type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,freezer)
cgroup on /sys/fs/cgroup/rdma type cgroup (rw,nosuid,nodev,noexec,relatime,seclabel,rdma)

```

The **cgroups-v1** filesystems that correspond to various **cgroup-v1** controllers, were successfully mounted on the **/sys/fs/cgroup/** directory.

2. Optionally, inspect the contents of the **/sys/fs/cgroup/** directory:

```

# ll /sys/fs/cgroup/
dr-xr-xr-x. 10 root root 0 Mar 16 09:34 blkio
lrwxrwxrwx. 1 root root 11 Mar 16 09:34 cpu -> cpu,cpuacct
lrwxrwxrwx. 1 root root 11 Mar 16 09:34 cpuacct -> cpu,cpuacct
dr-xr-xr-x. 10 root root 0 Mar 16 09:34 cpu,cpuacct
dr-xr-xr-x. 2 root root 0 Mar 16 09:34 cpuset
dr-xr-xr-x. 10 root root 0 Mar 16 09:34 devices
dr-xr-xr-x. 2 root root 0 Mar 16 09:34 freezer
dr-xr-xr-x. 2 root root 0 Mar 16 09:34 hugetlb
dr-xr-xr-x. 10 root root 0 Mar 16 09:34 memory
dr-xr-xr-x. 2 root root 0 Mar 16 09:34 misc
lrwxrwxrwx. 1 root root 16 Mar 16 09:34 net_cls -> net_cls,net_prio
dr-xr-xr-x. 2 root root 0 Mar 16 09:34 net_cls,net_prio
lrwxrwxrwx. 1 root root 16 Mar 16 09:34 net_prio -> net_cls,net_prio
dr-xr-xr-x. 2 root root 0 Mar 16 09:34 perf_event
dr-xr-xr-x. 10 root root 0 Mar 16 09:34 pids
dr-xr-xr-x. 2 root root 0 Mar 16 09:34 rdma
dr-xr-xr-x. 11 root root 0 Mar 16 09:34 systemd

```

The **/sys/fs/cgroup/** directory, also called the *root control group*, by default, contains controller-specific directories such as **cpuset**. In addition, there are some directories related to **systemd**.

Additional resources

- [Understanding control groups](#)

- [What are kernel resource controllers](#)
- **cgroups(7)**, **sysfs(5)** manual pages
- [cgroup-v2 enabled by default in RHEL 9](#)

35.4. SETTING CPU LIMITS TO APPLICATIONS USING CGROUPS-V1

Sometimes an application consumes a lot of CPU time, which may negatively impact the overall health of your environment. Use the **/sys/fs/** virtual file system to configure CPU limits to an application using *control groups version 1* (**cgroups-v1**).

Prerequisites

- You have root permissions.
- You have an application whose CPU consumption you want to restrict.
- You configured the system to mount **cgroups-v1** by default during system boot by the **systemd** system and service manager:

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --
args="systemd.unified_cgroup_hierarchy=0
systemd.legacy_systemd_cgroup_controller"
```

This adds the necessary kernel command-line parameters to the current boot entry.

Procedure

1. Identify the process ID (PID) of the application you want to restrict in CPU consumption:

```
# top
top - 11:34:09 up 11 min, 1 user, load average: 0.51, 0.27, 0.22
Tasks: 267 total, 3 running, 264 sleeping, 0 stopped, 0 zombie
%Cpu(s): 49.0 us, 3.3 sy, 0.0 ni, 47.5 id, 0.0 wa, 0.2 hi, 0.0 si, 0.0 st
MiB Mem : 1826.8 total, 303.4 free, 1046.8 used, 476.5 buff/cache
MiB Swap: 1536.0 total, 1396.0 free, 140.0 used. 616.4 avail Mem

  PID USER   PR NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6955 root    20  0 228440 1752 1472 R 99.3 0.1   0:32.71 sha1sum
 5760 jdoe    20  0 3603868 205188 64196 S  3.7 11.0   0:17.19 gnome-shell
 6448 jdoe    20  0 743648 30640 19488 S  0.7  1.6   0:02.73 gnome-terminal-
    1 root    20  0 245300 6568 4116 S  0.3  0.4   0:01.87 systemd
   505 root    20  0     0     0  0  I  0.3  0.0   0:00.75 kworker/u4:4-events_unbound
...
```

The example output of the **top** program reveals that **PID 6955** (illustrative application **sha1sum**) consumes a lot of CPU resources.

2. Create a sub-directory in the **cpu** resource controller directory:

```
# mkdir /sys/fs/cgroup/cpu/Example/
```

The directory above represents a control group, where you can place specific processes and apply certain CPU limits to the processes. At the same time, some **cgroups-v1** interface files and **cpu** controller-specific files will be created in the directory.

3. Optionally, inspect the newly created control group:

```
# ll /sys/fs/cgroup/cpu/Example/
-rw-r--r--. 1 root root 0 Mar 11 11:42 cgroup.clone_children
-rw-r--r--. 1 root root 0 Mar 11 11:42 cgroup.procs
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.stat
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_all
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu_sys
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu_user
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_sys
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_user
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.cfs_period_us
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.cfs_quota_us
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.rt_period_us
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.rt_runtime_us
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.shares
-r--r--r--. 1 root root 0 Mar 11 11:42 cpu.stat
-rw-r--r--. 1 root root 0 Mar 11 11:42 notify_on_release
-rw-r--r--. 1 root root 0 Mar 11 11:42 tasks
```

The example output shows files, such as **cpuacct.usage**, **cpu.cfs_period_us**, that represent specific configurations and/or limits, which can be set for processes in the **Example** control group. Notice that the respective file names are prefixed with the name of the control group controller to which they belong.

By default, the newly created control group inherits access to the system's entire CPU resources without a limit.

4. Configure CPU limits for the control group:

```
# echo "1000000" > /sys/fs/cgroup/cpu/Example/cpu.cfs_period_us
# echo "200000" > /sys/fs/cgroup/cpu/Example/cpu.cfs_quota_us
```

The **cpu.cfs_period_us** file represents a period of time in microseconds (μ s, represented here as "us") for how frequently a control group's access to CPU resources should be reallocated. The upper limit is 1 second and the lower limit is 1000 microseconds.

The **cpu.cfs_quota_us** file represents the total amount of time in microseconds for which all processes collectively in a control group can run during one period (as defined by **cpu.cfs_period_us**). As soon as processes in a control group, during a single period, use up all the time specified by the quota, they are throttled for the remainder of the period and not allowed to run until the next period. The lower limit is 1000 microseconds.

The example commands above set the CPU time limits so that all processes collectively in the **Example** control group will be able to run only for 0.2 seconds (defined by **cpu.cfs_quota_us**) out of every 1 second (defined by **cpu.cfs_period_us**).

5. Optionally, verify the limits:

```
# cat /sys/fs/cgroup/cpu/Example/cpu.cfs_period_us
```

```
/sys/fs/cgroup/cpu/Example/cpu.cfs_quota_us
1000000
200000
```

6. Add the application's PID to the **Example** control group:

```
# echo "6955" > /sys/fs/cgroup/cpu/Example/cgroup.procs
```

or

```
# echo "6955" > /sys/fs/cgroup/cpu/Example/tasks
```

The previous command ensures that a desired application becomes a member of the **Example** control group and hence does not exceed the CPU limits configured for the **Example** control group. The PID should represent an existing process in the system. The **PID 6955** here was assigned to process **sha1sum /dev/zero &**, used to illustrate the use-case of the **cpu** controller.

7. Verify that the application runs in the specified control group:

```
# cat /proc/6955/cgroup
12:cpuset:/
11:hugetlb:/
10:net_cls,net_prio:/
9:memory:/user.slice/user-1000.slice/user@1000.service
8:devices:/user.slice
7:blkio:/
6:freezer:/
5:rdma:/
4:pids:/user.slice/user-1000.slice/user@1000.service
3:perf_event:/
2:cpu,cpuacct:/Example
1:name=systemd:/user.slice/user-1000.slice/user@1000.service/gnome-terminal-
server.service
```

The example output above shows that the process of the desired application runs in the **Example** control group, which applies CPU limits to the application's process.

8. Identify the current CPU consumption of your throttled application:

```
# top
top - 12:28:42 up 1:06, 1 user, load average: 1.02, 1.02, 1.00
Tasks: 266 total, 6 running, 260 sleeping, 0 stopped, 0 zombie
%Cpu(s): 11.0 us, 1.2 sy, 0.0 ni, 87.5 id, 0.0 wa, 0.2 hi, 0.0 si, 0.2 st
MiB Mem : 1826.8 total, 287.1 free, 1054.4 used, 485.3 buff/cache
MiB Swap: 1536.0 total, 1396.7 free, 139.2 used. 608.3 avail Mem

  PID USER   PR NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6955 root    20  0 228440 1752 1472 R  20.6  0.1  47:11.43 sha1sum
 5760 jdoe    20  0 3604956 208832 65316 R   2.3 11.2   0:43.50 gnome-shell
 6448 jdoe    20  0 743836 31736 19488 S   0.7  1.7   0:08.25 gnome-terminal-
 505 root    20  0     0     0  0 0.3  0.0   0:03.39 kworker/u4:4-events_unbound
 4217 root    20  0  74192 1612 1320 S   0.3  0.1   0:01.19 spice-vdagentd
...
```

Notice that the CPU consumption of the **PID 6955** has decreased from 99% to 20%.



IMPORTANT

The **cgroups-v2** counterpart for **cpu.cfs_period_us** and **cpu.cfs_quota_us** is the **cpu.max** file. The **cpu.max** file is available through the **cpu** controller.

Additional resources

- [Understanding control groups](#)
- [What kernel resource controllers are](#)
- **cgroups(7)**, **sysfs(5)** manual pages

CHAPTER 36. ANALYZING SYSTEM PERFORMANCE WITH BPF COMPILER COLLECTION

As a system administrator, you can use the BPF Compiler Collection (BCC) library to create tools for analyzing the performance of your Linux operating system and gathering information, which could be difficult to obtain through other interfaces.

36.1. INSTALLING THE BCC-TOOLS PACKAGE

Install the **bcc-tools** package, which also installs the BPF Compiler Collection (BCC) library as a dependency.

Procedure

1. Install **bcc-tools**.

```
# dnf install bcc-tools
```

The BCC tools are installed in the **/usr/share/bcc/tools/** directory.

2. Optionally, inspect the tools:

```
# ll /usr/share/bcc/tools/
...
-rwxr-xr-x. 1 root root 4198 Dec 14 17:53 dcsnoop
-rwxr-xr-x. 1 root root 3931 Dec 14 17:53 dcstat
-rwxr-xr-x. 1 root root 20040 Dec 14 17:53 deadlock_detector
-rw-r--r--. 1 root root 7105 Dec 14 17:53 deadlock_detector.c
drwxr-xr-x. 3 root root 8192 Mar 11 10:28 doc
-rwxr-xr-x. 1 root root 7588 Dec 14 17:53 execsnoop
-rwxr-xr-x. 1 root root 6373 Dec 14 17:53 ext4dist
-rwxr-xr-x. 1 root root 10401 Dec 14 17:53 ext4slower
...
```

The **doc** directory in the listing above contains documentation for each tool.

36.2. USING SELECTED BCC-TOOLS FOR PERFORMANCE ANALYSES

Use certain pre-created programs from the BPF Compiler Collection (BCC) library to efficiently and securely analyze the system performance on the per-event basis. The set of pre-created programs in the BCC library can serve as examples for creation of additional programs.

Prerequisites

- [Installed bcc-tools package](#)
- Root permissions

Using execsnoop to examine the system processes

1. Run the **execsnoop** program in one terminal:

```
# /usr/share/bcc/tools/execsnoop
```

2. In another terminal run, for example:

```
$ ls /usr/share/bcc/tools/doc/
```

The above creates a short-lived process of the **ls** command.

3. The terminal running **execsnoop** shows the output similar to the following:

```
PCOMM PID  PPID  RET ARGS
ls  8382  8287   0 /usr/bin/ls --color=auto /usr/share/bcc/tools/doc/
...
```

The **execsnoop** program prints a line of output for each new process, which consumes system resources. It even detects processes of programs that run very shortly, such as **ls**, and most monitoring tools would not register them.

The **execsnoop** output displays the following fields:

- **PCOMM** - The parent process name. (**ls**)
- **PID** - The process ID. (**8382**)
- **PPID** - The parent process ID. (**8287**)
- **RET** - The return value of the **exec()** system call (**0**), which loads program code into new processes.
- **ARGS** - The location of the started program with arguments.

To see more details, examples, and options for **execsnoop**, refer to the **/usr/share/bcc/tools/doc/execsnoop_example.txt** file.

For more information about **exec()**, see **exec(3)** manual pages.

Using opensnoop to track what files a command opens

1. Run the **opensnoop** program in one terminal:

```
# /usr/share/bcc/tools/opensnoop -n uname
```

The above prints output for files, which are opened only by the process of the **uname** command.

2. In another terminal, enter:

```
$ uname
```

The command above opens certain files, which are captured in the next step.

3. The terminal running **opensnoop** shows the output similar to the following:

```
PID  COMM  FD ERR PATH
8596  uname  3  0  /etc/ld.so.cache
8596  uname  3  0  /lib64/libc.so.6
8596  uname  3  0  /usr/lib/locale/locale-archive
...
```


The **opensnoop** program watches the **open()** system call across the whole system, and prints a line of output for each file that **uname** tried to open along the way.

The **opensnoop** output displays the following fields:

- **PID** - The process ID. (**8596**)
- **COMM** - The process name. (**uname**)
- **FD** - The file descriptor - a value that **open()** returns to refer to the open file. (**3**)
- **ERR** - Any errors.
- **PATH** - The location of files that **open()** tried to open.
If a command tries to read a non-existent file, then the **FD** column returns **-1** and the **ERR** column prints a value corresponding to the relevant error. As a result, **opensnoop** can help you identify an application that does not behave properly.

To see more details, examples, and options for **opensnoop**, refer to the **/usr/share/bcc/tools/doc/opensnoop_example.txt** file.

For more information about **open()**, see **open(2)** manual pages.

Using biotop to examine the I/O operations on the disk

1. Run the **biotop** program in one terminal:

```
# /usr/share/bcc/tools/biotop 30
```

The command enables you to monitor the top processes, which perform I/O operations on the disk. The argument ensures that the command will produce a 30 second summary.



NOTE

When no argument provided, the output screen by default refreshes every 1 second.

2. In another terminal enter, for example :

```
# dd if=/dev/vda of=/dev/zero
```

The command above reads the content from the local hard disk device and writes the output to the **/dev/zero** file. This step generates certain I/O traffic to illustrate **biotop**.

3. The terminal running **biotop** shows the output similar to the following:

```
PID  COMM      D MAJ MIN DISK   I/O Kbytes  AVGms
9568 dd        R 252 0  vda    16294 14440636.0 3.69
48  kswapd0    W 252 0  vda     1763 120696.0 1.65
7571 gnome-shell R 252 0  vda      834 83612.0 0.33
1891 gnome-shell R 252 0  vda     1379 19792.0 0.15
7515 Xorg       R 252 0  vda      280 9940.0 0.28
7579 llvmpipe-1 R 252 0  vda      228 6928.0 0.19
9515 gnome-control-c R 252 0  vda       62 6444.0 0.43
8112 gnome-terminal- R 252 0  vda       67 2572.0 1.54
```

```

7807 gnome-software R 252 0 vda      31 2336.0  0.73
9578 awk             R 252 0 vda      17 2228.0  0.66
7578 llvmpipe-0      R 252 0 vda      156 2204.0 0.07
9581 pgrep           R 252 0 vda      58 1748.0  0.42
7531 InputThread     R 252 0 vda      30 1200.0  0.48
7504 gdbus          R 252 0 vda       3 1164.0  0.30
1983 llvmpipe-1      R 252 0 vda      39 724.0   0.08
1982 llvmpipe-0      R 252 0 vda      36 652.0   0.06
...

```

The **biotop** output displays the following fields:

- **PID** - The process ID. (**9568**)
- **COMM** - The process name. (**dd**)
- **DISK** - The disk performing the read operations. (**vda**)
- **I/O** - The number of read operations performed. (16294)
- **Kbytes** - The amount of Kbytes reached by the read operations. (14,440,636)
- **AVGms** - The average I/O time of read operations. (3.69)

To see more details, examples, and options for **biotop**, refer to the **/usr/share/bcc/tools/doc/biotop_example.txt** file.

For more information about **dd**, see **dd(1)** manual pages.

Using **xfsslower** to expose unexpectedly slow file system operations

1. Run the **xfsslower** program in one terminal:

```
# /usr/share/bcc/tools/xfsslower 1
```

The command above measures the time the XFS file system spends in performing read, write, open or sync (**fsync**) operations. The **1** argument ensures that the program shows only the operations that are slower than 1 ms.



NOTE

When no arguments provided, **xfsslower** by default displays operations slower than 10 ms.

2. In another terminal enter, for example, the following:

```
$ vim text
```

The command above creates a text file in the **vim** editor to initiate certain interaction with the XFS file system.

3. The terminal running **xfsslower** shows something similar upon saving the file from the previous step:

```
TIME    COMM      PID  T BYTES  OFF_KB  LAT(ms)  FILENAME
```

```

13:07:14 b'bash'      4754  R 256   0      7.11 b'vim'
13:07:14 b'vim'      4754  R 832   0      4.03 b'libgpm.so.2.1.0'
13:07:14 b'vim'      4754  R 32    20     1.04 b'libgpm.so.2.1.0'
13:07:14 b'vim'      4754  R 1982  0      2.30 b'vimrc'
13:07:14 b'vim'      4754  R 1393  0      2.52 b'getscriptPlugin.vim'
13:07:45 b'vim'      4754  S 0     0      6.71 b'text'
13:07:45 b'pool'     2588  R 16    0      5.58 b'text'
...

```

Each line above represents an operation in the file system, which took more time than a certain threshold. **xfsslower** is good at exposing possible file system problems, which can take form of unexpectedly slow operations.

The **xfsslower** output displays the following fields:

- **COMM** - The process name. (**b'bash'**)
- **T** - The operation type. (**R**)
 - **Read**
 - **Write**
 - **Sync**
- **OFF_KB** - The file offset in KB. (0)
- **FILENAME** - The file being read, written, or synced.

To see more details, examples, and options for **xfsslower**, refer to the **/usr/share/bcc/tools/doc/xfsslower_example.txt** file.

For more information about **fsync**, see **fsync(2)** manual pages.

CHAPTER 37. CONFIGURING AN OPERATING SYSTEM TO OPTIMIZE MEMORY ACCESS

You can configure the operating system to optimize memory access across workloads with the tools that are included in RHEL.

37.1. TOOLS FOR MONITORING AND DIAGNOSING SYSTEM MEMORY ISSUES

The following tools are available in Red Hat Enterprise Linux 9 for monitoring system performance and diagnosing performance problems related to system memory:

- **vmstat** tool, provided by the **procps-ng** package, displays reports of a system's processes, memory, paging, block I/O, traps, disks, and CPU activity. It provides an instantaneous report of the average of these events since the machine was last turned on, or since the previous report.
- **valgrind** framework provides instrumentation to user-space binaries. Install this tool, using the **dnf install valgrind** command. It includes a number of tools, that you can use to profile and analyze program performance, such as:
 - **memcheck** option is the default **valgrind** tool. It detects and reports on a number of memory errors that can be difficult to detect and diagnose, such as:
 - Memory access that should not occur
 - Undefined or uninitialized value use
 - Incorrectly freed heap memory
 - Pointer overlap
 - Memory leaks



NOTE

Memcheck can only report these errors, it cannot prevent them from occurring. However, **memcheck** logs an error message immediately before the error occurs.

- **cachegrind** option simulates application interaction with a system's cache hierarchy and branch predictor. It gathers statistics for the duration of application's execution and outputs a summary to the console.
- **massif** option measures the heap space used by a specified application. It measures both useful space and any additional space allocated for bookkeeping and alignment purposes.

Additional resources

- **vmstat(8)** and **valgrind(1)** man pages
- **/usr/share/doc/valgrind-version/valgrind_manual.pdf** file

37.2. OVERVIEW OF A SYSTEM'S MEMORY

The Linux Kernel is designed to maximize the utilization of a system's memory resources (RAM). Due to these design characteristics, and depending on the memory requirements of the workload, part of the system's memory is in use within the kernel on behalf of the workload, while a small part of the memory is free. This free memory is reserved for special system allocations, and for other low or high priority system services.

The rest of the system's memory is dedicated to the workload itself, and divided into the following two categories:

File memory

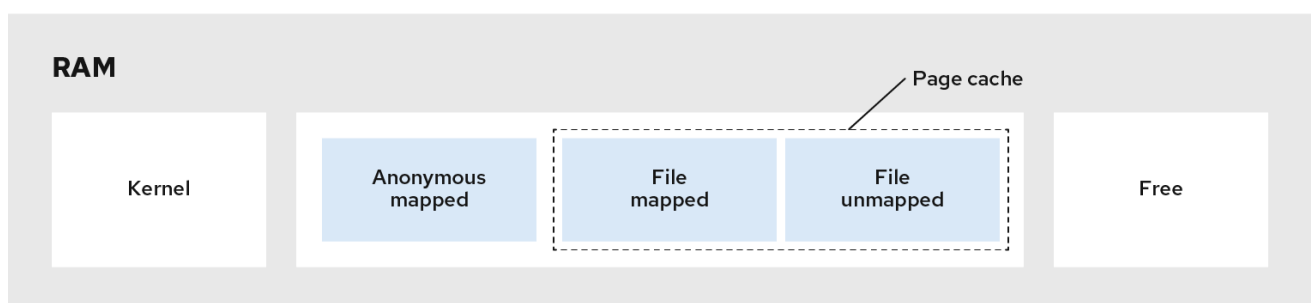
Pages added in this category represent parts of files in permanent storage. These pages, from the page cache, can be mapped or unmapped in an application's address spaces. You can use applications to map files into their address space using the **mmap** system calls, or to operate on files via the buffered I/O read or write system calls.

Buffered I/O system calls, as well as applications that map pages directly, can re-utilize unmapped pages. As a result, these pages are stored in the cache by the kernel, especially when the system is not running any memory intensive tasks, to avoid re-issuing costly I/O operations over the same set of pages.

Anonymous memory

Pages in this category are in use by a dynamically allocated process, or are not related to files in permanent storage. This set of pages back up the in-memory control structures of each task, such as the application stack and heap areas.

Figure 37.1. Memory usage patterns



133_RHEL_0121

37.3. VIRTUAL MEMORY PARAMETERS

The virtual memory parameters are listed in the **/proc/sys/vm** directory.

The following are the available virtual memory parameters:

vm.dirty_ratio

Is a percentage value. When this percentage of the total system memory is modified, the system begins writing the modifications to the disk with the **pdflush** operation. The default value is **20** percent.

vm.dirty_background_ratio

A percentage value. When this percentage of total system memory is modified, the system begins writing the modifications to the disk in the background. The default value is **10** percent.

vm.overcommit_memory

Defines the conditions that determine whether a large memory request is accepted or denied. The default value is **0**.

By default, the kernel performs checks if a virtual memory allocation request fits into the present amount of memory (total + swap) and rejects only large requests. Otherwise virtual memory allocations are granted, and this means they allow memory overcommitment.

Setting the **overcommit_memory** parameter's value:

- When this parameter is set to **1**, the kernel performs no memory overcommit handling. This increases the possibility of memory overload, but improves performance for memory-intensive tasks.
- When this parameter is set to **2**, the kernel denies requests for memory equal to or larger than the sum of the total available swap space and the percentage of physical RAM specified in the **overcommit_ratio**. This reduces the risk of overcommitting memory, but is recommended only for systems with swap areas larger than their physical memory.

vm.overcommit_ratio

Specifies the percentage of physical RAM considered when **overcommit_memory** is set to **2**. The default value is **50**.

vm.max_map_count

Defines the maximum number of memory map areas that a process can use. The default value is **65530**. Increase this value if your application needs more memory map areas.

vm.min_free_kbytes

Sets the size of the reserved free pages pool. It is also responsible for setting the **min_page**, **low_page**, and **high_page** thresholds that govern the behavior of the Linux kernel's page reclaim algorithms. It also specifies the minimum number of kilobytes to keep free across the system. This calculates a specific value for each low memory zone, each of which is assigned a number of reserved free pages in proportion to their size.

Setting the **vm.min_free_kbytes** parameter's value:

- Increasing the parameter value effectively reduces the application working set usable memory. Therefore, you might want to use it for only kernel-driven workloads, where driver buffers need to be allocated in atomic contexts.
- Decreasing the parameter value might render the kernel unable to service system requests, if memory becomes heavily contended in the system.



WARNING

Extreme values can be detrimental to the system's performance. Setting the **vm.min_free_kbytes** to an extremely low value prevents the system from reclaiming memory effectively, which can result in system crashes and failure to service interrupts or other kernel services. However, setting **vm.min_free_kbytes** too high considerably increases system reclaim activity, causing allocation latency due to a false direct reclaim state. This might cause the system to enter an out-of-memory state immediately.

The **vm.min_free_kbytes** parameter also sets a page reclaim watermark, called **min_pages**. This watermark is used as a factor when determining the two other memory watermarks, **low_pages**, and **high_pages**, that govern page reclaim algorithms.

/proc/PID/oom_adj

In the event that a system runs out of memory, and the **panic_on_oom** parameter is set to **0**, the **oom_killer** function kills processes, starting with the process that has the highest **oom_score**, until the system recovers.

The **oom_adj** parameter determines the **oom_score** of a process. This parameter is set per process identifier. A value of **-17** disables the **oom_killer** for that process. Other valid values range from **-16** to **15**.



NOTE

Processes created by an adjusted process inherit the **oom_score** of that process.

vm.swappiness

The swappiness value, ranging from **0** to **200**, controls the degree to which the system favors reclaiming memory from the anonymous memory pool, or the page cache memory pool.

Setting the **swappiness** parameter's value:

- Higher values favor file-mapped driven workloads while swapping out the less actively accessed processes' anonymous mapped memory of RAM. This is useful for file-servers or streaming applications that depend on data, from files in the storage, to reside on memory to reduce I/O latency for the service requests.
- Low values favor anonymous-mapped driven workloads while reclaiming the page cache (file mapped memory). This setting is useful for applications that do not depend heavily on the file system information, and heavily utilize dynamically allocated and private memory, such as mathematical and number crunching applications, and few hardware virtualization supervisors like QEMU.

The default value of the **vm.swappiness** parameter is **60**.



WARNING

Setting the **vm.swappiness** to **0** aggressively avoids swapping anonymous memory out to a disk, this increases the risk of processes being killed by the **oom_killer** function when under memory or I/O intensive workloads.

Additional resources

- **sysctl(8)** man page
- [Setting memory-related kernel parameters](#)

37.4. FILE SYSTEM PARAMETERS

The file system parameters are listed in the `/proc/sys/fs` directory. The following are the available file system parameters:

aio-max-nr

Defines the maximum allowed number of events in all active asynchronous input/output contexts. The default value is **65536**, and modifying this value does not pre-allocate or resize any kernel data structures.

file-max

Determines the maximum number of file handles for the entire system. The default value on Red Hat Enterprise Linux 9 is either **8192** or one tenth of the free memory pages available at the time the kernel starts, whichever is higher.

Raising this value can resolve errors caused by a lack of available file handles.

Additional resources

- **sysctl(8)** man page

37.5. KERNEL PARAMETERS

The default values for the kernel parameters are located in the `/proc/sys/kernel/` directory. These are set default values provided by the kernel or values specified by a user via **sysctl**.

The following are the available kernel parameters used to set up limits for the **msg*** and **shm*** System V IPC (**sysvipc**) system calls:

msgmax

Defines the maximum allowed size in bytes of any single message in a message queue. This value must not exceed the size of the queue (**msgmnb**). Use the **sysctl msgmax** command to determine the current **msgmax** value on your system.

msgmnb

Defines the maximum size in bytes of a single message queue. Use the **sysctl msgmnb** command to determine the current **msgmnb** value on your system.

msgmni

Defines the maximum number of message queue identifiers, and therefore the maximum number of queues. Use the **sysctl msgmni** command to determine the current **msgmni** value on your system.

shmall

Defines the total amount of shared memory **pages** that can be used on the system at one time. For example, a page is **4096** bytes on the AMD64 and Intel 64 architecture. Use the **sysctl shmall** command to determine the current **shmall** value on your system.

shmmax

Defines the maximum size in bytes of a single shared memory segment allowed by the kernel. Shared memory segments up to 1Gb are now supported in the kernel. Use the **sysctl shmmax** command to determine the current **shmmax** value on your system.

shmmni

Defines the system-wide maximum number of shared memory segments. The default value is **4096** on all systems.

Additional resources

Additional resources

- **sysvipc(7)** and **sysctl(8)** man pages

37.6. SETTING MEMORY-RELATED KERNEL PARAMETERS

Setting a parameter temporarily is useful for determining the effect the parameter has on a system. You can later set the parameter persistently when you are sure that the parameter value has the desired effect.

This procedure describes how to set a memory-related kernel parameter temporarily and persistently.

Procedure

- To temporarily set the memory-related kernel parameters, edit the respective files in the **/proc** file system or the **sysctl** tool.

For example, to temporarily set the **vm.overcommit_memory** parameter to **1**:

```
# echo 1 > /proc/sys/vm/overcommit_memory
# sysctl -w vm.overcommit_memory=1
```

- To persistently set the memory-related kernel parameter, edit the **/etc/sysctl.conf** file and reload the settings.

For example, to persistently set the **vm.overcommit_memory** parameter to **1**:

- Add the following content in the **/etc/sysctl.conf** file:

```
vm.overcommit_memory=1
```

- Reload the **sysctl** settings from the **/etc/sysctl.conf** file:

```
# sysctl -p
```

Additional resources

- **sysctl(8)** man page
- **proc(5)** man page

CHAPTER 38. CONFIGURING HUGE PAGES

Physical memory is managed in fixed-size chunks called pages. On the x86_64 architecture, supported by Red Hat Enterprise Linux 9, the default size of a memory page is **4 KB**. This default page size has proved to be suitable for general-purpose operating systems, such as Red Hat Enterprise Linux, which supports many different kinds of workloads.

However, specific applications can benefit from using larger page sizes in certain cases. For example, an application that works with a large and relatively fixed data set of hundreds of megabytes or even dozens of gigabytes can have performance issues when using **4 KB** pages. Such data sets can require a huge amount of **4 KB** pages, which can lead to overhead in the operating system and the CPU.

This section provides information about huge pages available in RHEL 9 and how you can configure them.

38.1. AVAILABLE HUGE PAGE FEATURES

With Red Hat Enterprise Linux 9, you can use huge pages for applications that work with big data sets, and improve the performance of such applications.

The following are the huge page methods, which are supported in RHEL 9:

HugeTLB pages

HugeTLB pages are also called static huge pages. There are two ways of reserving HugeTLB pages:

- **At boot time:** It increases the possibility of success because the memory has not yet been significantly fragmented. However, on NUMA machines, the number of pages is automatically split among the NUMA nodes.

For more information about parameters that influence HugeTLB page behavior at boot time, see [Parameters for reserving HugeTLB pages at boot time](#) and how to use these parameters to configure HugeTLB pages at boot time, see [Configuring HugeTLB at boot time](#).

- **At run time:** It allows you to reserve the huge pages per NUMA node. If the run-time reservation is done as early as possible in the boot process, the probability of memory fragmentation is lower.

For more information about parameters that influence HugeTLB page behavior at run time, see [Parameters for reserving HugeTLB pages at run time](#) and how to use these parameters to configure HugeTLB pages at run time, see [Configuring HugeTLB at run time](#).

Transparent HugePages (THP)

With THP, the kernel automatically assigns huge pages to processes, and therefore there is no need to manually reserve the static huge pages. The following are the two modes of operation in THP:

- **system-wide:** Here, the kernel tries to assign huge pages to a process whenever it is possible to allocate the huge pages and the process is using a large contiguous virtual memory area.
- **per-process:** Here, the kernel only assigns huge pages to the memory areas of individual processes which you can specify using the **madvise()** system call.



NOTE

The THP feature only supports **2 MB** pages.

For more information about parameters that influence HugeTLB page behavior at boot time, see [Enabling transparent hugepages](#) and [Disabling transparent hugepages](#).

38.2. PARAMETERS FOR RESERVING HUGETLB PAGES AT BOOT TIME

Use the following parameters to influence HugeTLB page behavior at boot time.

For more information on how to use these parameters to configure HugeTLB pages at boot time, see [Configuring HugeTLB at boot time](#).

Table 38.1. Parameters used to configure HugeTLB pages at boot time

Parameter	Description	Default value
hugepages	<p>Defines the number of persistent huge pages configured in the kernel at boot time.</p> <p>In a NUMA system, huge pages, that have this parameter defined, are divided equally between nodes.</p> <p>You can assign huge pages to specific nodes at runtime by changing the value of the nodes in the <code>/sys/devices/system/node/node_id/hugepages/hugepages-size/nr_hugepages</code> file.</p>	<p>The default value is 0.</p> <p>To update this value at boot, change the value of this parameter in the <code>/proc/sys/vm/nr_hugepages</code> file.</p>
hugepagesz	Defines the size of persistent huge pages configured in the kernel at boot time.	Valid values are 2 MB and 1 GB . The default value is 2 MB .
default_hugepagesz	Defines the default size of persistent huge pages configured in the kernel at boot time.	Valid values are 2 MB and 1 GB . The default value is 2 MB .

38.3. CONFIGURING HUGETLB AT BOOT TIME

The page size, which the HugeTLB subsystem supports, depends on the architecture. The x86_64 architecture supports **2 MB** huge pages and **1 GB** gigantic pages.

This procedure describes how to reserve a **1 GB** page at boot time.

Procedure

1. To create a HugeTLB pool for **1 GB** pages, enable the **default_hugepagesz=1G** and **hugepagesz=1G** kernel options:

```
# grubby --update-kernel=ALL --args="default_hugepagesz=1G hugepagesz=1G"
```

2. Create a new file called **hugetlb-gigantic-pages.service** in the `/usr/lib/systemd/system/` directory and add the following content:

```
[Unit]
Description=HugeTLB Gigantic Pages Reservation
DefaultDependencies=no
Before=dev-hugepages.mount
ConditionPathExists=/sys/devices/system/node
ConditionKernelCommandLine=hugepagesz=1G

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/usr/lib/systemd/hugetlb-reserve-pages.sh

[Install]
WantedBy=sysinit.target
```

3. Create a new file called **hugetlb-reserve-pages.sh** in the `/usr/lib/systemd/` directory and add the following content:

While adding the following content, replace *number_of_pages* with the number of 1GB pages you want to reserve, and *node* with the name of the node on which to reserve these pages.

```
#!/bin/sh

nodes_path=/sys/devices/system/node/
if [ ! -d $nodes_path ]; then
    echo "ERROR: $nodes_path does not exist"
    exit 1
fi

reserve_pages()
{
    echo $1 > $nodes_path/$2/hugepages/hugepages-1048576kB/nr_hugepages
}

reserve_pages number_of_pages node
```

For example, to reserve two **1 GB** pages on *node0* and one 1GB page on *node1*, replace the *number_of_pages* with 2 for *node0* and 1 for *node1*:

```
reserve_pages 2 node0
reserve_pages 1 node1
```

4. Create an executable script:

```
# chmod +x /usr/lib/systemd/hugetlb-reserve-pages.sh
```

5. Enable early boot reservation:

```
# systemctl enable hugetlb-gigantic-pages
```

**NOTE**

- You can try reserving more **1 GB** pages at runtime by writing to **nr_hugepages** at any time. However, to prevent failures due to memory fragmentation, reserve **1 GB** pages early during the boot process.
- Reserving static huge pages can effectively reduce the amount of memory available to the system, and prevents it from properly utilizing its full memory capacity. Although a properly sized pool of reserved huge pages can be beneficial to applications that utilize it, an oversized or unused pool of reserved huge pages will eventually be detrimental to overall system performance. When setting a reserved huge page pool, ensure that the system can properly utilize its full memory capacity.

Additional resources

- **systemd.service(5)** man page
- **/usr/share/doc/kernel-doc-kernel_version/Documentation/vm/hugetlbpage.txt** file

38.4. PARAMETERS FOR RESERVING HUGETLB PAGES AT RUN TIME

Use the following parameters to influence HugeTLB page behavior at run time.

For more information about how to use these parameters to configure HugeTLB pages at run time, see [Configuring HugeTLB at run time](#).

Table 38.2. Parameters used to configure HugeTLB pages at run time

Parameter	Description	File name
nr_hugepages	Defines the number of huge pages of a specified size assigned to a specified NUMA node.	/sys/devices/system/node/node_id/hugepages/hugepages-size/nr_hugepages
nr_overcommit_hugepages	<p>Defines the maximum number of additional huge pages that can be created and used by the system through overcommitting memory.</p> <p>Writing any non-zero value into this file indicates that the system obtains that number of huge pages from the kernel's normal page pool if the persistent huge page pool is exhausted. As these surplus huge pages become unused, they are then freed and returned to the kernel's normal page pool.</p>	/proc/sys/vm/nr_overcommit_hugepages

38.5. CONFIGURING HUGETLB AT RUN TIME

This procedure describes how to add *20 2048 kB* huge pages to *node2*.

To reserve pages based on your requirements, replace:

- 20 with the number of huge pages you wish to reserve,
- 2048kB with the size of the huge pages,
- node2 with the node on which you wish to reserve the pages.

Procedure

1. Display the memory statistics:

```
# numastat -cm | egrep 'Node|Huge'
      Node 0 Node 1 Node 2 Node 3 Total add
AnonHugePages      0   2   0   8   10
HugePages_Total    0   0   0   0   0
HugePages_Free     0   0   0   0   0
HugePages_Surp     0   0   0   0   0
```

2. Add the number of huge pages of a specified size to the node:

```
# echo 20 > /sys/devices/system/node/node2/hugepages/hugepages-2048kB/nr_hugepages
```

Verification steps

- Ensure that the number of huge pages are added:

```
# numastat -cm | egrep 'Node|Huge'
      Node 0 Node 1 Node 2 Node 3 Total
AnonHugePages      0   2   0   8   10
HugePages_Total    0   0  40   0  40
HugePages_Free     0   0  40   0  40
HugePages_Surp     0   0   0   0   0
```

Additional resources

- **numastat(8)** man page

38.6. ENABLING TRANSPARENT HUGEPAGES

THP is enabled by default in Red Hat Enterprise Linux 9. However, you can enable or disable THP.

This procedure describes how to enable THP.

Procedure

1. Check the current status of THP:

```
# cat /sys/kernel/mm/transparent_hugepage/enabled
```

2. Enable THP:

```
# echo always > /sys/kernel/mm/transparent_hugepage/enabled
```

3. To prevent applications from allocating more memory resources than necessary, disable the system-wide transparent huge pages and only enable them for the applications that explicitly request it through the **madvise**:

```
# echo madvise > /sys/kernel/mm/transparent_hugepage/enabled
```

NOTE

Sometimes, providing low latency to short-lived allocations has higher priority than immediately achieving the best performance with long-lived allocations. In such cases, you can disable direct compaction while leaving THP enabled.

Direct compaction is a synchronous memory compaction during the huge page allocation. Disabling direct compaction provides no guarantee of saving memory, but can decrease the risk of higher latencies during frequent page faults. Note that if the workload benefits significantly from THP, the performance decreases. Disable direct compaction:

```
# echo madvise > /sys/kernel/mm/transparent_hugepage/defrag
```

Additional resources

- **madvise(2)** man page
- [Disabling transparent hugepages](#).

38.7. DISABLING TRANSPARENT HUGE PAGES

THP is enabled by default in Red Hat Enterprise Linux 9. However, you can enable or disable THP.

This procedure describes how to disable THP.

Procedure

1. Check the current status of THP:

```
# cat /sys/kernel/mm/transparent_hugepage/enabled
```

2. Disable THP:

```
# echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

38.8. IMPACT OF PAGE SIZE ON TRANSLATION LOOKASIDE BUFFER SIZE

Reading address mappings from the page table is time-consuming and resource-expensive, so CPUs are built with a cache for recently-used addresses, called the Translation Lookaside Buffer (TLB). However, the default TLB can only cache a certain number of address mappings.

If a requested address mapping is not in the TLB, called a TLB miss, the system still needs to read the page table to determine the physical to virtual address mapping. Because of the relationship between application memory requirements and the size of pages used to cache address mappings, applications

with large memory requirements are more likely to suffer performance degradation from TLB misses than applications with minimal memory requirements. It is therefore important to avoid TLB misses wherever possible.

Both HugeTLB and Transparent Huge Page features allow applications to use pages larger than **4 KB**. This allows addresses stored in the TLB to reference more memory, which reduces TLB misses and improves application performance.

CHAPTER 39. GETTING STARTED WITH SYSTEMTAP

As a system administrator, you can use SystemTap to identify underlying causes of a bug or performance problem on a running Linux system.

As an application developer, you can use SystemTap to monitor in fine detail how your application behaves within the Linux system.

39.1. THE PURPOSE OF SYSTEMTAP

SystemTap is a tracing and probing tool that you can use to study and monitor the activities of your operating system (particularly, the kernel) in fine detail. SystemTap provides information similar to the output of tools such as **netstat**, **ps**, **top**, and **iostat**. However, SystemTap provides more filtering and analysis options for collected information. In SystemTap scripts, you specify the information that SystemTap gathers.

SystemTap aims to supplement the existing suite of Linux monitoring tools by providing users with the infrastructure to track kernel activity and combining this capability with two attributes:

Flexibility

the SystemTap framework enables you to develop simple scripts for investigating and monitoring a wide variety of kernel functions, system calls, and other events that occur in kernel space. With this, SystemTap is not so much a tool as it is a system that allows you to develop your own kernel-specific forensic and monitoring tools.

Ease-of-Use

SystemTap enables you to monitor kernel activity without having to recompile the kernel or reboot the system.

39.2. INSTALLING SYSTEMTAP

To begin using SystemTap, install the required packages. To use SystemTap on more than one kernel where a system has multiple kernels installed, install the corresponding required kernel packages for *each* kernel version.

Prerequisites

- You have enabled debug repositories as described in [Enabling debug and source repositories](#).

Procedure

1. Install the required SystemTap packages:

```
# dnf install systemtap
```

2. Install the required kernel packages:

- a. Using **stap-prep**:

```
# stap-prep
```

- b. If **stap-prep** does not work, install the required kernel packages manually:

```
# dnf install kernel-debuginfo-$(uname -r) kernel-debuginfo-common-$(uname -i)-
$(uname -r) kernel-devel-$(uname -r)
```

\$(uname -i) is automatically replaced with the hardware platform of your system and **\$(uname -r)** is automatically replaced with the version of your running kernel.

Verification steps

- If the kernel to be probed with SystemTap is currently in use, test if your installation was successful:

```
# stap -v -e 'probe kernel.function("vfs_read") {printf("read performed\n"); exit();}'
```

A successful SystemTap deployment results in an output similar to the following:

```
Pass 1: parsed user script and 45 library script(s) in 340usr/0sys/358real ms.
Pass 2: analyzed script: 1 probe(s), 1 function(s), 0 embed(s), 0 global(s) in
290usr/260sys/568real ms.
Pass 3: translated to C into
"/tmp/stapiArgLX/stap_e5886fa50499994e6a87aacdc43cd392_399.c" in
490usr/430sys/938real ms.
Pass 4: compiled C into "stap_e5886fa50499994e6a87aacdc43cd392_399.ko" in
3310usr/430sys/3714real ms.
Pass 5: starting run. ❶
read performed ❷
Pass 5: run completed in 10usr/40sys/73real ms. ❸
```

The last three lines of output (beginning with **Pass 5**) indicate that:

- ❶ SystemTap successfully created the instrumentation to probe the kernel and ran the instrumentation.
- ❷ SystemTap detected the specified event (in this case, A VFS read).
- ❸ SystemTap executed a valid handler (printed text and then closed it with no errors).

39.3. PRIVILEGES TO RUN SYSTEMTAP

Running SystemTap scripts requires elevated system privileges but, in some instances, non-privileged users might need to run SystemTap instrumentation on their machine.

To allow users to run SystemTap without root access, add users to **both** of these user groups:

stapdev

Members of this group can use **stap** to run SystemTap scripts, or **staprun** to run SystemTap instrumentation modules.

Running **stap** involves compiling SystemTap scripts into kernel modules and loading them into the kernel. This requires elevated privileges to the system, which are granted to **stapdev** members. Unfortunately, such privileges also grant effective root access to **stapdev** members. As such, only grant **stapdev** group membership to users who can be trusted with root access.

stapusr

Members of this group can only use **staprun** to run SystemTap instrumentation modules. In addition, they can only run those modules from the **/lib/modules/kernel_version/systemtap/** directory. This directory must be owned only by the root user, and must only be writable by the root user.

39.4. RUNNING SYSTEMTAP SCRIPTS

You can run SystemTap scripts from standard input or from a file.

Sample scripts that are distributed with the installation of SystemTap can be found in the **/usr/share/systemtap/examples** directory.

Prerequisites

1. SystemTap and the associated required kernel packages are installed as described in [Installing Systemtap](#).
2. To run SystemTap scripts as a normal user, add the user to the SystemTap groups:

```
# usermod --append --groups
stapdev,stapusr user-name
```

Procedure

- Run the SystemTap script:

- From standard input:

```
# echo "probe timer.s(1) {exit()}" | stap -
```

This command instructs **stap** to run the script passed by **echo** to standard input. To add **stap** options, insert them before the **-** character. For example, to make the results from this command more verbose, the command is:

```
# echo "probe timer.s(1) {exit()}" | stap -v -
```

- From a file:

```
# stap file_name
```

CHAPTER 40. CROSS-INSTRUMENTATION OF SYSTEMTAP

Cross-instrumentation of SystemTap is creating SystemTap instrumentation modules from a SystemTap script on one system to be used on another system that does not have SystemTap fully deployed.

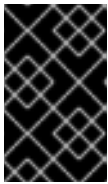
40.1. SYSTEMTAP CROSS-INSTRUMENTATION

When you run a SystemTap script, a kernel module is built out of that script. SystemTap then loads the module into the kernel.

Normally, SystemTap scripts can run only on systems where SystemTap is deployed. To run SystemTap on ten systems, SystemTap needs to be deployed on all those systems. In some cases, this might be neither feasible nor desired. For example, corporate policy might prohibit you from installing packages that provide compilers or debug information about specific machines, which will prevent the deployment of SystemTap.

To work around this, use *cross-instrumentation*. Cross-instrumentation is the process of generating SystemTap instrumentation modules from a SystemTap script on one system to be used on another system. This process offers the following benefits:

- The kernel information packages for various machines can be installed on a single host machine.



IMPORTANT

Kernel packaging bugs may prevent the installation. In such cases, the **kernel-debuginfo** and **kernel-devel** packages for the *host system* and *target system* must match. If a bug occurs, report the bug at <https://bugzilla.redhat.com/>.

- Each *target machine* needs only one package to be installed to use the generated SystemTap instrumentation module: **systemtap-runtime**.



IMPORTANT

The *host system* must be the same architecture and running the same distribution of Linux as the *target system* in order for the built *instrumentation module* to work.



TERMINOLOGY

instrumentation module

The kernel module built from a SystemTap script; the SystemTap module is built on the *host system*, and will be loaded on the *target kernel* of the *target system*.

host system

The system on which the instrumentation modules (from SystemTap scripts) are compiled, to be loaded on *target systems*.

target system

The system in which the *instrumentation module* is being built (from SystemTap scripts).

target kernel

The kernel of the *target system*. This is the kernel that loads and runs the *instrumentation module*.

40.2. INITIALIZING CROSS-INSTRUMENTATION OF SYSTEMTAP

Initialize cross-instrumentation of SystemTap to build SystemTap instrumentation modules from a SystemTap script on one system and use them on another system that does not have SystemTap fully deployed.

Prerequisites

- SystemTap is installed on the *host system* as described in [Installing Systemtap](#).
- The **systemtap-runtime** package is installed on each *target system*:

```
# dnf install systemtap-runtime
```
- Both the *host system* and *target system* are the same architecture.
- Both the *host system* and *target system* are running the same major version of Red Hat Enterprise Linux (such as Red Hat Enterprise Linux 9).



IMPORTANT

Kernel packaging bugs may prevent multiple **kernel-debuginfo** and **kernel-devel** packages from being installed on one system. In such cases, the minor version for the *host system* and *target system* must match. If a bug occurs, report it at <https://bugzilla.redhat.com/>.

Procedure

1. Determine the kernel running on each *target system*:

```
$ uname -r
```

Repeat this step for each *target system*.

2. On the *host system*, install the *target kernel* and related packages for each *target system* by the method described in [Installing Systemtap](#).

3. Build an instrumentation module on the *host system*, copy this module to and run this module on on the *target system* either:

- a. Using remote implementation:

```
# stap --remote target_system script
```

This command remotely implements the specified script on the *target system*. You must ensure an SSH connection can be made to the *target system* from the *host system* for this to be successful.

- b. Manually:

- i. Build the instrumentation module on the *host system*:

```
# stap -r kernel_version script -m module_name -p 4
```

Here, *kernel_version* refers to the version of the *target kernel* determined in step 1, *script* refers to the script to be converted into an *instrumentation module*, and *module_name* is the desired name of the *instrumentation module*. The **-p4** option tells SystemTap to not load and run the compiled module.

- ii. Once the *instrumentation module* is compiled, copy it to the target system and load it using the following command:

```
# staprun module_name.ko
```

CHAPTER 41. MONITORING NETWORK ACTIVITY WITH SYSTEMTAP

You can use helpful example SystemTap scripts available in the `/usr/share/systemtap/testsuite/systemtap.examples/` directory, upon installing the **systemtap-testsuite** package, to monitor and investigate the network activity of your system.

41.1. PROFILING NETWORK ACTIVITY WITH SYSTEMTAP

You can use the **nettop.stp** example SystemTap script to profile network activity. The script tracks which processes are generating network traffic on the system, and provides the following information about each process:

PID

The ID of the listed process.

UID

User ID. A user ID of 0 refers to the root user.

DEV

Which ethernet device the process used to send or receive data (for example, eth0, eth1).

XMIT_PK

The number of packets transmitted by the process.

RECV_PK

The number of packets received by the process.

XMIT_KB

The amount of data sent by the process, in kilobytes.

RECV_KB

The amount of data received by the service, in kilobytes.

Prerequisites

- You have installed SystemTap as described in [Installing SystemTap](#).

Procedure

- Run the **nettop.stp** script:

```
# stap --example nettop.stp
```

The **nettop.stp** script provides network profile sampling every 5 seconds.

Output of the **nettop.stp** script looks similar to the following:

```
[...]
PID UID DEV XMIT_PK RECV_PK XMIT_KB RECV_KB COMMAND
0 0 eth0 0 5 0 0 swapper
11178 0 eth0 2 0 0 0 synergyc
PID UID DEV XMIT_PK RECV_PK XMIT_KB RECV_KB COMMAND
2886 4 eth0 79 0 5 0 cups-polld
11362 0 eth0 0 61 0 5 firefox
```

```

    0 0 eth0      3 32 0 3 swapper
2886 4 lo        4 4 0 0 cups-polld
11178 0 eth0     3 0 0 0 synergyc
  PID UID DEV  XMIT_PK RECV_PK XMIT_KB RECV_KB COMMAND
    0 0 eth0    0 6 0 0 swapper
2886 4 lo      2 2 0 0 cups-polld
11178 0 eth0   3 0 0 0 synergyc
3611 0 eth0    0 1 0 0 Xorg
  PID UID DEV  XMIT_PK RECV_PK XMIT_KB RECV_KB COMMAND
    0 0 eth0    3 42 0 2 swapper
11178 0 eth0   43 1 3 0 synergyc
11362 0 eth0    0 7 0 0 firefox
3897 0 eth0    0 1 0 0 multiload-apple

```

41.2. TRACING FUNCTIONS CALLED IN NETWORK SOCKET CODE WITH SYSTEMTAP

You can use the **socket-trace.stp** example SystemTap script to trace functions called from the kernel's `net/socket.c` file. This helps you identify, in finer detail, how each process interacts with the network at the kernel level.

Prerequisites

- You have installed SystemTap as described in [Installing SystemTap](#).

Procedure

- Run the **socket-trace.stp** script:

```
# stap --example socket-trace.stp
```

A 3-second excerpt of the output of the **socket-trace.stp** script looks similar to the following:

```

[...]
0 Xorg(3611): -> sock_poll
3 Xorg(3611): <- sock_poll
0 Xorg(3611): -> sock_poll
3 Xorg(3611): <- sock_poll
0 gnome-terminal(11106): -> sock_poll
5 gnome-terminal(11106): <- sock_poll
0 scim-bridge(3883): -> sock_poll
3 scim-bridge(3883): <- sock_poll
0 scim-bridge(3883): -> sys_socketcall
4 scim-bridge(3883): -> sys_recv
8 scim-bridge(3883): -> sys_recvfrom
12 scim-bridge(3883):-> sock_from_file
16 scim-bridge(3883):<- sock_from_file
20 scim-bridge(3883):-> sock_recvmsg
24 scim-bridge(3883):<- sock_recvmsg
28 scim-bridge(3883): <- sys_recvfrom
31 scim-bridge(3883): <- sys_recv
35 scim-bridge(3883): <- sys_socketcall
[...]
```


41.3. MONITORING NETWORK PACKET DROPS WITH SYSTEMTAP

The network stack in Linux can discard packets for various reasons. Some Linux kernels include a tracepoint, `kernel.trace("kfree_skb")`, which tracks where packets are discarded.

The **dropwatch.stp** SystemTap script uses `kernel.trace("kfree_skb")` to trace packet discards; the script summarizes what locations discard packets in every 5-second interval.

Prerequisites

- You have installed SystemTap as described in [Installing SystemTap](#).

Procedure

- Run the **dropwatch.stp** script:

```
# stap --example dropwatch.stp
```

Running the **dropwatch.stp** script for 15 seconds results in output similar to the following:

```
Monitoring for dropped packets
51 packets dropped at location 0xffffffff8024cd0f
2 packets dropped at location 0xffffffff8044b472
51 packets dropped at location 0xffffffff8024cd0f
1 packets dropped at location 0xffffffff8044b472
97 packets dropped at location 0xffffffff8024cd0f
1 packets dropped at location 0xffffffff8044b472
Stopping dropped packet monitor
```

NOTE

To make the location of packet drops more meaningful, see the **/boot/System.map-\$(uname -r)** file. This file lists the starting addresses for each function, enabling you to map the addresses in the output of the **dropwatch.stp** script to a specific function name. Given the following snippet of the **/boot/System.map-\$(uname -r)** file, the address **0xffffffff8024cd0f** maps to the function **unix_stream_recvmsg** and the address **0xffffffff8044b472** maps to the function **arp_rcv**:

```
[...]
ffffff8024c5cd T unlock_new_inode
ffffff8024c5da t unix_stream_sendmsg
ffffff8024c920 t unix_stream_recvmsg
ffffff8024cea1 t udp_v4_lookup_longway
[...]
ffffff8044addc t arp_process
ffffff8044b360 t arp_rcv
ffffff8044b487 t parp_redo
ffffff8044b48c t arp_solicit
[...]
```

CHAPTER 42. PROFILING KERNEL ACTIVITY WITH SYSTEMTAP

You can profile the kernel activity by monitoring function calls with the following scripts.

42.1. COUNTING FUNCTION CALLS WITH SYSTEMTAP

You can use the **functioncallcount.stp** SystemTap script to count specific kernel function calls. You can also use this script to target multiple kernel functions.

Prerequisites

- You have installed SystemTap as described in [Installing Systemtap](#).

Procedure

- Run the **functioncallcount.stp** script:

```
# stap --example functioncallcount.stp 'argument'
```

This script takes the targeted kernel function as an argument. You can use the argument wildcards to target multiple kernel functions up to a certain extent.

The output of the script, in alphabetical order, contains the names of the functions called and how many times it was called during the sample time.

Consider the following example:

```
# stap -w -v --example functioncallcount.stp "*"@mm*.c" -c /bin/true
```

where:

- **-w** : Suppresses warnings.
- **-v** : Makes the output of starting kernel visible.
- **-c command** : Tells SystemTap to count function calls during the execution of a command, in this example being **/bin/true**.

The output should look similar to the following:

```
[...]
__vma_link 97
__vma_link_file 66
__vma_link_list 97
__vma_link_rb 97
__xchg 103
add_page_to_active_list 102
add_page_to_inactive_list 19
add_to_page_cache 19
add_to_page_cache_lru 7
all_vm_events 6
alloc_pages_node 4630
alloc_slabmgmt 67
```

```
anon_vma_alloc 62
anon_vma_free 62
anon_vma_lock 66
anon_vma_prepare 98
anon_vma_unlink 97
anon_vma_unlock 66
arch_get_unmapped_area_topdown 94
arch_get_unmapped_exec_area 3
arch_unmap_area_topdown 97
atomic_add 2
atomic_add_negative 97
atomic_dec_and_test 5153
atomic_inc 470
atomic_inc_and_test 1
[...]
```

42.2. TRACING FUNCTION CALLS WITH SYSTEMTAP

You can use the **para-callgraph.stp** SystemTap script to trace function calls and function returns.

Prerequisites

- You have installed SystemTap as described in [Installing Systemtap](#).

Procedure

- Run the **para-callgraph.stp** script.

```
# stap --example para-callgraph.stp 'argument1' 'argument2'
```

The script **para-callgraph.stp** takes two command-line arguments:

1. The name of the function(s) whose entry/exit you'd like to trace.
2. An optional trigger function, which enables or disables tracing on a per-thread basis. Tracing in each thread will continue as long as the trigger function has not exited yet.

Consider the following example:

```
# stap -wv --example para-callgraph.stp 'kernel.function("*@fs/proc.c*")' 'kernel.function("vfs_read")' -
c "cat /proc/sys/vm/* || true"
```

where:

- **-w** : Suppresses warnings.
- **-v** : Makes the output of starting kernel visible.
- **-c command** : Tells SystemTap to count function calls during the execution of a command, in this example being **/bin/true**.

The output should look similar to the following:

```
[...]
```

```

267 gnome-terminal(2921): <-do_sync_read return=0xffffffffffff5
269 gnome-terminal(2921):<-vfs_read return=0xffffffffffff5
0 gnome-terminal(2921):->fput file=0xffff880111eebbc0
2 gnome-terminal(2921):<-fput
0 gnome-terminal(2921):->fget_light fd=0x3 fput_needed=0xffff88010544df54
3 gnome-terminal(2921):<-fget_light return=0xffff8801116ce980
0 gnome-terminal(2921):->vfs_read file=0xffff8801116ce980 buf=0xc86504 count=0x1000
pos=0xffff88010544df48
4 gnome-terminal(2921): ->rw_verify_area read_write=0x0 file=0xffff8801116ce980
ppos=0xffff88010544df48 count=0x1000
7 gnome-terminal(2921): <-rw_verify_area return=0x1000
12 gnome-terminal(2921): ->do_sync_read filp=0xffff8801116ce980 buf=0xc86504 len=0x1000
ppos=0xffff88010544df48
15 gnome-terminal(2921): <-do_sync_read return=0xffffffffffff5
18 gnome-terminal(2921):<-vfs_read return=0xffffffffffff5
0 gnome-terminal(2921):->fput file=0xffff8801116ce980

```

42.3. DETERMINING TIME SPENT IN KERNEL AND USER SPACE WITH SYSTEMTAP

You can use the **thread-times.stp** SystemTap script to determine the amount of time any given thread is spending in either the kernel or user-space.

Prerequisites

- You have installed SystemTap as described in [Installing Systemtap](#).

Procedure

- Run the **thread-times.stp** script:

```
# stap --example thread-times.stp
```

This script will display the top 20 processes taking up CPU time during a 5-second period, along with the total number of CPU ticks made during the sample. The output of this script also notes the percentage of CPU time each process used, as well as whether that time was spent in kernel space or user space.

```

tid  %user %kernel (of 20002 ticks)
0    0.00% 87.88%
32169 5.24% 0.03%
9815  3.33% 0.36%
9859  0.95% 0.00%
3611  0.56% 0.12%
9861  0.62% 0.01%
11106 0.37% 0.02%
32167 0.08% 0.08%
3897  0.01% 0.08%
3800  0.03% 0.00%
2886  0.02% 0.00%
3243  0.00% 0.01%
3862  0.01% 0.00%
3782  0.00% 0.00%
21767 0.00% 0.00%

```

```
2522 0.00% 0.00%
3883 0.00% 0.00%
3775 0.00% 0.00%
3943 0.00% 0.00%
3873 0.00% 0.00%
```

42.4. MONITORING POLLING APPLICATIONS WITH SYSTEMTAP

You can use **timeout.stp** SystemTap script to identify and monitor which applications are polling. Doing so allows you to track unnecessary or excessive polling, which helps you pinpoint areas for improvement in terms of CPU usage and power savings.

Prerequisites

- You have installed SystemTap as described in [Installing Systemtap](#).

Procedure

- Run the **timeout.stp** script:

```
# stap --example timeout.stp
```

This script will track how many times each application uses the following system calls over time:

- **poll**
- **select**
- **epoll**
- **itimer**
- **futex**
- **nanosleep**
- **signal**

In this example output you can see which process used which system call and how many times.

```
uid | poll select  epoll itimer  futex nanosle  signal| process
28937 | 148793  0    0  4727 37288  0    0| firefox
22945 |  0 56949  0    1    0    0    0| scim-bridge
0 |  0    0    0 36414  0    0    0| swapper
4275 | 23140  0    0    1    0    0    0| mixer_applet2
4191 |  0 14405  0    0    0    0    0| scim-launcher
22941 | 7908  1    0    62    0    0    0| gnome-terminal
4261 |  0    0    0    2    0 7622  0| escd
3695 |  0    0    0    0    0 7622  0| gdm-binary
3483 |  0 7206  0    0    0    0    0| dhcdbd
4189 | 6916  0    0    2    0    0    0| scim-panel-gtk
1863 | 5767  0    0    0    0    0    0| iscsid
```

42.5. TRACKING MOST FREQUENTLY USED SYSTEM CALLS WITH SYSTEMTAP

You can use the **topsys.stp** SystemTap script to list the top 20 system calls used by the system per 5-second interval. It also lists how many times each system call was used during that period.

Prerequisites

- You have installed SystemTap as described in [Installing Systemtap](#).

Procedure

- Run the **topsys.stp** script:

```
# stap --example topsys.stp
```

Consider the following example:

```
# stap -v --example topsys.stp
```

where `-v` makes the output of starting kernel visible.

The output should look similar to the following:

```
-----
      SYSCALL    COUNT
gettimeofday    1857
      read      1821
      ioctl     1568
      poll      1033
      close      638
      open       503
      select     455
      write      391
      writev     335
      futex      303
      recvmmsg   251
      socket     137
      clock_gettime 124
      rt_sigprocmask 121
      sendto     120
      setitimer   106
      stat        90
      time        81
      sigreturn   72
      fstat       66
-----
```

42.6. TRACKING SYSTEM CALL VOLUME PER PROCESS WITH SYSTEMTAP

You can use the **syscalls_by_proc.stp** SystemTap script to see which processes are performing the highest volume of system calls. It displays 20 processes performing the most of system calls.

Prerequisites

- You have installed SystemTap as described in [Installing Systemtap](#).

Procedure

- Run the **syscalls_by_proc.stp** script:

```
# stap --example syscalls_by_proc.stp
```

Output of the **syscalls_by_proc.stp** script looks similar to the following:

```
Collecting data... Type Ctrl-C to exit and display results
#SysCalls Process Name
1577    multiloader-apple
692     synergyc
408     pcscd
376     mixer_applet2
299     gnome-terminal
293     Xorg
206     scim-panel-gtk
95      gnome-power-man
90      artsd
85      dhcdbd
84      scim-bridge
78      gnome-screensav
66      scim-launcher
[...]
```

CHAPTER 43. MONITORING DISK AND I/O ACTIVITY WITH SYSTEMTAP

You can monitor disk and I/O activity with the following scripts.

43.1. SUMMARIZING DISK READ/WRITE TRAFFIC WITH SYSTEMTAP

You can use the **disktop.stp** SystemTap script to identify which processes are performing the heaviest disk reads and writes to the system.

Prerequisites

- You have installed SystemTap as described in [Installing Systemtap](#).

Procedure

- Run the **disktop.stp** script:

```
# stap --example disktop.stp
```

The script displays the top ten processes responsible for the heaviest reads or writes to a disk.

The output includes the following data per listed process:

UID

User ID. A user ID of **0** refers to the root user.

PID

The ID of the listed process.

PPID

The process ID of the listed process's parent process.

CMD

The name of the listed process.

DEVICE

Which storage device the listed process is reading from or writing to.

T

The type of action performed by the listed process, where **W** refers to write, and **R** refers to read.

BYTES

The amount of data read to or written from disk.

Output of the **disktop.stp** script looks similar to the following:

```
[...]
Mon Sep 29 03:38:28 2008 , Average: 19Kb/sec, Read: 7Kb, Write: 89Kb
UID  PID  PPID      CMD  DEVICE  T  BYTES
0  26319  26294    firefox  sda5  W   90229
0  2758   2757  pam_timestamp_c  sda5  R    8064
0  2885    1    cupsd  sda5  W    1678
Mon Sep 29 03:38:38 2008 , Average: 1Kb/sec, Read: 7Kb, Write: 1Kb
```


UID	PID	PPID	CMD	DEVICE	T	BYTES
0	2758	2757	pam_timestamp_c	sda5	R	8064
0	2885	1	cupsd	sda5	W	1678

43.2. TRACKING I/O TIME FOR EACH FILE READ OR WRITE WITH SYSTEMTAP

You can use the **iotime.stp** SystemTap script to monitor the amount of time it takes for each process to read from or write to any file. This helps you to determine what files are slow to load on a system.

Prerequisites

- You have installed SystemTap as described in [Installing Systemtap](#).

Procedure

- Run the **iotime.stp** script:

```
# stap --example iotime.stp
```

The script tracks each time a system call opens, closes, reads from, and writes to a file. For each file any system call accesses, It counts the number of microseconds it takes for any reads or writes to finish and tracks the amount of data , in bytes, read from or written to the file.

The output contains:

- A timestamp, in microseconds
- Process ID and process name
- An **access** or **iotime** flag
- The file accessed

If a process was able to read or write any data, a pair of access and **iotime** lines should appear together. The access line refers to the time that a given process started accessing a file. The end of the access line will show the amount of data read or written. The **iotime** line will show the amount of time, in microseconds, that the process took in order to perform the read or write.

Output of the **iotime.stp** script looks similar to the following:

```
[...]
825946 3364 (NetworkManager) access /sys/class/net/eth0/carrier read: 8190 write: 0
825955 3364 (NetworkManager) iotime /sys/class/net/eth0/carrier time: 9
[...]
117061 2460 (pcscd) access /dev/bus/usb/003/001 read: 43 write: 0
117065 2460 (pcscd) iotime /dev/bus/usb/003/001 time: 7
[...]
3973737 2886 (sendmail) access /proc/loadavg read: 4096 write: 0
3973744 2886 (sendmail) iotime /proc/loadavg time: 11
[...]
```

43.3. TRACKING CUMULATIVE I/O WITH SYSTEMTAP

You can use the **traceio.stp** SystemTap script to track the cumulative amount of I/O to the system.

Prerequisites

- You have installed SystemTap as described in [Installing Systemtap](#).

Procedure

- Run the **traceio.stp** script:

```
# stap --example traceio.stp
```

The script prints the top ten executables generating I/O traffic over time. It also tracks the cumulative amount of I/O reads and writes done by those executables. This information is tracked and printed out in 1-second intervals, and in descending order.

Output of the **traceio.stp** script looks similar to the following:

```
[...]
  Xorg r: 583401 KiB w:    0 KiB
 floaters r:   96 KiB w:  7130 KiB
 multiload-apple r:   538 KiB w:   537 KiB
  sshd r:   71 KiB w:   72 KiB
 pam_timestamp_c r:   138 KiB w:    0 KiB
  staprun r:   51 KiB w:   51 KiB
  snmpd r:   46 KiB w:    0 KiB
  pcscd r:   28 KiB w:    0 KiB
 irqbalance r:   27 KiB w:    4 KiB
  cupsd r:    4 KiB w:   18 KiB
  Xorg r: 588140 KiB w:    0 KiB
 floaters r:   97 KiB w:  7143 KiB
 multiload-apple r:   543 KiB w:   542 KiB
  sshd r:   72 KiB w:   72 KiB
 pam_timestamp_c r:   138 KiB w:    0 KiB
  staprun r:   51 KiB w:   51 KiB
  snmpd r:   46 KiB w:    0 KiB
  pcscd r:   28 KiB w:    0 KiB
 irqbalance r:   27 KiB w:    4 KiB
  cupsd r:    4 KiB w:   18 KiB
```

43.4. MONITORING I/O ACTIVITY ON A SPECIFIC DEVICE WITH SYSTEMTAP

You can use the **traceio2.stp** SystemTap script to monitor I/O activity on a specific device.

Prerequisites

- You have installed SystemTap as described in [Installing Systemtap](#).

Procedure

- Run the **traceio2.stp** script.

```
# stap --example traceio2.stp 'argument'
```

This script takes the whole device number as an argument. To find this number you can use:

```
# stat -c "0x%D" directory
```

Where *directory* is located on the device you want to monitor.

The output contains following:

- The name and ID of any process performing a read or write
- The function it is performing (**vfs_read** or **vfs_write**)
- The kernel device number

Consider following output of **# stap traceio2.stp 0x805**

```
[...]
synergyc(3722) vfs_read 0x800005
synergyc(3722) vfs_read 0x800005
cupsd(2889) vfs_write 0x800005
cupsd(2889) vfs_write 0x800005
cupsd(2889) vfs_write 0x800005
[...]
```

43.5. MONITORING READS AND WRITES TO A FILE WITH SYSTEMTAP

You can use the **inodewatch.stp** SystemTap script to monitor reads from and writes to a file in real time.

Prerequisites

- You have installed SystemTap as described in [Installing Systemtap](#).

Procedure

- Run the **inodewatch.stp** script.

```
# stap --example inodewatch.stp 'argument1' 'argument2' 'argument3'
```

The script **inodewatch.stp** takes three command-line arguments:

1. The file's major device number.
2. The file's minor device number.
3. The file's inode number.

You can get these numbers using:

```
# stat -c '%D %i' filename
```

Where *filename* is an absolute path.

Consider following example:

```
# stat -c '%D %i' /etc/crontab
```

The output should look like:

```
805 1078319
```

where:

- **805** is the base-16 (hexadecimal) device number. The last two digits are the minor device number, and the remaining digits are the major number.
- **1078319** is the inode number.

To start monitoring **/etc/crontab**, run:

```
# stap inodewatch.stp 0x8 0x05 1078319
```

In the first two arguments you must use 0x prefixes for base-16 numbers.

The output contains following:

- The name and ID of any process performing a read or write
- The function it is performing (**vfs_read** or **vfs_write**)
- The kernel device number

The output of this example should look like:

```
cat(16437) vfs_read 0x8000005/1078319
cat(16437) vfs_read 0x8000005/1078319
```