Red Hat Enterprise Linux 8

Monitoring and managing system status and performance

Optimizing system throughput, latency, and power consumption
Red Hat Enterprise Linux 8 Monitoring and managing system status and performance

Optimizing system throughput, latency, and power consumption
Abstract

This documentation collection provides instructions on how to monitor and optimize the throughput, latency, and power consumption of Red Hat Enterprise Linux 8 in different scenarios.
Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE .................................................. 8

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION .............................. 9

CHAPTER 1. OVERVIEW OF PERFORMANCE MONITORING OPTIONS ................ 10

CHAPTER 2. GETTING STARTED WITH TUNED ............................................ 12
  2.1. THE PURPOSE OF TUNED ................................................................. 12
  2.2. TUNED PROFILES ............................................................................ 12
    The default profile ............................................................................... 12
    Merged profiles .................................................................................. 13
    The location of profiles ...................................................................... 13
    The syntax of profile configuration .................................................... 14
  2.3. TUNED PROFILES DISTRIBUTED WITH RHEL ................................. 14
    Real-time profiles ............................................................................... 17
  2.4. STATIC AND DYNAMIC TUNING IN TUNED ...................................... 17
  2.5. TUNED NO-DAEMON MODE ............................................................. 18
  2.6. INSTALLING AND ENABLING TUNED .......................................... 18
  2.7. LISTING AVAILABLE TUNED PROFILES .................................... 19
  2.8. SETTING A TUNED PROFILE .......................................................... 19
  2.9. DISABLING TUNED .......................................................................... 21
  2.10. RELATED INFORMATION ............................................................... 21

CHAPTER 3. CUSTOMIZING TUNED PROFILES ............................................ 22
  3.1. TUNED PROFILES ............................................................................ 22
    The default profile ............................................................................... 22
    Merged profiles .................................................................................. 22
    The location of profiles ...................................................................... 23
    The syntax of profile configuration .................................................... 23
  3.2. INHERITANCE BETWEEN TUNED PROFILES ................................... 23
  3.3. STATIC AND DYNAMIC TUNING IN TUNED ...................................... 24
  3.4. TUNED PLUG-INS ............................................................................ 25
    Monitoring plug-ins ........................................................................... 25
    Tuning plug-ins .................................................................................. 25
    Syntax for plug-ins in Tuned profiles .................................................. 25
    Short plug-in syntax ........................................................................... 26
    Conflicting plug-in definitions in a profile ............................................ 26
    Functionality not implemented in any plug-in ...................................... 27
  3.5. AVAILABLE TUNED PLUG-INS ....................................................... 27
    Monitoring plug-ins ........................................................................... 27
    Tuning plug-ins .................................................................................. 27
  3.6. VARIABLES AND BUILT-IN FUNCTIONS IN TUNED PROFILES ....... 30
    Variables ........................................................................................... 31
    Functions .......................................................................................... 31
  3.7. BUILT-IN FUNCTIONS AVAILABLE IN TUNED PROFILES ............. 32
  3.8. CREATING NEW TUNED PROFILES ............................................ 33
  3.9. MODIFYING EXISTING TUNED PROFILES .................................... 34
  3.10. SETTING THE DISK SCHEDULER USING TUNED ......................... 35
  3.11. RELATED INFORMATION ............................................................... 37

CHAPTER 4. REVIEWING A SYSTEM USING TUNA INTERFACE ................. 38
  4.1. INSTALLING TUNA TOOL ................................................................. 38
CHAPTER 13. GETTING STARTED WITH PERF ................................................................. 119
13.1. INTRODUCTION TO PERF .............................................................................. 119
13.2. INSTALLING PERF ........................................................................................... 119
13.3. COMMON PERF COMMANDS ........................................................................... 119
13.4. REAL TIME PROFILING OF CPU USAGE WITH PERF TOP ....................... 120
   13.4.1. The purpose of perf top ................................................................................. 120
   13.4.2. Profiling CPU usage with perf top ................................................................. 120
   13.4.3. Interpretation of perf top output .................................................................... 121
   13.4.4. Why perf displays some function names as raw function addresses .......... 121
   13.4.5. Enabling debug and source repositories ..................................................... 121
   13.4.6. Getting debuginfo packages for an application or library using GDB ........ 122
13.5. COUNTING EVENTS DURING PROCESS EXECUTION ...................................... 123
   13.5.1. The purpose of perf stat ................................................................................. 123
   13.5.2. Counting events with perf stat ...................................................................... 123
   13.5.3. Interpretation of perf stat output ................................................................... 125
   13.5.4. Attaching perf stat to a running process ..................................................... 125
13.6. RECORDING AND ANALYZING PERFORMANCE PROFILES WITH PERF ........ 125
   13.6.1. The purpose of perf record .......................................................................... 125
   13.6.2. Recording a performance profile without root access .................................. 126
   13.6.3. Recording a performance profile with root access ....................................... 126
   13.6.4. Recording a performance profile in per-CPU mode ...................................... 127
   13.6.5. Capturing call graph data with perf record ................................................. 127
   13.6.6. Analyzing perf.data with perf report ......................................................... 128
   13.6.7. Interpretation of perf report output .............................................................. 129
   13.6.8. Why perf displays some function names as raw function addresses ........... 129
   13.6.9. Enabling debug and source repositories ..................................................... 129
   13.6.10. Getting debuginfo packages for an application or library using GDB ........ 130

CHAPTER 14. MONITORING SYSTEM PERFORMANCE WITH PERF ............................... 132
14.1. RECORDING A PERFORMANCE PROFILE IN PER-CPU MODE .................... 132
14.2. CAPTURING CALL GRAPH DATA WITH PERF RECORD ............................... 132
14.3. IDENTIFYING BUSY CPUS WITH PERF .......................................................... 133
   14.3.1. Displaying which CPU events were counted on with perf stat ................. 133
   14.3.2. Displaying which CPU samples were taken on with perf report ............... 133
   14.3.3. Displaying specific CPUs during profiling with perf top ......................... 134
14.4. MONITORING SPECIFIC CPUS WITH PERF .................................................... 134
   14.4.1. Monitoring specific CPUs with perf record and perf report ....................... 134
   14.4.2. Displaying specific CPUs during profiling with perf top ......................... 135
14.5. GENERATING A PERF.DAT A FILE THAT IS READABLE ON A DIFFERENT DEVICE .................................................. 136
14.6. ANALYZING A PERF.DAT A FILE THAT WAS CREATED ON A DIFFERENT DEVICE .................................................. 137

CHAPTER 15. MONITORING APPLICATION PERFORMANCE WITH PERF .................... 138
15.1. ATTACHING PERF RECORD TO A RUNNING PROCESS ................................. 138
15.2. CAPTURING CALL GRAPH DATA WITH PERF RECORD ............................... 138
15.3. ANALYZING PERF.DAT A FILE WITH PERF REPORT ...................................... 139

CHAPTER 16. PROFILING MEMORY ACCESES WITH PERF MEM .............................. 141
16.1. THE PURPOSE OF PERF MEM .......................................................................... 141
16.2. SAMPLING MEMORY ACCESS WITH PERF MEM ......................................... 141
16.3. INTERPRETATION OF PERF MEM REPORT OUTPUT ..................................... 143

CHAPTER 17. DETECTING FALSE SHARING WITH PERF C2C ................................. 145
17.1. THE PURPOSE OF PERF C2C .......................................................................... 145
17.2. FALSE SHARING ............................................................................................... 145
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>17.3. DETECTING CACHE-LINE CONTENTION WITH PERF C2C</td>
<td>145</td>
</tr>
<tr>
<td></td>
<td>17.4. VISUALIZING A PERF.DATA FILE RECORDED WITH PERF C2C RECORD</td>
<td>145</td>
</tr>
<tr>
<td></td>
<td>17.5. INTERPRETATION OF PERF C2C REPORT OUTPUT</td>
<td>148</td>
</tr>
<tr>
<td></td>
<td>17.6. DETECTING FALSE SHARING WITH PERF C2C</td>
<td>149</td>
</tr>
<tr>
<td>18</td>
<td>CHAPTER 18. GETTING STARTED WITH FLAMEGRAPHS</td>
<td>151</td>
</tr>
<tr>
<td></td>
<td>18.1. INSTALLING FLAMEGRAPHS</td>
<td>151</td>
</tr>
<tr>
<td></td>
<td>18.2. CREATING FLAMEGRAPHS OVER THE ENTIRE SYSTEM</td>
<td>151</td>
</tr>
<tr>
<td></td>
<td>18.3. CREATING FLAMEGRAPHS OVER SPECIFIC PROCESSES</td>
<td>152</td>
</tr>
<tr>
<td></td>
<td>18.4. INTERPRETING FLAMEGRAPHS</td>
<td>153</td>
</tr>
<tr>
<td>19</td>
<td>CHAPTER 19. PROFILING MEMORY ALLOCATION WITH NUMASTAT</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>19.1. DEFAULT NUMASTAT STATISTICS</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>19.2. PROFILING MEMORY ALLOCATION WITH NUMASTAT</td>
<td>155</td>
</tr>
<tr>
<td>20</td>
<td>CHAPTER 20. CONFIGURING AN OPERATING SYSTEM TO OPTIMIZE CPU UTILIZATION</td>
<td>157</td>
</tr>
<tr>
<td></td>
<td>20.1. TOOLS FOR MONITORING AND DIAGNOSING PROCESSOR ISSUES</td>
<td>157</td>
</tr>
<tr>
<td></td>
<td>20.2. DETERMINING SYSTEM TOPOLOGY</td>
<td>158</td>
</tr>
<tr>
<td></td>
<td>20.2.1. Types of system topology</td>
<td>158</td>
</tr>
<tr>
<td></td>
<td>20.2.2. Displaying system topologies</td>
<td>158</td>
</tr>
<tr>
<td></td>
<td>20.3. TUNING SCHEDULING POLICY</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>20.3.1. Static priority scheduling with SCHED_FIFO</td>
<td>161</td>
</tr>
<tr>
<td></td>
<td>20.3.2. Round robin priority scheduling with SCHED_RR</td>
<td>162</td>
</tr>
<tr>
<td></td>
<td>20.3.3. Normal scheduling with SCHED_OTHER</td>
<td>162</td>
</tr>
<tr>
<td></td>
<td>20.3.4. Setting scheduler policies</td>
<td>162</td>
</tr>
<tr>
<td></td>
<td>20.3.5. Policy options for the chrt command</td>
<td>163</td>
</tr>
<tr>
<td></td>
<td>20.3.6. Changing the priority of services during the boot process</td>
<td>163</td>
</tr>
<tr>
<td></td>
<td>20.3.7. Priority map</td>
<td>165</td>
</tr>
<tr>
<td></td>
<td>20.3.8. cpu-partitioning profile</td>
<td>165</td>
</tr>
<tr>
<td></td>
<td>20.3.9. Additional resources</td>
<td>166</td>
</tr>
<tr>
<td></td>
<td>20.4. CONFIGURING KERNEL TICK TIME</td>
<td>166</td>
</tr>
<tr>
<td></td>
<td>20.5. SETTING INTERRUPT AFFINITY SYSTEMS</td>
<td>168</td>
</tr>
<tr>
<td></td>
<td>20.5.1. Balancing interrupts manually</td>
<td>168</td>
</tr>
<tr>
<td></td>
<td>20.5.2. Setting the smp_affinity mask</td>
<td>169</td>
</tr>
<tr>
<td></td>
<td>20.5.3. Additional resources</td>
<td>169</td>
</tr>
<tr>
<td>21</td>
<td>CHAPTER 21. FACTORS AFFECTING I/O AND FILE SYSTEM PERFORMANCE</td>
<td>170</td>
</tr>
<tr>
<td></td>
<td>21.1. TOOLS FOR MONITORING AND DIAGNOSING I/O AND FILE SYSTEM ISSUES</td>
<td>170</td>
</tr>
<tr>
<td></td>
<td>21.2. AVAILABLE TUNING OPTIONS FOR FORMATTING A FILE SYSTEM</td>
<td>172</td>
</tr>
<tr>
<td></td>
<td>21.3. AVAILABLE TUNING OPTIONS FOR MOUNTING A FILE SYSTEM</td>
<td>173</td>
</tr>
<tr>
<td></td>
<td>21.4. TYPES OF DISCARDING UNUSED BLOCKS</td>
<td>174</td>
</tr>
<tr>
<td></td>
<td>21.5. SOLID-STATE DISKS TUNING CONSIDERATIONS</td>
<td>174</td>
</tr>
<tr>
<td></td>
<td>21.6. GENERIC BLOCK DEVICE TUNING PARAMETERS</td>
<td>175</td>
</tr>
<tr>
<td>22</td>
<td>CHAPTER 22. CONFIGURING RHEL TO OPTIMIZE ACCESS TO NETWORK RESOURCES</td>
<td>177</td>
</tr>
<tr>
<td></td>
<td>22.1. TOOLS FOR MONITORING AND DIAGNOSING PERFORMANCE ISSUES</td>
<td>177</td>
</tr>
<tr>
<td></td>
<td>22.2. BOTTLENECKS IN A PACKET RECEPTION</td>
<td>178</td>
</tr>
<tr>
<td></td>
<td>22.3. CONFIGURING BUSY POLLING</td>
<td>179</td>
</tr>
<tr>
<td></td>
<td>22.3.1. Enabling busy polling</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>22.4. RECEIVE-SIDE SCALING</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>22.4.1. Viewing the interrupt request queues</td>
<td>181</td>
</tr>
<tr>
<td></td>
<td>22.5. RECEIVE PACKET STEERING</td>
<td>182</td>
</tr>
<tr>
<td></td>
<td>22.6. RECEIVE FLOW STEERING</td>
<td>182</td>
</tr>
<tr>
<td></td>
<td>22.6.1. Enabling Receive Flow Steering</td>
<td>183</td>
</tr>
</tbody>
</table>
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 input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages:
  1. Make sure you are viewing the documentation in the *Multi-page HTML* format. In addition, ensure you see the *Feedback* button in the upper right corner of the document.
  2. Use your mouse cursor to highlight the part of text that you want to comment on.
  3. Click the *Add Feedback* pop-up that appears below the highlighted text.
  4. Follow the displayed instructions.

- For submitting more complex feedback, create a Bugzilla ticket:
  1. Go to the *Bugzilla* website.
  2. As the Component, use *Documentation*.
  3. Fill in the *Description* field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
  4. Click *Submit Bug*. 
CHAPTER 1. OVERVIEW OF PERFORMANCE MONITORING OPTIONS

The following are some of the performance monitoring and configuration tools available in Red Hat Enterprise Linux 8:

- Performance Co-Pilot (pcp) is used for monitoring, visualizing, storing, and analyzing system-level performance measurements. It allows the monitoring and management of real-time data, and logging and retrieval of historical data.

- Red Hat Enterprise Linux 8 provides several tools that can be used from the command line to monitor a system outside run level 5. The following are the built-in command line tools:
  - **top** is provided by the procps-ng package. It gives a dynamic view of the processes in a running system. It displays a variety of information, including a system summary and a list of tasks currently being managed by the Linux kernel.
  - **ps** is provided by the procps-ng package. It captures a snapshot of a select group of active processes. By default, the examined group is limited to processes that are owned by the current user and associated with the terminal where the **ps** command is executed.
  - Virtual memory statistics (**vmstat**) is provided by the procps-ng package. It provides instant reports of your system’s processes, memory, paging, block input/output, interrupts, and CPU activity.
  - System activity reporter (**sar**) is provided by the sysstat package. It collects and reports information about system activity that has occurred so far on the current day.
  - **perf** uses hardware performance counters and kernel trace-points to track the impact of other commands and applications on a system.
  - **bcc-tools** is used for BPF Compiler Collection (BCC). It provides over 100 eBPF scripts that monitor kernel activities. For more information on each of this tool, see the man page describing how to use it and what functions it performs.
  - **turbostat** is provided by the kernel-tools package. It reports on processor topology, frequency, idle power-state statistics, temperature, and power usage on the Intel 64 processors.
  - **iostat** is provided by the sysstat package. It monitors and reports on system input/output device loading to help administrators make decisions about how to balance input/output load between physical disks.
  - **irqbalance** distributes hardware interrupts across processors to improve system performance.
  - **ss** prints statistical information about sockets, allowing administrators to assess device performance over time. Red Hat recommends using **ss** over netstat in Red Hat Enterprise Linux 8.
  - **numastat** is provided by the numactl package. By default, numastat displays per-node NUMA hit and miss system statistics from the kernel memory allocator. Optimal performance is indicated by high numa_hit values and low numa_miss values.
  - **numad** is an automatic NUMA affinity management daemon. It monitors NUMA topology and resource usage within a system that dynamically improves NUMA resource allocation, management, and therefore system performance.
- **SystemTap** monitors and analyzes operating system activities, especially the kernel activities.

- **valgrind** analyzes applications by running it on a synthetic CPU and instrumenting existing application code as it is executed. It then prints commentary that clearly identifies each process involved in application execution to a user-specified file, file descriptor, or network socket. It is also useful for finding memory leaks.

- **pqos** is provided by the the **intel-cmt-cat** package. It monitors and controls CPU cache and memory bandwidth on recent Intel processors.

### Additional resources

- For more information, see the man pages of **pcp**, **top**, **ps**, **vmstat**, **sar**, **perf**, **iostat**, **irqbalance**, **ss**, **numastat**, **numad**, **valgrind**, and **pqos**.

- For more information on **pcp**, see the documentation in the `/usr/share/doc/` directory.

- For more information on the **await** value and what can cause its values to be high, see the Red Hat Knowledgebase article: [What exactly is the meaning of value “await” reported by iostat?](#).
CHAPTER 2. 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.

### 2.1. 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.

### 2.2. 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

**The default 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:

<table>
<thead>
<tr>
<th>Environment</th>
<th>Default profile</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute nodes</td>
<td><strong>throughput-performance</strong></td>
<td>The best throughput performance</td>
</tr>
</tbody>
</table>
Virtual machines

<table>
<thead>
<tr>
<th>Environment</th>
<th>Default profile</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual machines</td>
<td>virtual-guest</td>
<td>The best performance. If you are not interested in the best performance, you can change it to the balanced or powersave profile.</td>
</tr>
<tr>
<td>Other cases</td>
<td>balanced</td>
<td>Balanced performance and power consumption</td>
</tr>
</tbody>
</table>

**Merged 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:

```bash
# 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.

**The location of 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.
The 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

- The tuned.conf(5) man page.

2.3. 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 either the latency-performance or throughput-performance profile, and 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 8 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 either the `latency-performance` or `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 either the `latency-performance` or `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.

**cpu-partitioning**

The `cpu-partitioning` profile partitions the system CPUs into isolated and housekeeping CPUs. To reduce jitter and interruptions on an isolated CPU, the profile clears the isolated CPU from user-space processes, movable kernel threads, interrupt handlers, and kernel timers. A housekeeping CPU can run all services, shell processes, and kernel threads.

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

- **isolated_cores=cpu-list**
  Lists CPUs to isolate. The list of isolated CPUs is comma-separated or the user can specify the range. 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.

- **no_balance_cores=cpu-list**
  Lists CPUs which are not considered by the kernel during system wide process load-balancing. This option is optional. This is usually the same list as `isolated_cores`.

For more information on `cpu-partitioning`, see the `tuned-profiles-cpu-partitioning(7)` man page.

**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.

**postgresql**

A profile optimized for PostgreSQL databases loads based on `throughput-performance` profile.
A profile optimized for PostgreSQL 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-postgresql` package.

**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
```

**Real-time profiles**
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.

### 2.4. STATIC AND DYNAMIC TUNING IN TUNED

This section explains the difference between the two categories of system tuning that Tuned applies: static and dynamic.

**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 2.2. 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.

### 2.5. 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:

```bash
daemon = 0
```

### 2.6. 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:

   ```bash
   # yum install tuned
   ```

2. Enable and start the tuned service:

   ```bash
   # systemctl enable --now tuned
   ```
3. Optionally, install Tuned profiles for real-time systems:

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

4. Verify that a Tuned profile is active and applied:

```
$ tuned-adm active
Current active profile: balanced

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

## 2.7. 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:**
- 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: balanced
```

### Additional resources

- The tuned-adm(8) man page.

## 2.8. SETTING A TUNED PROFILE
This procedure activates a selected Tuned profile on your system.

Prerequisites

- The tuned service is running. See Section 2.6, “Installing and enabling Tuned” for details.

Procedure

1. Optionally, you can let Tuned recommend the most suitable profile for your system:
   
   ```bash
   # tuned-adm recommend
   balanced
   ```

2. Activate a profile:
   
   ```bash
   # tuned-adm profile selected-profile
   ```

   Alternatively, you can activate a combination of multiple profiles:
   
   ```bash
   # tuned-adm profile profile1 profile2
   ```

   **Example 2.3. 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:
   
   ```bash
   # tuned-adm profile virtual-guest powersave
   ```

3. View the current active Tuned profile on your system:
   
   ```bash
   # tuned-adm active
   Current active profile: selected-profile
   ```

4. Reboot the system:
   
   ```bash
   # reboot
   ```

Verification steps

- Verify that the Tuned profile is active and applied:
  
  ```bash
  $ 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
2.9. 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

- The tuned-adm(8) man page.

2.10. RELATED INFORMATION

- The tuned(8) man page
- The tuned-adm(8) man page
- The Tuned project website: https://tuned-project.org/
CHAPTER 3. 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 Section 2.6, "Installing and enabling Tuned".

3.1. 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

The default 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:

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

Merged 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 3.1. 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.

**The location of 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.

**The 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**

- The `tuned.conf(5)` man page.

### 3.2. 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 3.2. 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

- The tuned.conf(5) man page

3.3. STATIC AND DYNAMIC TUNING IN TUNED

This section explains the difference between the two categories of system tuning that Tuned applies: static and dynamic.

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 3.3. 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.

### 3.4. 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
- tuning 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 3.4. 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 3.5. 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.

**Functionality not implemented in any plug-in**

*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

- The `tuned.conf(5)` man page

### 3.5. AVAILABLE TUNED PLUG-INS

This section lists all monitoring and tuning plug-ins currently available in *Tuned*.

#### 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 3.6. 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 3.7. 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
```

### 3.6. VARIABLES AND BUILT-IN FUNCTIONS IN TUNED PROFILES
Variables and built-in functions expand at run time when a Tuned profile is activated.

Using Tuned variables reduces the amount of necessary typing in Tuned profiles. You can also:

- 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

Variables
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 3.8. 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.

Functions
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 3.9. 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:

```plaintext
[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

- The `tuned.conf(5)` man page

3.7. 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.

3.8. CREATING NEW TUNED PROFILES

This procedure creates a new Tuned profile with custom performance rules.

Prerequisites

- The tuned service is installed and running. See Section 2.6, “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:

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

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

```bash
$ 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

- The `tuned.conf(5)` man page

### 3.9. MODIFYING EXISTING TUNED PROFILES

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

**Prerequisites**

- The `tuned` service is installed and running. See Section 2.6, “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:

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

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

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

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

3. Include your profile modifications.

   **Example 3.10. 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:

   ```ini
   [main]
   include=throughput-performance
   ```
[sysctl]
vm.swappiness=5

4. To activate the profile, use:

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

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

```bash
$ 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

- The `tuned.conf(5)` man page

### 3.10. 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 Section 2.6, “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 Section 2.3, "Tuned profiles distributed with RHEL".
   To see which profile is currently active, use:

   ```bash
   $ tuned-adm active
   ```

2. Create a new directory to hold your Tuned profile:

   ```bash
   # 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:

   - Optional: Include an existing profile:

     [main]
     include=existing-profile

   - 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

     - 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

6. 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.
Additional resources

- For more information on creating a Tuned profile, see Chapter 3, Customizing Tuned profiles.

3.11. RELATED INFORMATION

- The tuned.conf(5) man page

- The Tuned project website: https://tuned-project.org/
CHAPTER 4. 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.

4.1. 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.

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 on threads
- Displays the current policies and priorities of a system

Procedure

1. To install the tuna tool:

   # yum install tuna

2. To display the available tuna CLI options:

   # tuna -h

Additional resources

- The tuna man page.

4.2. 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 Section 4.1, “Installing tuna tool”.

Procedure

- To view the current policies and priorities:

  # tuna --show_threads

  thread
  pid  SCHED_rt pri  affinity  cmd
  1 OTHER  0  0,1  init
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 Section 4.3, “Tuning CPUs using tuna tool”.
- To tune the IRQs using the `tuna` tool, see Section 4.4, “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**

- The `tuna` man page.
- The `tuna -h` command displays available CLI options.

### 4.3. 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 Section 4.1, “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**

- To 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
  ```

<table>
<thead>
<tr>
<th>thread</th>
<th>ctxt_switches</th>
<th>pid SCHED_ rtpri</th>
<th>affinity</th>
<th>voluntary</th>
<th>nonvoluntary</th>
<th>cmd</th>
</tr>
</thead>
<tbody>
<tr>
<td>3861</td>
<td>OTHER</td>
<td>0</td>
<td>0,1</td>
<td>33997</td>
<td>58</td>
<td>gnome-screensav</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>thread</th>
<th>ctxt_switches</th>
<th>pid SCHED_ rtpri</th>
<th>affinity</th>
<th>voluntary</th>
<th>nonvoluntary</th>
<th>cmd</th>
</tr>
</thead>
<tbody>
<tr>
<td>3861</td>
<td>OTHER</td>
<td>0</td>
<td>0</td>
<td>33997</td>
<td>58</td>
<td>gnome-screensav</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>thread</th>
<th>ctxt_switches</th>
<th>pid SCHED_ rtpri</th>
<th>affinity</th>
<th>voluntary</th>
<th>nonvoluntary</th>
<th>cmd</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
- The `/proc/cpuinfo` file.
- The `tuna` man page.
- The `tuna -h` command displays available CLI options.

4.4. 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 Section 4.1, “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, `--isolate`.

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 Section 4.3, “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**

- The `/proc/interrupts` file.
- The `tuna` man page.
- The `tuna -h` command displays available CLI options.
CHAPTER 5. MONITORING PERFORMANCE USING RHEL SYSTEM ROLES

5.1. INTRODUCTION TO RHEL SYSTEM ROLES

RHEL System Roles is a collection of Ansible roles and modules. RHEL System Roles provide a configuration interface to remotely manage multiple RHEL systems. The interface enables managing system configurations across multiple versions of RHEL, as well as adopting new major releases.

On Red Hat Enterprise Linux 8, the interface currently consists of the following roles:

- kdump
- network
- selinux
- storage
- certificate
- kernel_settings
- logging
- metrics
- nbde_client and nbde_server
- timesync
- tlog

All these roles are provided by the `rhel-system-roles` package available in the AppStream repository.

Additional resources

- For RHEL System Roles overview, see the Red Hat Enterprise Linux (RHEL) System Roles Red Hat Knowledgebase article.

- For information on a particular role, see the documentation under the `/usr/share/doc/rhel-system-roles` directory. This documentation is installed automatically with the `rhel-system-roles` package.

5.2. RHEL SYSTEM ROLES TERMINOLOGY

You can find the following terms across this documentation:

System Roles terminology

Ansible playbook

Playbooks are Ansible’s configuration, deployment, and orchestration language. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process.

Control node
Any machine with Ansible installed. You can run commands and playbooks, invoking /usr/bin/ansible or /usr/bin/ansible-playbook, from any control node. You can use any computer that has Python installed on it as a control node - laptops, shared desktops, and servers can all run Ansible. However, you cannot use a Windows machine as a control node. You can have multiple control nodes.

Inventory
A list of managed nodes. An inventory file is also sometimes called a “hostfile”. Your inventory can specify information like IP address for each managed node. An inventory can also organize managed nodes, creating and nesting groups for easier scaling. To learn more about inventory, see the Working with Inventory section.

Managed nodes
The network devices, servers, or both that you manage with Ansible. Managed nodes are also sometimes called “hosts”. Ansible is not installed on managed nodes.

5.3. INSTALLING RHEL SYSTEM ROLES IN YOUR SYSTEM

This paragraph is the procedure module introduction: a short description of the procedure.

Prerequisites
- You have a Red Hat Ansible Engine Subscription. See the procedure How do I Download and Install Red Hat Ansible Engine?
- You have Ansible packages installed in the system you want to use as a control node:

Procedure
1. Install the rhel-system-roles package on the system that you want to use as a control node:

   # yum install rhel-system-roles

   If you do not have a Red Hat Ansible Engine Subscription, you can use a limited supported version of Red Hat Ansible Engine provided with your Red Hat Enterprise Linux subscription. In this case, follow these steps:

   a. Enable the RHEL Ansible Engine repository:

      # subscription-manager refresh

      # subscription-manager repos --enable ansible-2-for-rhel-8-x86_64-rpms

   b. Install Ansible Engine:

      # yum install ansible

As a result, you are able to create an Ansible playbook.

Additional resources
- For RHEL System Roles overview, see the Red Hat Enterprise Linux (RHEL) System Roles
- For more detailed information on using the ansible-playbook command, see the ansible-playbook man page.
5.4. APPLYING A ROLE

The following procedure describes how to apply a particular role.

Prerequisites

- The `rhel-system-roles` package is installed on the system that you want to use as a control node:

  ```
  # yum install rhel-system-roles
  ```

- The Ansible Engine repository is enabled, and the `ansible` package is installed on the system that you want to use as a control node. You need the `ansible` package to run playbooks that use RHEL System Roles.

  - If you do not have a Red Hat Ansible Engine Subscription, you can use a limited supported version of Red Hat Ansible Engine provided with your Red Hat Enterprise Linux subscription. In this case, follow these steps:
    1. Enable the RHEL Ansible Engine repository:

       ```
       # subscription-manager refresh
       # subscription-manager repos --enable ansible-2-for-rhel-8-x86_64-rpms
       ```

    2. Install Ansible Engine:

       ```
       # yum install ansible
       ```

      - If you have a Red Hat Ansible Engine Subscription, follow the procedure described in How do I Download and Install Red Hat Ansible Engine?

- You are able to create an Ansible playbook. Playbooks represent Ansible’s configuration, deployment, and orchestration language. By using playbooks, you can declare and manage configurations of remote machines, deploy multiple remote machines or orchestrate steps of any manual ordered process. A playbook is a list of one or more `plays`. Every `play` can include Ansible variables, tasks, or roles. Playbooks are human-readable, and they are expressed in the YAML format. For more information about playbooks, see Ansible documentation.

Procedure

1. Create an Ansible playbook including the required role. The following example shows how to use roles through the `roles:` option for a given `play`:

   ```yaml
   ---
   - hosts: webservers
     roles:
       - rhel-system-roles.network
       - rhel-system-roles.timesync
   ```

   For more information on using roles in playbooks, see Ansible documentation.
See Ansible examples for example playbooks.

NOTE

Every role includes a README file, which documents how to use the role and supported parameter values. You can also find an example playbook for a particular role under the documentation directory of the role. Such documentation directory is provided by default with the `rhel-system-roles` package, and can be found in the following location:

```
/usr/share/doc/rhel-system-roles/SUBSYSTEM/
```

Replace `SUBSYSTEM` with the name of the required role, such as `selinux`, `kdump`, `network`, `timesync`, or `storage`.

2. Verify the playbook syntax:

```
# ansible-playbook --syntax-check name.of.the.playbook
```

The `ansible-playbook` command offers a `--syntax-check` option that you can use to verify the syntax of a playbook.

3. Execute the playbook on targeted hosts by running the `ansible-playbook` command:

```
# ansible-playbook -i name.of.the.inventory name.of.the.playbook
```

An inventory is a list of systems against which Ansible works. For more information on how to create and inventory, and how to work with it, see Ansible documentation.

If you do not have an inventory, you can create it at the time of running `ansible-playbook`.

If you have only one targeted host against which you want to run the playbook, use:

```
# ansible-playbook -i host1, name.of.the.playbook
```

If you have multiple targeted hosts against which you want to run the playbook, use:

```
# ansible-playbook -i host1,host2,.....,hostn name.of.the.playbook
```

Additional resources

- For more detailed information on using the `ansible-playbook` command, see the `ansible-playbook` man page.

5.5. 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 5.1. Metrics system role variables

<table>
<thead>
<tr>
<th>Role variable</th>
<th>Description</th>
<th>Example usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>metrics_monitored_hosts</td>
<td>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 /var/log for each host.</td>
<td>metrics_monitored_hosts: [&quot;webserver.example.com&quot;, &quot;database.example.com&quot;]</td>
</tr>
<tr>
<td>metrics_retention_days</td>
<td>Configures the number of days for performance data retention before deletion.</td>
<td>metrics_retention_days: 14</td>
</tr>
<tr>
<td>metrics_graph_service</td>
<td>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.</td>
<td>metrics_graph_service: false</td>
</tr>
<tr>
<td>metrics_query_service</td>
<td>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.</td>
<td>metrics_query_service: false</td>
</tr>
<tr>
<td>metrics_provider</td>
<td>Specifies which metrics collector to use to provide metrics. Currently, pcp is the only supported metrics provider.</td>
<td>metrics_provider: &quot;pcp&quot;</td>
</tr>
</tbody>
</table>

**Additional resources**

- 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.

### 5.6. 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**

- You have Red Hat Ansible Engine installed on the machine you want to monitor.
- 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:

   ```plaintext
   ---
   - hosts: localhost
     vars:
       metrics_graph_service: yes
     roles:
       - rhel-system-roles.metrics
   ```

3. Run the Ansible playbook:

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

   **NOTE**

   Since the `metrics_graph_service` boolean is set to value="yes", `grafana` is automatically installed and provisioned with `pcp` added as a data source.

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.

### 5.7. USING THE METRICS SYSTEM ROLE TO SETUP 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**

- You have Red Hat Ansible Engine installed on the machine you want to use to run the playbook.
- You have the `rhel-system-roles` package installed on the machine you want to use to run the playbook.

**Procedure**

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

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

2. Create an Ansible playbook with the following content:

   ```plaintext
   ---
   - hosts: remotes
   ```
vars:
  metrics_retention_days: 0
roles:
  - rhel-system-roles.metrics

3. Run the Ansible playbook:

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

### 5.8. 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**

- You have Red Hat Ansible Engine installed on the machine you want to use to run the playbook.
- 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:

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

2. Run the Ansible playbook:

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

**NOTE**

Since 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.

3. To view 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.
CHAPTER 6. MONITORING PERFORMANCE WITH PERFORMANCE CO-PILOT

As a system administrator, you can monitor the system’s performance using the Performance Co-Pilot (PCP) application in Red Hat Enterprise Linux 8.

6.1. OVERVIEW OF PCP

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

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.

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.

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 yum install pcp-gui command. For more information, see Section 6.6, “Visually tracing PCP log archives with the PCP Charts application”.

Additional resources

- The /usr/share/doc/pcp-doc/ directory.
- Section 6.9, “Tools distributed with PCP”.
- The Index of Performance Co-Pilot (PCP) articles, solutions, tutorials and white papers on Red Hat Customer Portal.
- The Side-by-side comparison of PCP tools with legacy tools Red Hat Knowledgebase article.
- The upstream PCP documentation.

6.2. INSTALLING AND ENABLING PCP
To begin using PCP, install all the required packages and enable the PCP monitoring services.

**Procedure**

1. Install the PCP package:
   ```
   # yum install pcp
   ```

2. Enable and start the `pmcd` service on the host machine:
   ```
   # systemctl enable pmcd
   # systemctl start pmcd
   ```

3. 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**

- Section 6.9, “Tools distributed with PCP”.
- The `pmcd` man page.

### 6.3. 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 Section 6.2, “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:
3. Execute the required operations to record the performance data.

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

   ```bash
   # 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:

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

   Extract this file and analyze the data using PCP tools.

Additional resources

- Section 6.9, “Tools distributed with PCP”
- Section 6.8, “System services distributed with PCP”
- The `pmlogconf` man page.
- The `pmlogger` man page.
- The `pmcd` man page.

6.4. 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

6.4.1. 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 Section 6.2, “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

- Section 6.9, “Tools distributed with PCP”
- Section 6.8, “System services distributed with PCP”
- The `pmlogconf` man page.
- The `pmlogger` man page.

6.4.2. Editing the `pmlogger` configuration file manually

To create a tailored logging configuration with specific metrics and given intervals, edit the `pmlogger` configuration file manually.

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.

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.

Prerequisites

- PCP is installed. For more information, see Section 6.2, “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
  }
  ```
6.4.3. Enabling the pmlogger service

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

Prerequisites

- PCP is installed. For more information, see Section 6.2, “Installing and enabling PCP”.

Procedure

1. Start and enable the pmlogger service:

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

2. Verify that the pmlogger 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

- Section 6.9, “Tools distributed with PCP”
- Section 6.8, “System services distributed with PCP”
6.4.4. 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 Section 6.2, “Installing and enabling PCP”.

Procedure

1. Install the `pcp-system-tools` package:
   ```bash
   # yum install pcp-system-tools
   ```

2. Configure an IP address for `pmcd`:
   ```bash
   # 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:
   ```bash
   # firewall-cmd --permanent --zone=public --add-port=44321/tcp
   # firewall-cmd --reload
   ```

4. Set an SELinux boolean:
   ```bash
   # setsebool -P pcp_bind_all_unreserved_ports on
   ```

5. Enable the `pmcd` and `pmlogger` services:
   ```bash
   # systemctl enable pmcd pmlogger
   # systemctl restart pmcd pmlogger
   ```

Verification steps

- Verify if the `pmcd` is correctly listening on the configured IP address:
  ```bash
  # ss -tlp | 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 [::1]:44321 [::]:* users:(("pmcd",pid=151595,fd=7))
  ```
6.4.5. 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 Section 6.2, “Installing and enabling PCP”.
- Client is configured for metrics collection. For more information, see Section 6.4.4, “Setting up a client system for metrics collection”.

Procedure

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

   ```
   # yum install pcp-system-tools
   ```

2. Add clients for monitoring:

   ```
   # echo "192.168.4.13 n n PCP_LOG_DIR/pmlogger/rhel7u4a -r -T24h10m \
   -c config.remote" >> /etc/pcp/pmlogger/control.d/remote
   # echo "192.168.4.14 n n PCP_LOG_DIR/pmlogger/rhel6u10a -r -T24h10m \
   -c config.remote" >> /etc/pcp/pmlogger/control.d/remote
   # echo "192.168.4.62 n n PCP_LOG_DIR/pmlogger/rhel8u1a -r -T24h10m \
   -c config.remote" >> /etc/pcp/pmlogger/control.d/remote
   ```

   Replace **192.168.4.13**, **192.168.4.14**, and **192.168.4.62** 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:
The archive files from the `/var/log/pcp/pmlogger/` directory can be used for further analysis and graphing.

**Additional resources**

- Section 6.9, “Tools distributed with PCP”.
- Section 6.8, “System services distributed with PCP”.
- The `/var/lib/pcp/config/pmlogger/config.default` file.
- The `pmlogger` man page.

### 6.4.6. Replaying the PCP log archives with `pmdumptext`

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 `pmdumptext`, `pmrep`, or `pmlogsummary`.

Using the `pmdumptext` 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 Section 6.2, “Installing and enabling PCP”.
- The `pmlogger` service is enabled. For more information, see Section 6.4.3, “Enabling the pmlogger service”.
- Install the `pcp-gui` package:

  ```bash
  # yum install pcp-gui
  ```

**Procedure**
Display the data on the metric:

```
$ pmdumptext -t 5seconds -H -a 20170605 xfs.perdev.log.writes

Time local::xfs.perdev.log.writes["/dev/mapper/fedora-home"]
local::xfs.perdev.log.writes["/dev/mapper/fedora-root"]
? 0.000 0.000
none count / second count / second
Mon Jun 5 12:28:45 ??
Mon Jun 5 12:28:50 0.000 0.000
Mon Jun 5 12:28:55 0.200 0.200
Mon Jun 5 12:29:00 6.800 1.000
```

The mentioned example displays the data on the `xfs.perdev.log` metric collected in an archive at a 5 second interval and display all the headers.

Additional resources

- Section 6.9, “Tools distributed with PCP”
- Section 6.8, “System services distributed with PCP”
- The `pmdumptext` man page.
- The `pmrep` man page.
- The `pmlogsummary` man page.
- The `pmlogger` man page.

### 6.5. 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 Section 6.2, “Installing and enabling PCP”.
- The `pmlogger` service is enabled. For more information, see Section 6.4.3, “Enabling the pmlogger service”.

**Procedure**

1. Install the following packages:
   a. Install the `pcp-system-tools`:
      ```
      # yum install pcp-system-tools
      ```
   b. Install the `pmda-postfix` package to monitor `postfix`:
      ```
      # yum install pcp-pmda-postfix postfix
      ```
c. Install the logging daemon:

```
# yum install rsyslog
```

d. Install the mail client for testing:

```
# yum 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**

- Section 6.9, "Tools distributed with PCP"
- Section 6.8, "System services distributed with PCP"
6.6. VISUALLY TRACING PCP LOG ARCHIVES WITH THE PCP CHARTS APPLICATION

After recording metric data, you can replay the PCP log archives as 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.

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.

Following are the several ways to customize the PCP Charts application interface to display the data from the performance metrics:

- line plot
- bar graphs
- utilization graphs

Prerequisites

- PCP is installed. For more information, see Section 6.2, “Installing and enabling PCP”.

- Logged performance data with the pmlogger. For more information, see Section 6.4, “Logging performance data with pmlogger”.

- Install the pcp-gui package:

  ```shell
  # yum install pcp-gui
  ```

Procedure

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

  ```shell
  # pmchart
  ```
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. Go to File → 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 → Export to save an image of the current view.
   - Click Record → Start to start a recording. Click Record → 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 → 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

- Section 6.9, “Tools distributed with PCP”
- The `pmchart` man page.
- The `pmtime` man page.
6.7. XFS FILE SYSTEM PERFORMANCE ANALYSIS 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 PCP.

6.7.1. Installing XFS PMDA manually

If the XFS PMDA is not listed in PCP configuration readout, install the PMDA agent manually.

Procedure

1. Navigate to the xfs directory:

   ```bash
   # cd /var/lib/pcp/pmdas/xfs/
   ```

2. Install the XFS PMDA manually:

   ```bash
   xfs]
   # ./Install
   ```
   
   You will need to choose an appropriate configuration for install of the “xfs” Performance Metrics Domain Agent (PMDA).

   ```
   collector   collect performance statistics on this system
   monitor     allow this system to monitor local and/or remote systems
   both        collector and monitor configuration for this system
   ```

   Please enter c(ollector) or m(onitor) or (both) [b]

   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 xfs metrics have appeared ... 149 metrics and 149 values

3. Select the intended PMDA role by entering c for collector, m for monitor, or b for both. The PMDA installation script prompts you to specify one of the following PMDA roles:

   - The **collector** role allows the collection of performance metrics on the current system
   - The **monitor** role allows the system to monitor local systems, remote systems, or both
     The default option is both **collector** and **monitor**, which allows the XFS PMDA to operate correctly in most scenarios.

Additional resources

- Section 6.9, “Tools distributed with PCP”
- The `pmcd` man page.

6.7.2. Examining XFS performance metrics with pminfo

The `pminfo` tool displays information about the available performance metrics. This procedure displays a list of all available metrics provided by the XFS PMDA.
Prerequisites

- PCP is installed. For more information, see Section 6.2, “Installing and enabling PCP”.

Procedure

1. Display the list of all available metrics provided by the XFS PMDA:

   ```
   # pminfo xfs
   ```

2. 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
     ```

Additional resources

- Section 6.10, “PCP metric groups for XFS”.

- The `pminfo` man page.

6.7.3. 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.

Prerequisites

- PCP is installed. For more information, see Section 6.2, “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
   ```

3. View the information after resetting the metric:

   ```
   $ pminfo --fetch xfs.write
   xfs.write
   value 0
   ```

Additional resources

- Section 6.9, "Tools distributed with PCP"
- Section 6.10, "PCP metric groups for XFS"
- The `pmstore` man page.
- The `pminfo` man page.

### 6.7.4. Examining XFS metrics available per file system

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.

Prerequisites

- PCP is installed. For more information, see Section 6.2, “Installing and enabling PCP”.

Procedure

- 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

- Section 6.11, “Per-device PCP metric groups for XFS”.
- The `pminfo` man page.

6.8. SYSTEM SERVICES DISTRIBUTED WITH PCP

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmcd</td>
<td>The Performance Metric Collector Daemon (PMCD).</td>
</tr>
<tr>
<td>pmie</td>
<td>The Performance Metrics Inference Engine.</td>
</tr>
<tr>
<td>pmlogger</td>
<td>The performance metrics logger.</td>
</tr>
<tr>
<td>pmmgr</td>
<td>Manages a collection of PCP daemons for a set of discovered local and remote hosts running the Performance Metric Collector Daemon (PMCD) according to zero or more configuration directories.</td>
</tr>
<tr>
<td>pmproxy</td>
<td>The Performance Metric Collector Daemon (PMCD) proxy server.</td>
</tr>
</tbody>
</table>

6.9. TOOLS DISTRIBUTED WITH PCP

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pcp</td>
<td>Displays the current status of a Performance Co-Pilot installation.</td>
</tr>
<tr>
<td>pcp-atop</td>
<td>Shows the system-level occupation of the most critical hardware resources from the performance point of view: CPU, memory, disk, and network.</td>
</tr>
<tr>
<td>pcp-dstat</td>
<td>Displays metrics of one system at a time. To display metrics of multiple systems, use --host option.</td>
</tr>
<tr>
<td>pmchart</td>
<td>Plots performance metrics values available through the facilities of the Performance Co-Pilot.</td>
</tr>
<tr>
<td>pmclient</td>
<td>Displays high-level system performance metrics by using the Performance Metrics Application Programming Interface (PMAPI).</td>
</tr>
<tr>
<td>pmcollectl</td>
<td>Collects and displays system-level data, either from a live system or from a Performance Co-Pilot archive file.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>pmconfig</td>
<td>Displays the values of configuration parameters.</td>
</tr>
<tr>
<td>pmdbg</td>
<td>Displays available Performance Co-Pilot debug control flags and their values.</td>
</tr>
<tr>
<td>pmdiff</td>
<td>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.</td>
</tr>
<tr>
<td>pmdumplog</td>
<td>Displays control, metadata, index, and state information from a Performance Co-Pilot archive file.</td>
</tr>
<tr>
<td>pmdumptext</td>
<td>Outputs the values of performance metrics collected live or from a Performance Co-Pilot archive.</td>
</tr>
<tr>
<td>pmerr</td>
<td>Displays available Performance Co-Pilot error codes and their corresponding error messages.</td>
</tr>
<tr>
<td>pmfind</td>
<td>Finds PCP services on the network.</td>
</tr>
<tr>
<td>pmie</td>
<td>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.</td>
</tr>
<tr>
<td>pmieconf</td>
<td>Displays or sets configurable pmie variables.</td>
</tr>
<tr>
<td>pminfo</td>
<td>Displays information about performance metrics. The metrics are collected either from a live system, or from a Performance Co-Pilot archive file.</td>
</tr>
<tr>
<td>pmiostat</td>
<td>Reports I/O statistics for SCSI devices (by default) or device-mapper devices (with the -x dm option).</td>
</tr>
<tr>
<td>pmic</td>
<td>Interactively configures active pmlogger instances.</td>
</tr>
<tr>
<td>pmlogcheck</td>
<td>Identifies invalid data in a Performance Co-Pilot archive file.</td>
</tr>
<tr>
<td>pmlogconf</td>
<td>Creates and modifies a pmlogger configuration file.</td>
</tr>
<tr>
<td>pmloglabel</td>
<td>Verifies, modifies, or repairs the label of a Performance Co-Pilot archive file.</td>
</tr>
<tr>
<td>pmlogsummary</td>
<td>Calculates statistical information about performance metrics stored in a Performance Co-Pilot archive file.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>pmprobe</td>
<td>Determines the availability of performance metrics.</td>
</tr>
<tr>
<td>pmrep</td>
<td>Reports on selected, easily customizable, performance metrics values.</td>
</tr>
<tr>
<td>pmsocks</td>
<td>Allows access to a Performance Co-Pilot host through a firewall.</td>
</tr>
<tr>
<td>pmstat</td>
<td>Periodically displays a brief summary of system performance.</td>
</tr>
<tr>
<td>pmstore</td>
<td>Modifies the values of performance metrics.</td>
</tr>
<tr>
<td>pmtrace</td>
<td>Provides a command line interface to the trace Performance Metrics Domain Agent (PMDA).</td>
</tr>
<tr>
<td>pmval</td>
<td>Displays the current value of a performance metric.</td>
</tr>
</tbody>
</table>

### 6.10. PCP METRIC GROUPS FOR XFS

<table>
<thead>
<tr>
<th>Metric Group</th>
<th>Metrics provided</th>
</tr>
</thead>
<tbody>
<tr>
<td>xfs.*</td>
<td>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.</td>
</tr>
<tr>
<td>xfs.allocs.*</td>
<td>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.</td>
</tr>
<tr>
<td>xfs.alloc_btree.*</td>
<td></td>
</tr>
<tr>
<td>xfs.block_map.*</td>
<td>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.</td>
</tr>
<tr>
<td>xfs.bmap_tree.*</td>
<td></td>
</tr>
<tr>
<td>xfs.dir_ops.*</td>
<td>Counters for directory operations on XFS file systems for creation, entry deletions, count of “getdent” operations.</td>
</tr>
<tr>
<td>xfs.transactions.*</td>
<td>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.</td>
</tr>
<tr>
<td>Metric Group</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>xfs.inode_ops.*</td>
<td>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.</td>
</tr>
<tr>
<td>xfs.log.*</td>
<td>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.</td>
</tr>
<tr>
<td>xfs.log_tail.*</td>
<td>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.</td>
</tr>
<tr>
<td>xfs.xstrat.*</td>
<td>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.</td>
</tr>
<tr>
<td>xfs.attr.*</td>
<td>Counts for the number of attribute get, set, remove and list operations over all XFS file systems.</td>
</tr>
<tr>
<td>xfs.quota.*</td>
<td>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.</td>
</tr>
<tr>
<td>xfs.buffer.*</td>
<td>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.</td>
</tr>
<tr>
<td>xfs.btree.*</td>
<td>Metrics regarding the operations of the XFS btree.</td>
</tr>
<tr>
<td>xfs.control.reset</td>
<td>Configuration metrics which are used to reset the metric counters for the XFS stats. Control metrics are toggled by means of the pmstore tool.</td>
</tr>
</tbody>
</table>

### 6.11. PER-DEVICE PCP METRIC GROUPS FOR XFS

<table>
<thead>
<tr>
<th>Metric Group</th>
<th>Metrics provided</th>
</tr>
</thead>
<tbody>
<tr>
<td>xfs.perdev.*</td>
<td>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.</td>
</tr>
<tr>
<td>Metric Code</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>xfs.perdev.allocs.*</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>xfs.perdev.alloc_btree.*</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>xfs.perdev.block_map.*</code></td>
<td>Counters for directory operations of XFS file systems for creation, entry deletions, count of &quot;getdent&quot; operations.</td>
</tr>
<tr>
<td><code>xfs.perdev.block_map.*</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>xfs.perdev.xstrat.*</code></td>
<td>Counters 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.</td>
</tr>
<tr>
<td><code>xfs.perdev.attr.*</code></td>
<td>Counts for the number of attribute get, set, remove and list operations over all XFS file systems.</td>
</tr>
<tr>
<td><code>xfs.perdev.quota.*</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>xfs.perdev.buffer.*</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>xfs.perdev.btree.*</code></td>
<td>Metrics regarding the operations of the XFS btree.</td>
</tr>
</tbody>
</table>
CHAPTER 7. SETTING UP GRAPHICAL REPRESENTATION OF PCP METRICS

Using a combination of **redis**, **pcp**, **bptrace**, **vector** and **grafana** provides graphs, based on the live data or data collected by Performance Co-Pilot (PCP). It allows to access graphs of PCP metrics using a web browser.

- The PCP is a generic framework which collects, monitors, analyzes, and stores performance related metrics. For more information about the PCP and its components, see Monitoring performance with Performance Co-Pilot.
- Redis is an **in-memory-database**. It is used to store data from the archived files which is easily accessible for the graph generation by the Grafana application.
- Bptrace allows access to the live data from sources which are not available as normal data from the **pmlogger** or archives.
- Vector provides access to the live data but does not provide access to data from past.
- Grafana generates graphs that are accessible from a browser. The **grafana-server** is a component that 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.

7.1. SETTING UP PCP ON A SYSTEM

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:
  
  ```
  # yum 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**

- The **pmlogger** man page.
- For more information about the PCP and its components, see Monitoring performance with Performance Co-Pilot.

7.2. SETTING UP A GRAFANA-SERVER
This procedure describes how to set up a **grafana-server**. It is a back-end server for the Grafana dashboard. The **grafana-server** component listens on the interface, and provides web services which are accessed via browser.

**Prerequisites**

- The **pcp-zeroconf** package is installed. For more information, see Section 7.1, “Setting up PCP on a system”.

**Procedure**

1. Install the following packages:
   ```
   # yum install grafana grafana-pcp
   ```

2. Restart and enable the following service:
   ```
   # systemctl restart grafana-server
   # systemctl enable grafana-server
   ```

**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,f=7))
  ```

- Ensure that the Performance Co-Pilot plugin is installed:
  ```
  # grafana-cli plugins ls | grep performancecopilot-pcp-app
  performancecopilot-pcp-app @ 1.0.5
  ```

**Additional resources**

- The **pmproxy** man page.

- The **grafana-server** man page.

**7.3. ACCESSING THE GRAFANA WEB UI**

This procedure describes how to access the Grafana web interface and generate graphs using different types of data sources.

**Prerequisites**

- The **pcp-zeroconf** package is installed. For more information, see Section 7.1, “Setting up PCP on a system”.

- The **grafana-server** is configured. For more information, see Section 7.2, “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 **username** and **password** field.

   **Figure 7.1. Grafana login page**

3. To create a secured account, Grafana prompts to set a new **password**.

4. From the menu, hover over the **configuration** icon > click **Plugins** > in the **Filter by name or type**, type performance co-pilot > click **Performance Co-Pilot** plugin and > click **Enable** to enable the **grafana-pcp** plugin.

   **Figure 7.2. Home Dashboard**
NOTE

The top corner of the screen has a similar icon, but it controls the general Dashboard settings.

5. Click the icon to view the Home Dashboard.

6. In the Home Dashboard, click Add data source to add PCP Redis, PCP bpftrace, and PCP Vector data sources. For more information on how to add PCP Redis, PCP bpftrace, and PCP Vector data sources, see:
   - Section 7.4, “Adding PCP Redis as a data source”
   - Section 7.6, “Adding PCP bpftrace as a data source”
   - Section 7.7, “Adding PCP Vector as a data source”.

7. Optional: In the pane, hover on the icon to change the Preferences or to Sign out.

Additional resources

- The grafana-cli man page.
- The grafana-server man page.

7.4. ADDING PCP REDIS AS A DATA SOURCE

The PCP Redis data source visualizes everything which the archive contains and queries time series capability provided by the pmseries language. It analyzes data across multiple hosts. This procedure describes how to add PCP Redis as a data source and how to view the dashboard with an overview of useful metrics.

Prerequisites

- The grafana-server is accessible. For more information, see Section 7.3, “Accessing the Grafana web UI”.

Procedure

1. Install the redis package:

   ```
   # yum install redis
   ```

2. Start and enable the following services:

   ```
   # systemctl start pmproxy redis
   # systemctl enable pmproxy redis
   ```
3. Log into the Grafana web UI. For more information, see Section 7.3, "Accessing the Grafana web UI".

4. Click the icon > click **Add data source** > in the **Filter by name or type**, type redis > and click **PCP Redis** > in the **URL** field, accept the given suggestion http://localhost:44322 and > click **Save & Test**

![Figure 7.3. Adding PCP Redis in the data source](image)

5. In the pane, hover on the filter icon > click **Manage** > in the **Filter Dashboard by name**, type pcp redis > select **PCP Redis Host Overview** to see a dashboard with an overview of any useful metrics.

![Figure 7.4. PCP Redis Host Overview](image)
6. In the pane, hover on the icon > click Dashboard option > click Add Query > from the Query list, select the PCP Redis and > in the text field of A, enter metric, for example, kernel.all.load to visualize the kernel load graph.

Figure 7.5. PCP Redis query

Additional resources

- The pmseries man page.

7.5. SETTING UP AUTHENTICATION BETWEEN PCP COMPONENTS

PCP supports the scram-sha-256 authentication mechanism through the Simple Authentication Security Layer (SASL) framework. This procedure describes how to setup authentication using the scram-sha-256 authentication mechanism.

NOTE

From Red Hat Enterprise Linux 8.3, PCP supports the scram-sha-256 authentication mechanism.

Prerequisites

- Install the sasl framework for the scram-sha-256 authentication mechanism:

  # yum install cyrus-sasl-scram cyrus-sasl-lib

Procedure

1. 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

2. Create a new user:

   # useradd -r metrics

   Replace metrics by your user name.

3. Add the created user in the user database:

   # saslpasswd2 -a pmcd metrics

   Password:
   Again (for verification):

4. Set the permissions of the user database:

   # chown root:pcp /etc/pcp/passwd.db
   # chmod 640 /etc/pcp/passwd.db

5. 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

- For more information, see the saslauthd, pminfo, and sha256 man pages.

- For more information on setting up authentication between PCP components in RHEL 8.2, see How can I setup authentication between PCP components, like PMDAs and pmcd? knowledgebase article.

7.6. ADDING PCP BPFTRACE AS A DATA SOURCE

The bpftrace agent enables system introspection using the bpftrace scripts, which uses the enhanced Berkeley Packet Filter (eBPF) to gather metrics from the kernel and user-space tracepoints. This procedure describes how to add PCP bpftrace as a data source and how to view the dashboard with an overview of any useful metrics.

Prerequisites

- The grafana-server is accessible. For more information, see Section 7.3, "Accessing the Grafana web UI".
The **scram-sha-256** authentication mechanism is configured. For more information, see Section 7.5, "Setting up authentication between PCP components".

**Procedure**

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

   ```
   # yum install pcp-pmda-bpftrace
   ```

2. Edit the **bpftrace.conf** file and add your user, which you have created in the Section 7.5, "Setting up authentication between PCP components":

   ```
   # vi /var/lib/pcp/pmdas/bpftrace/bpftrace.conf
   
   [dynamic_scripts]
   enabled = true
   auth_enabled = true
   allowed_users = root,metrics
   
   Replace metrics by your user name.
   ```

3. Install the **bpftrace** PMDA:

   ```
   # cd /var/lib/pcp/pmdas/bpftrace/
   # ./Install
   
   Updating the Performance Metrics Name Space (PMNS) ...
   
   Terminate PMDA if already installed ...
   
   Updating the PMCD control file, and notifying PMCD ...
   
   Check bpftrace metrics have appeared ... 7 metrics and 6 values
   
   The **pmda-bpftrace** is now installed, and can only be used after authenticating your user.
   ```

4. Log into the Grafana web UI. For more information, see Section 7.3, "Accessing the Grafana web UI".

5. Click the icon > click Add data source > in the Filter by name or type, type bpftrace > and click PCP bpftrace > in the URL field, accept the given suggestion http://localhost:44322.

   Select the Basic Auth option > add the created user credentials in the User and Password field and > click Save & Test
6. In the pane, hover on the filter icon > click Manage > in the Filter Dashboard by name, type pcp bpftrace > select PCP bpftrace System Analysis to see a dashboard with an overview of useful metrics.

![Figure 7.7. PCP bpftrace System Analysis](image)

Additional resources

- The `pmdabpftrace` man page.
- The `bpftrace` man page.

### 7.7. ADDING PCP VECTOR AS A DATA SOURCE

The PCP Vector data source displays live metrics and uses the `pcp` metrics. It analyzes data for individual hosts. This procedure describes how to add a PCP Vector as a data source and how to view the dashboard with an overview of any useful metrics.
Prerequisites

- The **grafana-server** is accessible. For more information, see Section 7.3, "Accessing the Grafana web UI".

Procedure

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

   ```
   # yum 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
   ```

3. Log into the Grafana web UI. For more information, see Section 7.3, "Accessing the Grafana web UI".

4. Click the icon > click **Add data source** > in the **Filter by name or type**, type **vector** > and click **PCP Vector** > in the **URL** field, accept the given suggestion **http://localhost:44322** and > click **Save & Test**

Figure 7.8. Adding PCP Vector in the data source
5. In the pane, hover on the ➕ icon > click Manage > in the Filter Dashboard by name, type pcp vector > select PCP Vector Host Overview to see a dashboard with an overview of useful metrics.

**Figure 7.9. PCP Vector Host Overview**

![](image)

**Additional resources**

- The `pmdabcc` man page.
CHAPTER 8. OPTIMIZING THE SYSTEM PERFORMANCE USING THE WEB CONSOLE

Learn how to set a performance profile in the RHEL 8 web console to optimize the performance of the system for a selected task.

8.1. PERFORMANCE TUNING OPTIONS IN THE WEB CONSOLE

Red Hat Enterprise Linux 8 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

- For details about the tuned service, see Monitoring and managing system status and performance.

8.2. SETTING A PERFORMANCE PROFILE IN THE WEB CONSOLE

This procedure uses the web console to optimize the system performance for a selected task.

Prerequisites

- The web console is installed and accessible. For details, see Installing the web console.

Procedure

1. Log into the RHEL 8 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.
CHAPTER 9. 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 Section 9.6, “Setting the disk scheduler using Tuned”
- Set the scheduler using udev, as described in Section 9.7, “Setting the disk scheduler using udev rules”
- Temporarily change the scheduler on a running system, as described in Section 9.8, “Temporarily setting a scheduler for a specific disk”

9.1. DISK SCHEDULER CHANGES IN RHEL 8

In RHEL 8, 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 RHEL 7 and earlier versions, have been removed.

9.2. AVAILABLE DISK SCHEDULERS

The following multi-queue disk schedulers are supported in RHEL 8:

- **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.
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.

9.3. 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 9.1. Disk schedulers for different use cases

<table>
<thead>
<tr>
<th>Use case</th>
<th>Disk scheduler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional HDD with a SCSI interface</td>
<td>Use <em>mq-deadline</em> or <em>bfq</em>.</td>
</tr>
<tr>
<td>High-performance SSD or a CPU-bound system with fast storage</td>
<td>Use <em>none</em>, especially when running enterprise applications. Alternatively, use <em>kyber</em>.</td>
</tr>
<tr>
<td>Desktop or interactive tasks</td>
<td>Use <em>bfq</em>.</td>
</tr>
<tr>
<td>Virtual guest</td>
<td>Use <em>mq-deadline</em>. With a host bus adapter (HBA) driver that is multi-queue capable, use <em>none</em>.</td>
</tr>
</tbody>
</table>

9.4. 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.

9.5. 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 ([ ]).

9.6. 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 Section 2.6, "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 Section 2.3, "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:

   - Optional: Include an existing profile:
9.7. 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:

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

**Procedure**

1. Find the system unique identifier of the block device:

```
5. Enable your profile:

   # tuned-adm profile \textit{my-profile}

6. Verify that the Tuned profile is active and applied:

   $ tuned-adm active
   Current active profile: \textit{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

- For more information on creating a Tuned profile, see Chapter 3, Customizing Tuned profiles.
$ udevadm info --name=/dev/device | grep -E '(WWN|SERIAL)'
E: ID_WWN=0x5002538d00000000
E: ID_SERIAL=Generic_ SD_MMC_201205010309000000-0:0
E: 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.

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"
```

- 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
```

5. Verify the active scheduler:

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

9.8. 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 10. TUNING THE PERFORMANCE OF A SAMBA SERVER

This chapter describes 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
  See Using Samba as a server.

10.1. 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 support 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
   ```

10.2. TUNING SHOARES 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.

**10.3. 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 11. 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 8, so that your hardware infrastructure resources can be used as efficiently as possible.

11.1. 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 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 8 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.
11.2. 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.

Red Hat recommends using the following profiles when using virtualization in RHEL 8:

- For RHEL 8 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 8 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.

```bash
# 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.

```bash
# 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
For more information about tuned and tuned profiles, see Monitoring and managing system status and performance.

11.3. 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.

11.3.1. 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 membaloop device:

    ```
    # virsh dumpxml testguest | grep membaloop
    <membaloop model="virtio">
    </membaloop>
    ```

    If this commands displays any output and the model is not set to none, the membaloop 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 membaloop device is present.

- To use the web console to manage VMs, install the web console VM plug-in.

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 a row with the name of the VMs for which you want to view and adjust the allocated memory.

   The row expands to reveal the Overview pane with basic information about the selected VMs.

3. Click the value of the Memory line in the Overview pane.

   The Memory Adjustment dialog appears.
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. Increasing this value improves the performance potential of the VM, and reducing the value lowers the performance footprint the VM has on your host. Adjusting maximum memory allocation is only possible on a shut-off VM.

   - **Current allocation** - Sets the actual amount of memory allocated to the VM. You can adjust the value to regulate the memory available to the VM for its processes. This value cannot exceed the maximum allocation value.

5. Click **Save**.
   The memory allocation of the VM is adjusted.

Additional resources

- For instructions for adjusting VM memory setting using the command-line interface, see Section 11.3.2, “Adding and removing virtual machine memory using the command-line interface”.

- To optimize how the VM uses the allocated memory, you can modify your vCPU setting. For more information, see Section 11.5, “Optimizing virtual machine CPU performance”.

11.3.2. 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:

```bash
# virsh dumpxml testguest | grep membalo\n<memballoon model='\n```

```bash
virtio'>
</membalo\n```
If this command 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.

**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.
   ```

1. **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

- For instructions for adjusting VM memory setting using the web console, see Section 11.3.1, “Adding and removing virtual machine memory using the web console”.
- To optimize how the VM uses the allocated memory, you can modify your vCPU setting. For more information, see Section 11.5, “Optimizing virtual machine CPU performance”.

11.3.3. Additional resources

- 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.
  Note that removing a memory device from a VM, also known as memory hot unplug, is not supported in RHEL 8, and Red Hat highly discourages its use.

11.4. 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.

11.4.1. 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:
   ```bash
   # virsh dumpxml VM-name
   ```

   `<domain>`:
   ```xml
   [...] ```
2. Edit the I/O weight of a specified device:

```
# virsh blkiotune VM-name --device-weights device, I/O-weight
```

For example, the following changes the weight of the /dev/sda device in the liftrul VM to 500.

```
# virsh blkiotune liftrul --device-weights /dev/sda, 500
```

### 11.4.2. 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
```
3. Set I/O limits for the block device using the `virsh blkiotune` command.

   ```bash
   # 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.

   ```bash
   # 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 on unsupported features when using RHEL 8 as a VM host, see [Unsupported features in RHEL 8 virtualization](#).

### 11.4.3. 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:

  ```xml
  <controller type='scsi' index='0' model='virtio-scsi'>
    <driver queues='N'/>
  </controller>
  ```

### 11.5. 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](https://access.redhat.com/documentation/rhel/8/html/system_administration_guide/using_the_command_line) or the [web console](https://access.redhat.com/documentation/rhel/8/html/system_administration_guide/web_console).
2. Ensure that the vCPU model is aligned with the CPU model of the host. For example, to set the testguest1 VM to use the CPU model of the host:

```
# virt-xml testguest1 --edit --cpu host-model
```

3. Deactivate 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 Section 11.5.3, “Configuring NUMA in a virtual machine” and Section 11.5.4, “Sample vCPU performance tuning scenario”.

### 11.5.1. 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 8, 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 testguest VM:

```
# virsh vcpucount testguest
```

```
maximum  config     4
maximum  live       2
current   config    2
current   live      1
```

This output indicates that testguest 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 testguest 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 testguest VM to 8:

   ```
   # virsh setvcpus testguest 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 testguest VM to 4:
# virsh setvcpus testguest 4 --live

This increases the VM’s performance and host load footprint of testguest until the VM’s next boot.

- To permanently decrease the number of vCPUs attached to the testguest VM to 1:

  # virsh setvcpus testguest 1 --config

  This decreases the VM’s performance and host load footprint of testguest 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 testguest
  
  maximum  config  8  
  maximum  live  4  
  current  config  1  
  current  live  4  

**Additional resources**

- For information on adding and removing vCPUs using the web console, see Section 11.5.2, "Managing virtual CPUs using the web console”.

### 11.5.2. Managing virtual CPUs using the web console

Using the RHEL 8 web console, you can review and configure virtual CPUs used by virtual machines (VMs) to which the web console is connected.

**Prerequisites**

- To use the web console to manage VMs, install the web console VM plug-in.

**Procedure**

1. In the Virtual Machines interface, click a row with the name of the VMs for which you want to view and configure virtual CPU parameters.
   The row expands to reveal the Overview pane with basic information about the selected VMs, including the number of virtual CPUs, and controls for shutting down and deleting the VM.

2. Click the number of vCPUs in the Overview pane.
   The vCPU details dialog appears.
NOTE

The warning in the vCPU details dialog only appears after the virtual CPU settings are changed.

3. 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, Red Hat recommends keeping the default values.

4. 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:

- For information on managing your vCPUs using the command-line interface, see Section 11.5.1, “Adding and removing virtual CPUs using the command-line interface”.
11.5.3. 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 8 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
   ```
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
```

3. 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
```

Additional resources

- Note that for best performance results, it is recommended to use all of the manual tuning methods listed above. For an example of such a configuration, see Section 11.5.4, “Sample vCPU performance tuning scenario”.

- To see the current NUMA configuration of your system, you can use the `numastat` utility. For details on using `numastat`, see Section 11.7, “Virtual machine performance monitoring tools”.

- NUMA tuning is currently not possible to perform on IBM Z hosts. For further information, see How virtualization on IBM Z differs from AMD64 and Intel 64.

11.5.4. Sample vCPU performance tuning scenario

To obtain the best vCPU performance possible, Red Hat recommends using manual `vcpupin`, `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 on 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>
    </cell>
    <cell id="1">
      <memory unit="KiB">15624346</memory>
      <sibling id="0" value="10"/>
      <sibling id="1" value="21"/>
    </cell>
  </cells>
  <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>
</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 Additional Resources section below.

The XML configuration for steps a. and b. can look similar to:

```
<cpuset>
    <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'/>
</cpuset>
```

c. Set the VM to use 1GiB 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:

```
<memnode cellid="0" mode="strict" nodeset="0"/>
<memnode cellid="1" mode="strict" nodeset="1"/>
```

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"/>
```

6. The resulting XML configuration of the VM should include a section similar to the following:

```
[...]
<cpus>
	<cpupin vcpu="0" cpuset="1"/>
	<cpupin vcpu="1" cpuset="4"/>
	<cpupin vcpu="2" cpuset="2"/>
	<cpupin vcpu="3" cpuset="5"/>
	<cpupin vcpu="4" cpuset="7"/>
	<cpupin vcpu="5" cpuset="10"/>
	<cpupin vcpu="6" cpuset="8"/>
	<cpupin vcpu="7" cpuset="11"/>
	<emulatorpin cpuset="6,9"/>
</cpus>
```
7. **Optional:** Test the performance of the VM using the applicable tools and utilities to evaluate the impact of the VM’s optimization.

**Additional resources**

- The following tables illustrate the connections between the vCPUs and the host CPUs they should be pinned to:

**Table 11.1. Host topology**

<table>
<thead>
<tr>
<th>CPU threads</th>
<th>0</th>
<th>3</th>
<th>1</th>
<th>4</th>
<th>2</th>
<th>5</th>
<th>6</th>
<th>9</th>
<th>7</th>
<th>10</th>
<th>8</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sockets</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMA nodes</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 11.2. VM topology**

<table>
<thead>
<tr>
<th>vCPU threads</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sockets</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMA nodes</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 11.3. Combined host and VM topology**

<table>
<thead>
<tr>
<th>vCPU threads</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host CPU threads</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>Cores</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sockets</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMA nodes</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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:

```xml
<vcpupin vcpu='0' cpuset='1'/>
<vcpupin vcpu='1' cpuset='4'/>
<vcpupin vcpu='2' cpuset='2'/>
<vcpupin vcpu='3' cpuset='5'/>
```

11.5.5. Deactivating kernel same-page merging

Although kernel same-page merging (KSM) improves memory density, it increases CPU utilization, and might adversely affect overall performance depending on the workload. In such cases, you can improve the virtual machine (VM) performance by deactivating KSM.

Depending on your requirements, you can either deactivate KSM for a single session or persistently.

Procedure

- To deactivate KSM for a single session, use the `systemctl` utility to stop `ksm` and `ksmtuned` services.

  ```bash
  # systemctl stop ksm
  # systemctl stop ksmtuned
  ```

- To deactivate KSM persistently, use the `systemctl` utility to disable `ksm` and `ksmtuned` services.

  ```bash
  # 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:

```bash
# 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.

11.6. 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 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 128
```

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

- For additional information on virtual network connection types and tips for usage, see Understanding virtual networking.
11.7. 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 8 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 8 host. To do so:

1. On the host, install the `perf` package:
   ```bash
   # yum install perf
   ```

2. 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.

3. 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.

   ```bash
   # perf kvm stat report
   ```

   Analyze events for all VMs, all VCPUs:
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 on 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:

```bash
# 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:

```bash
# 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
```
11.8. RELATED INFORMATION

- When using Windows as the guest operating system of your VM, Red Hat recommends applying additional optimization measures. For details, see Optimizing Windows virtual machines.
CHAPTER 12. MANAGING POWER CONSUMPTION WITH POWERTOP

As a system administrator, you can use the PowerTOP tool to analyze and manage power consumption.

12.1. 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 8 uses version 2.x of PowerTOP.

12.2. USING POWERTOP

Prerequisites

- To be able to use PowerTOP, make sure that the powertop package has been installed on your system:

  # yum install powertop

12.2.1. Starting PowerTOP

Procedure

- To run PowerTOP, use the following command:

  # powertop

IMPORTANT
Laptops should run on battery power when running the powertop command.

12.2.2. 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.

12.2.3. 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
  ```

12.2.4. Related information

For more details on how to use **PowerTOP**, see the **powertop** man page.

12.3. **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**

You can use the **Tab** and **Shift+Tab** keys to cycle through these tabs.

12.3.1. 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

Wakes up 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

12.3.2. 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.

12.3.3. The Device stats tab

The Device stats tab provides similar information to the Overview tab but only for devices.

12.3.4. 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.

Figure 12.1. PowerTOP output
12.4. 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.

12.5. OPTIMIZING POWER CONSUMPTION

To optimize power consumption, you can use either the powertop service or the powertop2tuned utility.

12.5.1. 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
  ```

12.5.2. 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.

12.5.3. Optimizing power consumption using the powertop2tuned utility

**Prerequisites**
The `powertop2tuned` utility is installed on the system:

```
# yum 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
```

**12.5.4. 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 13. GETTING STARTED WITH PERF

As a system administrator, you can use the `perf` tool to collect and analyze performance data of your system.

13.1. 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.

13.2. INSTALLING PERF

This procedure installs the `perf` user-space tool.

Procedure

- Install the `perf` tool:

  ```bash
  # yum install perf
  ```

13.3. COMMON PERF COMMANDS

This section provides an overview of commonly used `perf` commands.

Commonly used `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

- To list additional subcommand options of the subcommands and their descriptions, add the `-h` option to the target command.

13.4. REAL TIME PROFILING OF CPU USAGE 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.

13.4.1. 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.

13.4.2. 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
```

Example 13.1. Perf top output

```
PerfTop: 20806 irqs/sec kernel:57.3% exact: 100.0% lost: 0/0 drop: 0/0 [4000Hz cycles], (all, 8 CPUs)

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    [] 0x000000000241d1cd
0.60% [kernel]  [k] do_sys_poll
0.58% [kernel]  [k] menu_select
0.56% [kernel]  [k] _raw_spin_lock_irqsave
0.55% perf        [] 0x0000000002ae0f3

In the previous example, the kernel function do_syscall_64 is using the most CPU time.

Additional resources

- The perf-top(1) man page.

13.4.3. Interpretation of perf top output

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 [.].

13.4.4. 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.

Additional Resources

- Enabling debugging with debugging information.

13.4.5. 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:

```
# subscription-manager repos --enable rhel-8-for-$(uname -i)-baseos-debug-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-baseos-source-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-appstream-debug-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-appstream-source-rpms
```

The `$(uname -i)` part is automatically replaced with a matching value for architecture of your system:

<table>
<thead>
<tr>
<th>Architecture name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>64-bit Intel and AMD</td>
<td>x86_64</td>
</tr>
<tr>
<td>64-bit ARM</td>
<td>aarch64</td>
</tr>
<tr>
<td>IBM POWER</td>
<td>ppc64le</td>
</tr>
<tr>
<td>64-bit IBM Z</td>
<td>s390x</td>
</tr>
</tbody>
</table>

### 13.4.6. 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](#).
- Channels providing `debuginfo` and `debugsource` packages must be configured and enabled on the system.

**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)
   
   (gdb) q
   ```

2. Exit GDB: type `q` and confirm with **Enter**.
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

- Red Hat Developer Toolset User Guide, section Installing Debugging Information
- How can I download or install debuginfo packages for RHEL systems? — Red Hat Knowledgebase solution

### 13.5. COUNTING EVENTS DURING PROCESS EXECUTION

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.

#### 13.5.1. 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.

#### 13.5.2. 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 13.2. 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 13.3. 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
13.5.3. Interpretation of perf stat output

perf stat executes a specified command and counts event occurrences during the command's 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.

13.5.4. 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:

```bash
$ 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**

- The perf-stat(1) man page.

13.6. 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.

13.6.1. 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.

### 13.6.2. 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:

  
  ```bash
  $ 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**

- The `perf-record(1)` man page.

### 13.6.3. 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:

  ```bash
  # 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**
13.6.4. 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 and user-space and the kernel-space simultaneously across all threads on a monitored CPU. By default, per-CPU mode monitors all online CPUs.

Prerequisites

- The perf user space tool is installed. For more information, see Installing perf.

Procedure

- Sample and record the performance data:

```bash
# 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

- The perf-record(1) man page.

13.6.5. 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

- The perf user space tool is installed. For more information, see Installing perf.

Procedure

- Sample and record performance data with the --call-graph option:

```bash
$ 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

- The `perf-record(1)` man page.

13.6.6. Analyzing perf.data with perf report

You can use `perf report` to display and analyze a `perf.data` file.

Prerequisites

- The `perf` user space tool is installed. For more information, see `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
```

<table>
<thead>
<tr>
<th>Samples: 2K of event 'cycles', Event count (approx.): 235462960</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overhead</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>2.36%</td>
</tr>
<tr>
<td>2.13%</td>
</tr>
<tr>
<td>2.13%</td>
</tr>
<tr>
<td>1.53%</td>
</tr>
<tr>
<td>1.17%</td>
</tr>
<tr>
<td>0.93%</td>
</tr>
<tr>
<td>0.89%</td>
</tr>
<tr>
<td>0.87%</td>
</tr>
<tr>
<td>0.86%</td>
</tr>
<tr>
<td>0.83%</td>
</tr>
<tr>
<td>0.63%</td>
</tr>
<tr>
<td>0.53%</td>
</tr>
<tr>
<td>0.53%</td>
</tr>
<tr>
<td>0.49%</td>
</tr>
<tr>
<td>0.48%</td>
</tr>
<tr>
<td>0.47%</td>
</tr>
<tr>
<td>0.45%</td>
</tr>
<tr>
<td>0.45%</td>
</tr>
<tr>
<td>g_type_check_instance_is_fundamentally_a</td>
</tr>
<tr>
<td>0.44%</td>
</tr>
<tr>
<td>0.41%</td>
</tr>
<tr>
<td>0.40%</td>
</tr>
<tr>
<td>0.39%</td>
</tr>
<tr>
<td>__raw_caller_save__pv_queued_spin_unlock</td>
</tr>
</tbody>
</table>
Additional resources

- The `perf-report(1)` man page.

### 13.6.7. 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.

### 13.6.8. 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.

Additional Resources

- Enabling debugging with debugging information.

**NOTE**

It is not necessary to re-run `perf record` after installing the `debuginfo` associated with an executable. Simply re-run `perf report`.

### 13.6.9. 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:

  ```
  # subscription-manager repos --enable rhel-8-for-$uname -i-baseos-debug-rpms
  # subscription-manager repos --enable rhel-8-for-$uname -i-baseos-source-rpms
  ```
# subscription-manager repos --enable rhel-8-for-$(uname -i)-appstream-debug-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-appstream-source-rpms

The $(uname -i) part is automatically replaced with a matching value for architecture of your system:

<table>
<thead>
<tr>
<th>Architecture name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>64-bit Intel and AMD</td>
<td>x86_64</td>
</tr>
<tr>
<td>64-bit ARM</td>
<td>aarch64</td>
</tr>
<tr>
<td>IBM POWER</td>
<td>ppc64le</td>
</tr>
<tr>
<td>64-bit IBM Z</td>
<td>s390x</td>
</tr>
</tbody>
</table>

13.6.10. 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.
- Channels providing debuginfo and debugsource packages must be configured and enabled on the system.

**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

- Red Hat Developer Toolset User Guide, section Installing Debugging Information
- How can I download or install debuginfo packages for RHEL systems? – Red Hat Knowledgebase solution
CHAPTER 14. MONITORING SYSTEM PERFORMANCE WITH PERF

As a system administrator you can use the `perf` tool to collect and analyze performance data of your system.

14.1. 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 and user-space and the kernel-space simultaneously across all threads on a monitored CPU. By default, per-CPU mode monitors all online CPUs.

**Prerequisites**

- The `perf` user space tool is installed. For more information, see Installing perf.

**Procedure**

- Sample and record the performance data:
  
  ```bash
  # 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**

- The `perf-record(1)` man page.

14.2. 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**

- The `perf` user space tool is installed. For more information, see Installing perf.

**Procedure**

- Sample and record performance data with the `--call-graph` option:
  
  ```bash
  $ 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
   - The `perf-record(1)` man page.

14.3. IDENTIFYING BUSY CPUs WITH PERF

When investigating performance issues on a system, you can use the `perf` tool to identify the busiest
CPUs in order to focus your efforts.

14.3.1. 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
   - The `perf` user space tool is installed. For more information, see 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
```

14.3.2. 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
   - The `perf` user space tool is installed. For more information, see 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

- For more information on recording a perf.data file, see Recording and analyzing performance profiles with perf.

14.3.3. 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

- The perf user space tool is installed. For more information, see 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.

14.4. MONITORING SPECIFIC CPUS WITH PERF

You can configure the perf tool to monitor only specific CPUs of interest.

14.4.1. 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

- The *perf* user space tool is installed. For more information, see 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.

14.4.2. 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

- The *perf* user space tool is installed. For more information, see 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.

14.5. 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
- The perf user space tool is installed. For more information, see 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 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
- For more information on recording a perf.data file, see Recording and analyzing performance profiles with perf.
- For more information on capturing call graph data with perf record, see Capturing call graph data with perf record.
You can use the `perf` tool to analyze a `perf.data` file that was generated on a different device.

**Prerequisites**

- The `perf` user space tool is installed. For more information, see [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`:

   ```bash
   # mkdir -p ~/.debug
   # tar xf perf.data.tar.bz2 -C ~/.debug
   ```

   **NOTE**
   
   The archive file might also be named `perf.data.tar.gz`.

3. Open the `perf.data` file for further analysis:

   ```bash
   # perf report
   ```
CHAPTER 15. MONITORING APPLICATION PERFORMANCE WITH PERF

This section describes how to use the perf tool to monitor application performance.

15.1. ATTACHING PERF RECORD TO A RUNNING PROCESS

Prerequisites
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
- The perf user space tool is installed. For more information, see 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 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.

15.2. 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
- The perf user space tool is installed. For more information, see 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**

- The *perf-record(1)* man page.

**15.3. ANALYZING PERF.DATA WITH PERF REPORT**

You can use *perf report* to display and analyze a *perf.data* file.

**Prerequisites**

- The *perf* user space tool is installed. For more information, see [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
  ```

**Example 15.1. Example output**

```
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
```
Additonal resources

- The `perf-report(1)` man page.
16.1. 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.

16.2. SAMPLING MEMORY ACCESS WITH PERF MEM

You can use `perf mem` 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:

   ```sh
   # 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:

   ```sh
   # 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
   ```
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:

```bash
# 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

<table>
<thead>
<tr>
<th>Overhead</th>
<th>Samples</th>
<th>Memory access</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.53%</td>
<td>9725</td>
<td>LFB or LFB hit</td>
</tr>
<tr>
<td>29.70%</td>
<td>12201</td>
<td>L1 or L1 hit</td>
</tr>
<tr>
<td>23.03%</td>
<td>9725</td>
<td>L3 or L3 hit</td>
</tr>
<tr>
<td>12.91%</td>
<td>2316</td>
<td>Local RAM or RAM hit</td>
</tr>
<tr>
<td>2.37%</td>
<td>743</td>
<td>L2 or L2 hit</td>
</tr>
<tr>
<td>0.34%</td>
<td>9</td>
<td>Uncached or N/A hit</td>
</tr>
<tr>
<td>0.10%</td>
<td>69</td>
<td>I/O or N/A hit</td>
</tr>
<tr>
<td>0.02%</td>
<td>825</td>
<td>L3 miss</td>
</tr>
</tbody>
</table>

Additional resources

- For an explanation of command options specific to the `mem` subcommand, see the `perf-mem(1)` man page.

### 16.3. 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 17. DETECTING FALSE SHARING WITH PERF C2C

17.1. 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

17.2. 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.

17.3. 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:

  ```shell
  # 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

- For an explanation of command options specific to the c2c subcommand, see the perf-c2c(1) man page.

17.4. VISUALIZING A PERF.DATA FILE RECORDED WITH PERF C2C RECORD
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 groupping : offset,pid,iaddr

Shared Data Cache Line Table

<table>
<thead>
<tr>
<th>Index</th>
<th>Cacheline</th>
<th>records</th>
<th>Hitm</th>
<th>Total</th>
<th>Rmt</th>
<th>Total</th>
<th>L1Hit</th>
<th>L1Miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x602180</td>
<td>149904</td>
<td>77.09%</td>
<td>12103</td>
<td>2269</td>
<td>9834</td>
<td>109504</td>
<td>109036</td>
</tr>
<tr>
<td>1</td>
<td>0x602100</td>
<td>12128</td>
<td>22.20%</td>
<td>3951</td>
<td>1119</td>
<td>2832</td>
<td>0</td>
<td>65</td>
</tr>
<tr>
<td>200</td>
<td>0xffff883ffb6a7e80</td>
<td>260</td>
<td>0.09%</td>
<td>15</td>
<td>3</td>
<td>12</td>
<td>161</td>
<td>97</td>
</tr>
<tr>
<td>3</td>
<td>0xffffffff81aec00</td>
<td>157</td>
<td>0.07%</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>25</td>
<td>0xffffffff81e3f540</td>
<td>179</td>
<td>0.06%</td>
<td>9</td>
<td>1</td>
<td>8</td>
<td>117</td>
<td>97</td>
</tr>
</tbody>
</table>

Shared Cache Line Distribution Pareto

<table>
<thead>
<tr>
<th>Num</th>
<th>Rmt</th>
<th>Lcl</th>
<th>L1 Hit</th>
<th>L1 Miss</th>
<th>Offset</th>
<th>Pid</th>
<th>Code address</th>
<th>rmt hitm</th>
<th>lcl</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>9834</td>
<td>2269</td>
<td>109036</td>
<td>468</td>
<td>0x602180</td>
<td>65.51%</td>
<td>55.88%</td>
<td>75.20%</td>
<td>0.00%</td>
</tr>
<tr>
<td>26039</td>
<td>26017</td>
<td>9</td>
<td>(.)</td>
<td>read_write_func no_false_sharing.exe</td>
<td>false_sharing_example.c:144</td>
<td>0[0-1,4]</td>
<td>1[24-25,120]</td>
<td>2[48,54]</td>
<td>3[169]</td>
</tr>
</tbody>
</table>

CHAPTER 17. DETECTING FALSE SHARING WITH PERF C2C
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  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  0x400bdd  2325  1897  1828  2  read_write_func no_false_sharing.exe false_sharing_example.c:162   0{96}  3{171}  17.5. 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 HITMs 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.

17.6. DETECTING FALSE SHARING WITH PERF C2C

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)"

   ```
   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

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.

3. Inspect the "Rmt" column of the "LLC Load Hitm" field of the the "Shared Data Cache Line Table":

```
<table>
<thead>
<tr>
<th>Index</th>
<th>Cacheline</th>
<th>records</th>
<th>Hitm</th>
<th>Total</th>
<th>Lcl</th>
<th>Rmt</th>
<th>Total</th>
<th>L1Hit</th>
<th>L1Miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x602180</td>
<td>149904</td>
<td>77.09%</td>
<td>12103</td>
<td>2269</td>
<td>9834</td>
<td>109504</td>
<td>109036</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0x602100</td>
<td>12128</td>
<td>22.20%</td>
<td>3951</td>
<td>1119</td>
<td>2832</td>
<td>0</td>
<td>0</td>
<td>65</td>
</tr>
<tr>
<td>2</td>
<td>0xffff883ffb6a7e80</td>
<td>260</td>
<td>0.09%</td>
<td>15</td>
<td>3</td>
<td>12</td>
<td>161</td>
<td>97</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>0xffffffff81aec00</td>
<td>157</td>
<td>0.07%</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>0xffffffff81e3f540</td>
<td>179</td>
<td>0.06%</td>
<td>9</td>
<td>1</td>
<td>8</td>
<td>117</td>
<td>97</td>
<td>20</td>
</tr>
</tbody>
</table>
```

This table is sorted in descending order by the amount of remote HITMs 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.
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.

### 18.1. INSTALLING FLAMEGRAPHS

To begin using **flamegraphs**, install the required package:

**Procedure**
- Install the **flamegraphs** package:

  ```bash
  # yum install js-d3-flame-graph
  ```

### 18.2. 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:

  ```bash
  # 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 will sample 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 flamegraph:
# xdg-open flamegraph.html

This command opens the flamegraph in the default browser:

![Flamegraph Image]

## 18.3. 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:

  ```bash
  # 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 will sample 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 flamegraph:

  ```bash
  # xdg-open flamegraph.html
  ```

  This previous command opens the flamegraph in the default browser:
18.4. INTERPRETING FLAMEGRAPHS

Each box in the flamegraph represents a different function in the stack. The y-axis shows the depth of the stack with the topmost box in each stack being the function that was actually on-CPU and everything below it being ancestry. The x-axis displays the population of the sampled call-graph data. 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.

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, the -g option in GCC, to display the function names or symbols in such a situation.

Procedure

- 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:
To return to the default view of the flamegraph, click the **Reset Zoom** button.

**Additional resources**

- Why perf displays some function names as raw functions addresses.
- Enabling debugging with debugging information.
CHAPTER 19. 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.

19.1. 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.

19.2. PROFILING MEMORY ALLOCATION WITH NUMASTAT

Profile the memory allocation of the system by using the numastat tool.

**Prerequisites**

- You have the numactl package installed.

**Procedure**

1. Profile the memory allocation of your system:

   ```
   $ numastat node0 node1
   numa_hit          76557759  92126519
   numa_miss         30772308  30827638
   numa_foreign      30827638  30772308
   ```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>interleave_hit</td>
<td>106507</td>
<td>103832</td>
</tr>
<tr>
<td>local_node</td>
<td>7650227</td>
<td>92086995</td>
</tr>
<tr>
<td>other_node</td>
<td>30827840</td>
<td>30867162</td>
</tr>
</tbody>
</table>

Additional resources

- The `numastat(8)` man page.
CHAPTER 20. CONFIGURING AN OPERATING SYSTEM TO OPTIMIZE CPU UTILIZATION

This section describes how to configure the operating system to optimize CPU utilization across their workloads.

20.1. TOOLS FOR MONITORING AND DIAGNOSING PROCESSOR ISSUES

The following are the tools available in Red Hat Enterprise Linux 8 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
**20.2. DETERMINING 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.

### 20.2.1. Types of system topology

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.

### 20.2.2. 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:

  ```bash
  $ numactl --hardware
  available: 4 nodes (0-3)
  node 0 cpus: 0 4 8 12 16 20 24 28 32 36
  ```
To gather the information about the CPU architecture, such as the number of CPUs, threads, cores, sockets, and NUMA nodes:

```bash
$ 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:

```bash
# yum install hwloc-gui
# lstopo
```
Figure 20.1. The `lstopo` output

```plaintext
To view the detailed textual output:

```
# yum install hwloc
# lstopo-no-graphics
```

Machine (15GB)

<table>
<thead>
<tr>
<th>Package</th>
<th>Core</th>
<th>Core</th>
<th>Core</th>
<th>Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>L3</td>
<td>L1d</td>
<td>L1i</td>
<td>PU #0</td>
<td>PU #4</td>
</tr>
<tr>
<td>L2</td>
<td>L1d</td>
<td>L1i</td>
<td>PU #1</td>
<td>PU #5</td>
</tr>
<tr>
<td>L1</td>
<td>L1</td>
<td></td>
<td>PU #2</td>
<td>PU #6</td>
</tr>
<tr>
<td>L0</td>
<td></td>
<td></td>
<td>PU #3</td>
<td>PU #7</td>
</tr>
</tbody>
</table>

Additionl resources

- For more information, see the `numactl`, `lscpu`, and `lstopo` man pages.

20.3. 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.

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 Section 20.3.1, “Static priority scheduling with SCHED_FIFO” and Section 20.3.2, “Round robin priority scheduling with SCHED_RR”.

20.3.1. 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:
20.3.2 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.

20.3.3 Normal scheduling with SCHED_OTHER

The SCHED_OTHER is the default scheduling policy in Red Hat Enterprise Linux 8. 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.

20.3.4 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

- The `chrt` man page.
- For more information on the policy options, see *Policy Options for the chrt command*.
- For information on setting the policy in a persistent manner, see Section 20.3.6, “Changing the priority of services during the boot process”.

20.3.5. Policy options for the chrt command

To set the scheduling policy of a process, use the appropriate command option:

**Table 20.1. Policy Options for the chrt Command**

<table>
<thead>
<tr>
<th>Short option</th>
<th>Long option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-f</code></td>
<td><code>--fifo</code></td>
<td>Set schedule to <em>SCHED_FIFO</em></td>
</tr>
<tr>
<td><code>-o</code></td>
<td><code>--other</code></td>
<td>Set schedule to <em>SCHED_OTHER</em></td>
</tr>
<tr>
<td><code>-r</code></td>
<td><code>--rr</code></td>
<td>Set schedule to <em>SCHED_RR</em></td>
</tr>
</tbody>
</table>

20.3.6. 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:
   ```bash
   # yum install tuned
   ```

2. Enable and start the tuned service:
   ```bash
   # systemctl enable --now tuned
   ```

**Procedure**

1. View the scheduling priorities of running threads:
   ```bash
   # 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:
   ```bash
   # cat <<<EOF > /etc/systemd/system/mcelog.system.d/priority.conf
   >
   [SERVICE]
   CPUSchedulingPolicy= _fifo_
   CPUSchedulingPriority= _20_
   EOF
   ```

3. Reload the systemd scripts configuration:
   ```bash
   # systemctl daemon-reload
   ```
4. Restart the mcelog service:

```bash
# systemctl restart mcelog
```

**Verification steps**

- Display the `mcelog` priority set by `systemd` issue:

```bash
# tuna -t mcelog -P
```

<table>
<thead>
<tr>
<th>thread</th>
<th>ctxt_switches</th>
<th>pid</th>
<th>SCHED_</th>
<th>rtpri</th>
<th>affinity voluntary nonvoluntary</th>
<th>cmd</th>
</tr>
</thead>
<tbody>
<tr>
<td>826</td>
<td>FIFO</td>
<td>20</td>
<td>0,1,2,3</td>
<td>13</td>
<td>0</td>
<td>mcelog</td>
</tr>
</tbody>
</table>

**Additional resources**

- For more information, see the man pages of `systemd` and `tuna`.
- For more information about priority range, see [Description of the priority range](#).

### 20.3.7. Priority map

Priorities are defined in groups, with some groups dedicated to certain kernel functions.

<table>
<thead>
<tr>
<th>Priority</th>
<th>Threads</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Low priority kernel threads</td>
<td>This priority is usually reserved for the tasks that need to be just above SCHED_OTHER.</td>
</tr>
<tr>
<td>2 - 49</td>
<td>Available for use</td>
<td>The range used for typical application priorities.</td>
</tr>
<tr>
<td>50</td>
<td>Default hard-IRQ value</td>
<td></td>
</tr>
<tr>
<td>51 - 98</td>
<td>High priority threads</td>
<td>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.</td>
</tr>
<tr>
<td>99</td>
<td>Watchdogs and migration</td>
<td>System threads that must run at the highest priority.</td>
</tr>
</tbody>
</table>

### 20.3.8. cpu-partitioning profile

The `cpu-partitioning` profile is used to isolate CPUs from system level interruptions. Once you have isolated these CPUs, you can allocate them for specific applications. This is very useful in low-latency environments or in environments where you wish to extract the maximum performance from your
This profile also lets you designate housekeeping CPUs. A housekeeping CPU is used to run all services, daemons, shell processes, and kernel threads.

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

- **isolated_cores=cpu-list**
  - Lists CPUs to isolate. 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.

- **no_balance_cores=cpu-list**
  - Lists CPUs which are not considered by the kernel during system wide process load-balancing. This option is optional. This is usually the same list as **isolated_cores**.

### 20.3.9. Additional resources

- For more information, see the man pages of `sched`, `sched_setscheduler`, `sched_getscheduler`, `cpuset`, `tuna`, `chrt`, `systemd`, and `tuned-profiles-cpu-partitioning`.

### 20.4. CONFIGURING KERNEL TICK TIME

By default, Red Hat Enterprise Linux 8 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 8 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, append this parameter on the `GRUB_CMDLINE_LINUX` option in the `/etc/default/grub` file:

   ```
   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. To persistently enable the dynamic tickless behavior, regenerate the GRUB2 configuration using the edited default file. On systems with BIOS firmware, execute the following command:

   ```
   # grub2-mkconfig -o /etc/grub2.cfg
   ```

   On systems with UEFI firmware, execute the following command:

   ```
   # grub2-mkconfig -o /etc/grub2-efi.cfg
   ```
3. 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
```

4. Optional: Use the `isolcpus` parameter on the kernel command line to isolate certain cores from user-space tasks.

5. 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
```

- 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

- For more information, see the man pages of `perf`, `tuned-profiles-cpu-partitioning`, and `cpuset`.
- All about nohz_full kernel parameter Red Hat Knowledgebase article.
20.5. SETTING INTERRUPT AFFINITY SYSTEMS

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.

20.5.1. 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:

   ```bash
   $ 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 on how to calculate the smp affinity mask, see Section 20.5.2, “Setting the smp affine mask”.

20.5.2. 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 20.3. Binary Bits for CPUs

<table>
<thead>
<tr>
<th>CPU</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

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

20.5.3. Additional resources

- For more information, see the man pages of irqbalance, journalctl, and taskset.
CHAPTER 21. 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

21.1. TOOLS FOR MONITORING AND DIAGNOSING I/O AND FILE SYSTEM ISSUES

The following tools are available in Red Hat Enterprise Linux 8 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](#).

- **bpftace** 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)`, `bpftrace`, and `iowatcher(1)` man pages
- Analyzing system performance with BPF Compiler Collection

21.2. 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.
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

### 21.3. 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

21.4. 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 8 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 8 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 8 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

21.5. 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 on 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.

### 21.6. GENERIC BLOCK DEVICE TUNING PARAMETERS

The generic tuning parameters listed in this section 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 `iostats`. 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 22. CONFIGURING RHEL TO OPTIMIZE ACCESS TO NETWORK RESOURCES

This section describes how to configure RHEL to present optimized access to network resources across their workloads. Network performance problems are sometimes the result of hardware malfunction or faulty infrastructure. Resolving these issues is beyond the scope of this document. The Tuned service provides a number of different profiles to improve performance in a number of specific use cases:

- latency-performance
- network-latency
- network-throughput

22.1. TOOLS FOR MONITORING AND DIAGNOSING PERFORMANCE ISSUES

The following are the available tools in Red Hat Enterprise Linux 8, which are used for monitoring system performance and diagnosing performance problems related to the networking subsystem:

- ss is a command-line utility. It prints statistical information about sockets, enables administrators to assess device performance over time. By default, ss displays open non-listening TCP sockets that have established connections. Using command-line options, administrators can filter out statistics about specific sockets. Red Hat recommends ss over the deprecated netstat in Red Hat Enterprise Linux.

- ip utility lets administrators manage and monitor routes, devices, routing policies, and tunnels. The ip monitor command can continuously monitor the state of devices, addresses, and routes. Use the -j option to display the output in JSON format, which can be further provided to other utilities to automate information processing.

- dropwatch is an interactive tool, provided by the dropwatch package. It monitors and records packets that are dropped by the kernel.

- ethtool is a utility that enables administrators to view and edit network interface card settings. Use this tool to observe the statistics, such as the number of packets dropped by that device, of certain devices. Using the ethtool -S device name command, view the status of a specified device’s counters of the device you want to monitor.

- The /proc/net/snmp file displays data that the snmp agent uses for IP, ICMP, TCP and UDP monitoring and management. Examining this file on a regular basis helps administrators to identify unusual values and thereby identify potential performance problems. For example, an increase in UDP input errors (InErrors) in the /proc/net/snmp file can indicate a bottleneck in a socket receive queue.

- nstat is a command-line tool, which monitors kernel SNMP and network interface statistics. This tool reads data from the /proc/net/snmp file and prints the information in a human readable format.

- By default, the SystemTap scripts, provided by the systemtap-client package are installed in the /usr/share/systemtap/examples/network directory:
  - nettop.stp: Every 5 seconds, the script displays a list of processes (process identifier and command) with the number of packets sent and received and the amount of data sent and received by the process during that interval.
- **socket-trace.stp**: Instruments each of the functions in the Linux kernel's `net/socket.c` file, and displays trace data.

- **dropwatch.stp**: Every 5 seconds, the script displays the number of socket buffers freed at locations in the kernel. Use the `--all-modules` option to see symbolic names.

- **latencytap.stp**: This script records the effect that different types of latency have on one or more processes. It prints a list of latency types every 30 seconds, sorted in descending order by the total time the process or processes spent waiting. This can be useful for identifying the cause of both storage and network latency.

Red Hat recommends using the `--all-modules` option with this script to better enable the mapping of latency events. By default, this script is installed in the `/usr/share/systemtap/examples/profiling` directory.

- **BPF Compiler Collection (BCC)** is a library, which facilitates the creation of the extended Berkeley Packet Filter (eBPF) programs. The main utility of the eBPF programs is analyzing OS performance and network performance without experiencing overhead or security issues.

**Additional resources**

- For more information, see the ss, ethtool, nettop, ip, dropwatch, and SystemTap man pages.

- The `/usr/share/systemtap/examples/network` directory.

- For more information about BCC, see the `/usr/share/doc/bcc/README.md` file, which is provided by the bcc package.

- How to write a NetworkManager dispatcher script to apply ethtool commands? Red Hat Knowledgebase solution.

- Configuring ethtool offload features using NetworkManager section.

**22.2. BOTTLENECKS IN A PACKET RECEPTION**

While the network stack is largely self-optimizing, there are a number of points during network packet processing that can become bottlenecks and reduce the performance. The following are the issues that can cause bottleneck:

**The buffer or ring buffer of the network card**

The hardware buffer can be a bottleneck if the kernel drops a large number of packets. Use the ethtool utility for monitoring a system for dropped packets.

**The hardware or software interrupt queues**

Interrupts can increase latency and processor contention. For information on how the processor handles interrupts, see Setting Interrupt Affinity systems.

**The socket receive queue of the application**

A large number of packets that are not copied or by an increase in the UDP input errors (InErrors) in the `/proc/net/snmp` file, indicates a bottleneck in an application’s receive queue.

If a the hardware buffer drops a large number of packets, the following are the few potential solutions:

**Slow the input traffic**

Filter the incoming traffic, reduce the number of joined multicast groups, or reduce the amount of broadcast traffic to decrease the rate at which the queue fills.
Resize the hardware buffer queue

Resize the hardware buffer queue: Reduce the number of packets being dropped by increasing the size of the queue so that it does not overflow as easily. You can modify the rx/tx parameters of the network device with the `ethtool` command:

```
ethtool --set-ring device-name value
```

Change the drain rate of the queue

- Decrease the rate at which the queue fills by filtering or dropping packets before they reach the queue, or by lowering the weight of the device. Filter incoming traffic or lower the network interface card’s device weight to slow incoming traffic.

The device weight refers to the number of packets a device can receive at one time in a single scheduled processor access. You can increase the rate at which a queue is drained by increasing its device weight that is controlled by the `dev_weight` kernel setting. To temporarily alter this parameter, change the contents of the `/proc/sys/net/core/dev_weight` file, or to permanently alter, use the `sysctl` command, which is provided by the `procps-ng` package.

- Increase the length of the application’s socket queue: This is typically the easiest way to improve the drain rate of a socket queue, but it is unlikely to be a long-term solution. If a socket queue receives a limited amount of traffic in bursts, increasing the depth of the socket queue to match the size of the bursts of traffic may prevent packets from being dropped. To increase the depth of a queue, increase the size of the socket receive buffer by making either of the following changes:

  - Increase the value of the `/proc/sys/net/core/rmem_default` parameter: This parameter controls the default size of the receive buffer used by sockets. This value must be smaller than or equal to the value of the `/proc/sys/net/core/rmem_max` parameter.

  - Use the `setsockopt` to configure a larger `SO_RCVBUF` value: This parameter controls the maximum size in bytes of a socket’s receive buffer. Use the `getsockopt` system call to determine the current value of the buffer.

Altering the drain rate of a queue is usually the simplest way to mitigate poor network performance. However, increasing the number of packets that a device can receive at one time uses additional processor time, during which no other processes can be scheduled, so this can cause other performance problems.

Additional resources

- For more information, see the `ss`, `socket`, and `ethtool` man pages.
- The `/proc/net/snmp` file.

22.3. CONFIGURING BUSY POLLING

If analysis reveals high latency, your system may benefit from the poll-based rather than interrupt-based packet receipt.

Busy polling helps to reduce latency in the network receive path by allowing socket layer code to poll the receive queue of a network device, and disables network interrupts. This removes delays caused by the interrupt and the resultant context switch. However, it also increases CPU utilization. Busy polling also prevents the CPU from sleeping, which can incur additional power consumption. Busy polling behavior is supported by all the device drivers.
22.3.1. Enabling busy polling

By default, the busy polling is disabled. This procedure describes how to enable busy polling.

Procedure

1. Ensure if the CONFIG_NET_RX_BUSY_POLL compilation option is enabled:

   ```bash
   # cat /boot/config-$uname -r | grep CONFIG_NET_RX_BUSY_POLL
   CONFIG_NET_RX_BUSY_POLL=y
   ```

2. Enable busy polling

   a. To enable busy polling on specific sockets, set the `sysctl.net.core.busy_poll` kernel value to a value other than `0`:

   ```bash
   # echo "net.core.busy_poll=50" > /etc/sysctl.d/95-enable-busy-polling-for-sockets.conf
   # sysctl -p /etc/sysctl.d/95-enable-busy-polling-for-sockets.conf
   ```

   This parameter controls the number of microseconds to wait for packets on the socket poll and select `syscalls`. Red Hat recommends a value of `50`.

   b. Add the `SO_BUSY_POLL` socket option to the socket.

   c. To enable busy polling globally, set the `sysctl.net.core.busy_read` to a value other than `0`:

   ```bash
   # echo "net.core.busy_read=50" > /etc/sysctl.d/95-enable-busy-polling-globally.conf
   # sysctl -p /etc/sysctl.d/95-enable-busy-polling-globally.conf
   ```

   The `net.core.busy_read` parameter controls the number of microseconds to wait for packets on the device queue for socket reads. It also sets the default value of the `SO_BUSY_POLL` option. Red Hat recommends a value of `50` for a small number of sockets, and a value of `100` for large numbers of sockets. For extremely large numbers of sockets, for example more than several hundred, use the `epoll` system call instead.

Verification steps

- Verify if the busy poll is enabled

  ```bash
  # ethtool -k device | grep "busy-poll"
  busy-poll: on [fixed]
  ```

  ```bash
  # cat /proc/sys/net/core/busy_read
  50
  ```

Additional resources

- For more information, see the `ethtool`, `socket`, `sysctl`, and `sysctl.conf` man pages.

- Configuring `ethtool` offload features using NetworkManager section.

22.4. RECEIVE-SIDE SCALING

Receive-Side Scaling (RSS), also known as multi-queue receive, distributes network receive processing
across several hardware-based receive queues, allowing inbound network traffic to be processed by multiple CPUs. RSS can be used to relieve bottlenecks in receive interrupt processing caused by overloading a single CPU, and to reduce network latency. By default, RSS is enabled.

The number of queues or the CPUs that should process network activity for RSS are configured in the appropriate network device driver:

- For the **bnx2x** driver, it is configured in the `num_queues` parameter.
- For the **sfc** driver, it is configured in the `rss_cpus` parameter.

Regardless, it is typically configured in the `/sys/class/net/device/queues/rx-queue` directory, where `device` is the name of the network device (such as `enp1s0`) and `rx-queue` is the name of the appropriate receive queue.

The **irqbalance** daemon can be used in conjunction with RSS to reduce the likelihood of cross-node memory transfers and cache line bouncing. This lowers the latency of processing network packets.

### 22.4.1. Viewing the interrupt request queues

When configuring RSS, Red Hat recommends limiting the number of queues to one per physical CPU core. Hyper-threads are often represented as separate cores in analysis tools, but configuring queues for all cores including logical cores such as hyper-threads has not proven beneficial to network performance.

When enabled, RSS distributes network processing equally between available CPUs based on the amount of processing each CPU has queued. However, use the `--show-rxfh-indir` and `--set-rxfh-indir` parameters of the **ethtool** utility, to modify how RHEL distributes network activity, and weigh certain types of network activity as more important than others.

This procedure describes how to view the interrupt request queues.

**Procedure**

- To determine whether your network interface card supports RSS, check whether multiple interrupt request queues are associated with the interface in `/proc/interrupts`:

```
# egrep 'CPU|p1p1' /proc/interrupts
CPU0  CPU1  CPU2  CPU3  CPU4  CPU5
89:  40187  0  0  0  0  0  IR-PCI-MSI-edge  p1p1-0
90:   0   790  0  0  0  0  IR-PCI-MSI-edge  p1p1-1
91:   0   959  0  0  0  0  IR-PCI-MSI-edge  p1p1-2
92:   0   0   0  3310  0  0  IR-PCI-MSI-edge  p1p1-3
93:   0   0   0  622  0  0  IR-PCI-MSI-edge  p1p1-4
94:   0   0   0  0  2475  0  IR-PCI-MSI-edge  p1p1-5
```

The output shows that the NIC driver created 6 receive queues for the **p1p1** interface (**p1p1-0** through **p1p1-5**). It also shows how many interrupts were processed by each queue, and which CPU serviced the interrupt. In this case, there are 6 queues because by default, this particular NIC driver creates one queue per CPU, and this system has 6 CPUs. This is a fairly common pattern among NIC drivers.

- To list the interrupt request queue for a PCI device with the address **0000:01:00.0**:

```
# ls -l /sys/devices/*/0000:01:00.0/msi_irqs
101
```
22.5. RECEIVE PACKET STEERING

Receive Packet Steering (RPS) is similar to RSS in that it is used to direct packets to specific CPUs for processing. However, RPS is implemented at the software level, and helps to prevent the hardware queue of a single network interface card from becoming a bottleneck in network traffic. RPS has several advantages over hardware-based RSS:

- RPS can be used with any network interface card.
- It is easy to add software filters to RPS to deal with new protocols.
- RPS does not increase the hardware interrupt rate of the network device. However, it does introduce inter-processor interrupts.

RPS is configured per network device and receive queue, in the `/sys/class/net/device/queues/rx-queue/rps_cpus` file, where `device` is the name of the network device, such as `enp1s0` and `rx-queue` is the name of the appropriate receive queue, such as `rx-0`.

The default value of the `rps_cpus` file is 0. This disables RPS, and the CPU handles the network interrupt and also processes the packet. To enable RPS, configure the appropriate `rps_cpus` file with the CPUs that should process packets from the specified network device and receive queue.

The `rps_cpus` files use comma-delimited CPU bitmaps. Therefore, to allow a CPU to handle interrupts for the receive queue on an interface, set the value of their positions in the bitmap to 1. For example, to handle interrupts with CPUs 0, 1, 2, and 3, set the value of the `rps_cpus` to f, which is the hexadecimal value for 15. In binary representation, 15 is `00001111` (1+2+4+8).

For network devices with single transmit queues, best performance can be achieved by configuring RPS to use CPUs in the same memory domain. On non-NUMA systems, this means that all available CPUs can be used. If the network interrupt rate is extremely high, excluding the CPU that handles network interrupts may also improve performance.

For network devices with multiple queues, there is typically no benefit to configure both RPS and RSS, as RSS is configured to map a CPU to each receive queue by default. However, RPS can still be beneficial if there are fewer hardware queues than CPUs, and RPS is configured to use CPUs in the same memory domain.

22.6. RECEIVE FLOW STEERING

Receive Flow Steering (RFS) extends RPS behavior to increase the CPU cache hit rate and thereby reduce network latency. Where RPS forwards packets based solely on queue length, RFS uses the RPS back end to calculate the most appropriate CPU, then forwards packets based on the location of the application consuming the packet. This increases CPU cache efficiency.

Data received from a single sender is not sent to more than one CPU. If the amount of data received from a single sender is greater than a single CPU can handle, configure a larger frame size to reduce the
number of interrupts and therefore the amount of processing work for the CPU. Alternatively, consider
NIC offload options or faster CPUs.

Consider using `numactl` or `taskset` in conjunction with RFS to pin applications to specific cores, sockets,
or NUMA nodes. This can help prevent packets from being processed out of order.

22.6.1. Enabling Receive Flow Steering

By default, Receive Flow Steering (RFS) is disabled. This procedure describes how to enable RFS.

**Procedure**

1. Set the value of the `net.core.rps_sock_flow_entries` kernel value to the maximum expected
   number of concurrently active connections:
   
   ```
   # echo "net.core.rps_sock_flow_entries=32768" > /etc/sysctl.d/95-enable-rps.conf
   
   Red Hat recommends a value of 32768 for moderate server loads. All values entered are
   rounded up to the nearest power of 2 in practice.
   ```

2. Persistently set the value of the `net.core.rps_sock_flow_entries`:
   
   ```
   # sysctl -p /etc/sysctl.d/95-enable-rps.conf
   ```

3. To temporarily set the value of the `sys/class/net/device/queues/rx-queue/rps_flow_cnt` file
to the value of the `(rps_sock_flow_entries/N)`, where \( N \) is the number of receive queues on a
device:
   
   ```
   # echo 2048 > /sys/class/net/device/queues/rx-queue/rps_flow_cnt
   
   Replace `device` with the name of the network device you wish to configure (for example,
enp1s0), and `rx-queue` with the receive queue you wish to configure (for example, `rx-0`).
   Replace \( N \) with the number of configured receive queues. For example, if the `rps_flow_entries`
is set to 32768 and there are 16 configured receive queues, the `rps_flow_cnt = 32786/16= 2048` (that is, \( rps_flow_cnt = rps_flow_enties/N \)).

   For single-queue devices, the value of `rps_flow_cnt` is the same as the value of
   `rps_sock_flow_entries`.

4. Persistently enable RFS on all network devices, create the `/etc/udev/rules.d/99-persistent-
   net.rules` file, and add the following content:
   
   ```
   SUBSYSTEM=="net", ACTION=="add", RUN[program]=="/bin/bash -c 'for x in
   /sys/$DEVPATH/queues/rx-*; do echo 2048 > $x/rps_flow_cnt; done'"
   ```

5. Optional: To enable RPS on a specific network device:
   
   ```
   SUBSYSTEM=="net", ACTION=="move", NAME="device name" RUN[program]=="/bin/bash
   -c 'for x in /sys/$DEVPATH/queues/rx-*; do echo 2048 > $x/rps_flow_cnt; done'"
   
   Replace `device name` with the actual network device name.

**Verification steps**
• Verify if RFS is enabled:

```
# cat /proc/sys/net/core/rps_sock_flow_entries
32768
# cat /sys/class/net/device/queues/rx-queue/rps_flow_cnt
2048
```

**Additional resources**

• For more information, see the `sysctl` man page.

### 22.7. ACCELERATED RFS

Accelerated RFS boosts the speed of RFS by adding hardware assistance. Like RFS, packets are forwarded based on the location of the application consuming the packet. Unlike traditional RFS, however, packets are sent directly to a CPU that is local to the thread consuming the data:

• either the CPU that is executing the application

• or a CPU local to that CPU in the cache hierarchy

Accelerated RFS is only available if the following conditions are met:

• NIC must support the accelerated RFS. Accelerated RFS is supported by cards that export the `ndo_rx_flow_steer()` net_device function. Check the NIC’s data sheet to ensure if this feature is supported.

• ntuple filtering must be enabled. For information on how to enable these filters, see Section 22.7.1, “Enabling the ntuple filters”.

Once these conditions are met, CPU to queue mapping is deduced automatically based on traditional RFS configuration. That is, CPU to queue mapping is deduced based on the IRQ affinities configured by the driver for each receive queue. For more information on enabling the traditional RFS, see Section 22.6.1, “Enabling Receive Flow Steering”.

#### 22.7.1. Enabling the ntuple filters

Use the `ethtool -k` command to check if the ntuple filters are enabled.

**Procedure**

1. Display the current status of the ntuple filter:

```
# ethtool -k enp1s0 | grep ntuple-filters
ntuple-filters: off
```

2. Enable the ntuple filters:

```
# ethtool -k enp1s0 ntuple on
```
NOTE

If the output is `ntuple-filters: off [fixed]`, then the `ntuple` filtering is disabled and you cannot configure it:

```
# ethtool -k enp1s0 | grep ntuple-filters
ntuple-filters: off [fixed]
```

Verification steps

- Ensure if `ntuple` filters are enabled:

```
# ethtool -k enp1s0 | grep ntuple-filters
ntuple-filters: on
```

Additional resources

- For more information, see the `ethtool` man page.
CHAPTER 23. CONFIGURING AN OPERATING SYSTEM TO OPTIMIZE MEMORY ACCESS

This section describes how to configure the operating system to optimize memory access across workloads, and the tools you can use to do so.

23.1. TOOLS FOR MONITORING AND DIAGNOSING SYSTEM MEMORY ISSUES

The following tools are available in Red Hat Enterprise Linux 8 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** is a framework that provides instrumentation to user-space binaries. Install this tool, using the `yum 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

- For more information, see the *vmstat* and *valgrind* man page.

- For more information on the *valgrind* framework, see the `/usr/share/doc/valgrind-version/valgrind_manual.pdf` file.
23.2. 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.

---

23.3. OPTIMIZING A SYSTEM’S MEMORY UTILIZATION

This section provides information about memory-related kernel parameters and how you can use them to improve a system’s memory utilization. The following are the memory-related kernel parameters available in Red Hat Enterprise Linux 8:

- Virtual memory parameter. For more information, see Section 23.3.1, “Virtual memory parameters”.
- File system parameter. For more information, see Section 23.3.2, “File system parameters”.
- Kernel parameter. For more information, see Section 23.3.3, “Kernel parameters”.

### 23.3.1. Virtual memory parameters

The virtual memory parameters are listed in the `/proc/sys/vm` directory unless otherwise indicated.
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 heuristic memory overcommit handling by estimating the amount of memory available and failing requests that are too large. However, since memory is allocated using a heuristic rather than a precise algorithm, overloading memory is possible with this setting.

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 100, 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 filesystem 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 30.
 Additional resources

- For more information, see the `sysctl` man page.
- For more information on how to set these parameters temporarily and persistently, see Section 23.3.4, “Setting memory-related kernel parameters”.

### 23.3.2. 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 8 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

- For more information, see the `sysctl` man page.

### 23.3.3. Kernel parameters

The default values for the kernel parameters are located in the `/proc/sys/kernel/` directory. These values are calculated by the kernel at boot time depending on the available system resources.

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. 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

- For more information, see the `sysvipc` and `sysctl` man page.

### 23.3.4. 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.
  
  For example, to temporarily set the `vm.overcommit_memory` parameter to 1:

  ```sh
  # echo 1 > /proc/sys/vm/overcommit_memory
  # sysctl -w vm.overcommit_memory=1
  ```

- To persistently set the memory-related kernel parameter, use the `sysctl` tool.
  
  For example, to persistently set the `vm.overcommit_memory` parameter to 1:

  a. Add the following content in the `/etc/sysctl.conf` file:

  ```
  vm.overcommit_memory=1
  ```

  b. Reload the `sysctl` settings from the `/etc/sysctl.conf` file:

  ```
  # sysctl -p
  ```

Additional resources

- For more information, see the `sysctl` man page.
CHAPTER 24. CONFIGURING HUGE PAGES

Physical memory is managed in fixed-size chunks called pages. On the x86_64 architecture, supported by Red Hat Enterprise Linux 8, 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 Red Hat Enterprise Linux 8 and how you can configure them.

24.1. AVAILABLE HUGE PAGE FEATURES

With Red Hat Enterprise Linux 8, 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 Red Hat Enterprise Linux 8:

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 on parameters that influence HugeTLB page behavior at boot time, see Section 24.2, “Parameters for reserving HugeTLB pages at boot time” and how to use these parameters to configure HugeTLB pages at boot time, see Section 24.4, “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 on parameters that influence HugeTLB page behavior at run time, see Section 24.3, “Parameters for reserving HugeTLB pages at run time” and how to use these parameters to configure HugeTLB pages at run time, see Section 24.5, “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 information on enabling and disabling THP, see Section 24.6, “Enabling transparent hugepages” and Section 24.7, “Disabling transparent hugepages”.

24.2. PARAMETERS FOR RESERVING HUGETLB PAGES AT BOOT TIME

Use the following parameters to influence HugeTLB page behavior at boot time:

Table 24.1. Parameters used to configure HugeTLB pages at boot time

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>hugepages</td>
<td>Defines the number of persistent huge pages configured in the kernel at boot time. In a NUMA system, huge pages, that have this parameter defined, are divided equally between nodes. 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.</td>
<td>The default value is 0. To update this value at boot, change the value of this parameter in the <code>/proc/sys/vm.nr_hugepages</code> file.</td>
</tr>
<tr>
<td>hugepagesizez</td>
<td>Defines the size of persistent huge pages configured in the kernel at boot time.</td>
<td>Valid values are 2 MB and 1 GB. The default value is 2 MB.</td>
</tr>
<tr>
<td>default_hugepagesizez</td>
<td>Defines the default size of persistent huge pages configured in the kernel at boot time.</td>
<td>Valid values are 2 MB and 1 GB. The default value is 2 MB.</td>
</tr>
</tbody>
</table>

24.3. PARAMETERS FOR RESERVING HUGETLB PAGES AT RUN TIME

Use the following parameters to influence HugeTLB page behavior at run time:

Table 24.2. Parameters used to configure HugeTLB pages at run time

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>File name</th>
</tr>
</thead>
<tbody>
<tr>
<td>nr_hugepages</td>
<td>Defines the number of huge pages of a specified size assigned to a specified NUMA node.</td>
<td><code>/sys/devices/system/node/node_id/hugepages/hugepages-size/nr_hugepages</code></td>
</tr>
</tbody>
</table>
### nr_overcommit_hugepages

Defines the maximum number of additional huge pages that can be created and used by the system through overcommitting memory.

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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>File name</th>
</tr>
</thead>
<tbody>
<tr>
<td>nr_overcommit_hugepages</td>
<td>Defines the maximum number of additional huge pages that can be created and used by the system through overcommitting memory. 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.</td>
<td>/proc/sys/vm/nr_overcommit_hugepages</td>
</tr>
</tbody>
</table>

### 24.4. 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. Create a HugeTLB pool for 1 GB pages by appending the following line to the kernel command-line options in the `/etc/default/grub` file as root:

   ```
   default_hugepagesz=1G hugepagesz=1G
   ```

2. Regenerate the GRUB2 configuration using the edited default file:
   
   a. If your system uses BIOS firmware, execute the following command:

   ```
   # grub2-mkconfig -o /boot/grub2/grub.cfg
   ```

   b. If your system uses UEFI framework, execute the following command:

   ```
   # grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
   ```

3. 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]
   ```
4. 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.

```bash
#!/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 1GB pages on `node0` and one 1GB page on `node1`, replace the `number_of_pages` with 2 for `node0` and 1 for `node1`:

```bash
reserve_pages 2 node0
reserve_pages 1 node1
```

5. Create an executable script:

```bash
# chmod +x /usr/lib/systemd/hugetlb-reserve-pages.sh
```

6. Enable early boot reservation:

```bash
# systemctl enable hugetlb-gigantic-pages
```
NOTE

- You can try reserving more 1GB pages at runtime by writing to `nr_hugepages` at any time. However, such reservations can fail due to memory fragmentation. The most reliable way to reserve 1GB pages is by using this `hugetlb-reserve-pages.sh` script, which runs early during boot.

- 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

- For more information, see the relevant kernel documentation, which is installed in the `/usr/share/doc/kernel-doc-kernel_version/Documentation/vm/hugetlbpage.txt` file.

- For more information, see the `systemd.service` man page.

### 24.5. CONFIGURING HUGETLB AT RUN TIME

This procedure describes how to add twenty 2048 kB huge pages to `node2`.

#### 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
   ```

   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.

#### 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
  ```
24.6. ENABLING TRANSPARENT HUGE PAGES

THP is enabled by default in Red Hat Enterprise Linux 8. However, you can enable or disable THP. This procedure describes how to enable THP.

**Procedure**

1. Check the current status of THP:

   ```bash
   # cat /sys/kernel/mm/transparent_hugepage/enabled
   ```

2. Enable THP:

   ```bash
   # 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`:

   ```bash
   # 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:

   ```bash
   # echo madvise > /sys/kernel/mm/transparent_hugepage/defrag
   ```

**Additional resources**

- For more information, see the `madvise` man page.

24.7. DISABLING TRANSPARENT HUGE PAGES

THP is enabled by default in Red Hat Enterprise Linux 8. However, you can enable or disable THP. This procedure describes how to disable THP.
Procedure

1. Check the current status of THP:
   
   ```bash
   # cat /sys/kernel/mm/transparent_hugepage/enabled
   ```

2. Disable THP:
   
   ```bash
   # echo never > /sys/kernel/mm/transparent_hugepage/enabled
   ```

### 24.8. 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 25. 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.

25.1. 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.

25.2. DEPLOYING SYSTEMTAP

To use SystemTap, you must install the associated SystemTap and kernel packages.

25.2.1. 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:

```bash
# subscription-manager repos --enable rhel-8-for-$(uname -i)-baseos-debug-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-baseos-source-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-appstream-debug-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-appstream-source-rpms
```

The `$(uname -i)` part is automatically replaced with a matching value for architecture of your system:
<table>
<thead>
<tr>
<th>Architecture name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>64-bit Intel and AMD</td>
<td>x86_64</td>
</tr>
<tr>
<td>64-bit ARM</td>
<td>aarch64</td>
</tr>
<tr>
<td>IBM POWER</td>
<td>ppc64le</td>
</tr>
<tr>
<td>64-bit IBM Z</td>
<td>s390x</td>
</tr>
</tbody>
</table>

### 25.2.2. 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:
   
   ```
   # yum 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:
      
      ```
      # yum install kernel-debuginfo-$(uname -r) kernel-debuginfo-common-$(uname -i)-$(uname -r) kernel-devel-$(uname -r)
      ```

      $\texttt{(uname -i)}$ is automatically replaced with the hardware platform of your system and $\texttt{(uname -r)}$ 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_e5886fa50499994e6a87aadcd43cd392_399.c" in
490usr/430sys/938real ms.
Pass 4: compiled C into "stap_e5886fa50499994e6a87aadcd43cd392_399.ko" in
3310usr/430sys/3714real ms.
Pass 5: starting run.
read performed
Pass 5: run completed in 10usr/40sys/73real ms.

The last three lines of output (beginning with Pass 5) indicate that:
1. SystemTap successfully created the instrumentation to probe the kernel and ran the instrumentation.
2. SystemTap detected the specified event (in this case, A VFS read).
3. SystemTap executed a valid handler (printed text and then closed it with no errors).

25.3. CROSS-INSTRUMENTATION OF SYSTEMTAP

25.3.1. 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 on 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.

**CAUTION**

Kernel packaging bugs may prevent this. In such cases, the kernel-debuginfo and kernel-devel packages for the host system and target system must match. If this 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*.

25.3.2. 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](https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/8/html/Sysdig_Guide/).
- 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 8), they can be running different minor versions (such as 8.1 and 8.2).

**CAUTION**

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 this occurs, report the bug at [https://bugzilla.redhat.com/](https://bugzilla.redhat.com/).

**Procedure**

1. Determine the kernel running on each *target system*:

   ```
   $ uname -r
   ```

   Repeat this step for each *target system*.

2. Install the package required to run pre-compiled modules on each *target system*:

   ```
   # yum install systemtap-runtime
   ```

3. On the *host system*, install the *target kernel* and related packages for each *target system* by the method described in [installing SystemTap](https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/8/html/Sysdig_Guide/).
4. Build an instrumentation module on the host system, copy to and run on the target system either:

a. Using remote implementation:

   ```bash
   # 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:

   ```bash
   # 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 `-p 4` 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:

   ```bash
   # staprun module_name.ko
   ```

   **NOTE**

   Running a pre-compiled module only requires the `systemtap-runtime` package.

25.4. RUNNING SYSTEMTAP SCRIPTS

25.4.1. 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.
25.4.2. 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 26. 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.

26.1. AN INTRODUCTION TO BCC

BPF Compiler Collection (BCC) is a library, which facilitates the creation of the extended Berkeley Packet Filter (eBPF) programs. The main utility of eBPF programs is analyzing OS performance and network performance without experiencing overhead or security issues.

BCC removes the need for users to know deep technical details of eBPF, and provides many out-of-the-box starting points, such as the bcc-tools package with pre-created eBPF programs.

NOTE

The eBPF programs are triggered on events, such as disk I/O, TCP connections, and process creations. It is unlikely that the programs should cause the kernel to crash, loop or become unresponsive because they run in a safe virtual machine in the kernel.

26.2. INSTALLING THE BCC-TOOLS PACKAGE

This section describes how to install the bcc-tools package, which also installs the BPF Compiler Collection (BCC) library as a dependency.

Prerequisites

- An active Red Hat Enterprise Linux subscription
- An enabled repository containing the bcc-tools package
- Updated kernel
- Root permissions

Procedure

1. Install bcc-tools:

   ```bash
   # yum install bcc-tools
   ```

   The BCC tools are installed in the /usr/share/bcc/tools/ directory.

2. Optionally, inspect the tools:

   ```bash
   # 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
   ```
The **doc** directory in the listing above contains documentation for each tool.

### 26.3. USING SELECTED BCC-TOOLS FOR PERFORMANCE ANALYSES

This section describes how to 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 library
- Root permissions

**Using execsnoop to examine the system processes**

1. Execute the **execsnoop** program in one terminal:

   ```
   # /usr/share/bcc/tools/execsnoop
   ```

2. In another terminal execute 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/
   sed  8385  8383  0 /usr/bin/sed s/^ *[0-9]\+ */
   ... 
   ```

   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. Execute 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 execute:

   `$ 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. Execute 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 execute 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        83  994.0     0.28
8112  gnome-terminal- R 252 0   vda       620 2099.0    0.43
7807  gnome-software R 252 0   vda       317 1325.0    0.33
9578  awk         R 252 0   vda         7  2228.0     0.66
7578  llvmpipe-0 R 252 0   vda       156 2204.0     0.07
9581  pgrep       R 252 0   vda        58  1484.0     0.30
7531  InputThread R 252 0   vda       1200.0    0.48
7504  gdbus       R 252 0   vda        37  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. Execute the `xfsslower` program in one terminal:
   ```bash
   # /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 execute, for example, the following:
   ```bash
   $ 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.