Red Hat Enterprise Linux 9.0

Monitoring and managing system status and performance

Optimizing system throughput, latency, and power consumption
Red Hat Enterprise Linux 9.0 Monitoring and managing system status and performance

Optimizing system throughput, latency, and power consumption
Abstract

This documentation collection provides instructions on how to monitor and optimize the throughput, latency, and power consumption of Red Hat Enterprise Linux 9 in different scenarios.
# Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE ......................................................... 7

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION .................................. 8

CHAPTER 1. GETTING STARTED WITH TUNED .................................................. 9

  1.1. THE PURPOSE OF TUNED ................................................................. 9
  1.2. TUNED PROFILES ............................................................................. 9
  1.2.1. Syntax of profile configuration .................................................... 9
  1.3. THE DEFAULT TUNED PROFILE ....................................................... 10
  1.4. MERGED TUNED PROFILES ............................................................ 10
  1.5. THE LOCATION OF TUNED PROFILES ........................................... 11
  1.6. TUNED PROFILES DISTRIBUTED WITH RHEL .............................. 11
  1.7. TUNED CPU-PARTITIONING PROFILE ........................................... 13
  1.8. USING THE TUNED CPU-PARTITIONING PROFILE FOR LOW-LATENCY TUNING ................................................................. 14
  1.9. CUSTOMIZING THE CPU-PARTITIONING TUNED PROFILE .............. 15
  1.10. REAL-TIME TUNED PROFILES DISTRIBUTED WITH RHEL .......... 16
  1.11. STATIC AND DYNAMIC TUNING IN TUNED .................................... 16
  1.12. TUNED NO-DAEMON MODE ......................................................... 17
  1.13. INSTALLING AND ENABLING TUNED ........................................... 17
  1.14. LISTING AVAILABLE TUNED PROFILES ....................................... 18
  1.15. SETTING A TUNED PROFILE .......................................................... 19
  1.16. DISABLING TUNED ....................................................................... 20

CHAPTER 2. CUSTOMIZING TUNED PROFILES ................................................. 21

  2.1. TUNED PROFILES ............................................................................. 21
  2.2. Syntax of profile configuration ....................................................... 21
  2.3. THE DEFAULT TUNED PROFILE ....................................................... 21
  2.4. MERGED TUNED PROFILES ............................................................ 22
  2.5. THE LOCATION OF TUNED PROFILES ........................................... 22
  2.6. INHERITANCE BETWEEN TUNED PROFILES ................................. 23
  2.7. STATIC AND DYNAMIC TUNING IN TUNED .................................... 23
  2.8. TUNED PLUG-INS ........................................................................... 24
  2.9. Syntax for plug-ins in TuneD profiles .............................................. 25
  2.10. Short plug-in syntax ................................................................. 25
  2.11. Conflicting plug-in definitions in a profile .................................... 26
  2.12. AVAILABLE TUNED PLUG-INS ..................................................... 26
  2.13. Monitoring plug-ins ................................................................. 26
  2.14. Tuning plug-ins ................................................................. 26
  2.15. VARIABLES IN TUNED PROFILES ............................................. 30
  2.16. BUILT-IN FUNCTIONS IN TUNED PROFILES ............................ 31
  2.17. BUILT-IN FUNCTIONS AVAILABLE IN TUNED PROFILES ....... 31
  2.18. CREATING NEW TUNED PROFILES ........................................... 32
  2.19. MODIFYING EXISTING TUNED PROFILES ................................. 33
  2.20. SETTING THE DISK SCHEDULER USING TUNED ....................... 34

CHAPTER 3. MONITORING PERFORMANCE USING RHEL SYSTEM ROLES ........ 37

  3.1. INTRODUCTION TO RHEL SYSTEM ROLES .................................... 37
  3.2. RHEL SYSTEM ROLES TERMINOLOGY ........................................ 37
  3.3. INSTALLING RHEL SYSTEM ROLES IN YOUR SYSTEM .................. 38
  3.4. APPLYING A ROLE ...................................................................... 38
  3.5. INTRODUCTION TO THE METRICS SYSTEM ROLE ....................... 41
  3.6. USING THE METRICS SYSTEM ROLE TO MONITOR YOUR LOCAL SYSTEM WITH VISUALIZATION ................................................................. 42
### Chapter 3.7. Using the Metrics System Role to Setup a Fleet of Individual Systems to Monitor Themselves

- Using the Metrics System Role to Monitor a Fleet of Machines Centrally Via Your Local Machine
- Setting Up Authentication While Monitoring a System Using the Metrics System Role
- Using the Metrics System Role to Configure and Enable Metrics Collection for SQL Server

### Chapter 4. Setting Up PCP

- Overview of PCP
- Installing and Enabling PCP
- Deploying a Minimal PCP Setup
- System Services Distributed with PCP
- Tools Distributed with PCP
- PCP Deployment Architectures
- Recommended Deployment Architecture
- Sizing Factors
- Configuration Options for PCP Scaling
- Example: Analyzing the Centralized Logging Deployment
- Example: Analyzing the Federated Setup Deployment
- Troubleshooting High Memory Usage

### Chapter 5. Logging Performance Data with PMLogger

- Modifying the PMLogger Configuration File with PMLOGCONF
- Editing the PMLogger Configuration File Manually
- Enabling the PMLogger Service
- Setting Up a Client System for Metrics Collection
- Setting Up a Central Server to Collect Data
- Replaying the PCP Log Archives with PMREP

### Chapter 6. Monitoring Performance with Performance Co-Pilot

- Monitoring Postfix with PMDA-Postfix
- Visually Tracing PCP Log Archives with the PCP Charts Application
- Collecting Data from SQL Server Using PCP
- Generating PCP Archives from SADC Archives

### Chapter 7. Performance Analysis of XFS with PCP

- Installing XFS PMDA Manually
- Examining XFS Performance Metrics with PMINFO
- Resetting XFS Performance Metrics with PMSTORE
- PCP Metric Groups for XFS
- Per-Device PCP Metric Groups for XFS

### Chapter 8. Setting Up Graphical Representation of PCP Metrics

- Setting Up PCP with PCP-ZEROCONF
- Setting Up a Grafana-Server
- Accessing the Grafana Web UI
- Configuring PCP Redis
- Creating Panels and Alert in PCP Redis Data Source
- Adding Notification Channels for Alerts
- Setting Up Authentication Between PCP Components
- Installing PCP BPFTrace
- Viewing the PCP BPFTrace System Analysis Dashboard
- Installing PCP Vector
20.4. PARAMETERS FOR RESERVING HUGETLB PAGES AT RUN TIME .......................... 167
20.5. CONFIGURING HUGETLB AT RUN TIME .............................................. 168
20.6. ENABLING TRANSPARENT HUGEPAGES ........................................... 169
20.7. DISABLING TRANSPARENT HUGEPAGES .......................................... 169
20.8. IMPACT OF PAGE SIZE ON TRANSLATION LOOKASIDE BUFFER SIZE ... 170

**CHAPTER 21. GETTING STARTED WITH SYSTEMTAP** ............................................ 171
21.1. THE PURPOSE OF SYSTEMTAP ............................................................... 171
21.2. INSTALLING SYSTEMTAP ...................................................................... 171
21.3. PRIVILEGES TO RUN SYSTEMTAP ....................................................... 172
21.4. RUNNING SYSTEMTAP SCRIPTS ............................................................. 173

**CHAPTER 22. CROSS-INSTRUMENTATION OF SYSTEMTAP** .......................... 174
22.1. SYSTEMTAP CROSS-INSTRUMENTATION ............................................... 174
22.2. Initializing CROSS-INSTRUMENTATION OF SYSTEMTAP .................. 175

**CHAPTER 23. MONITORING NETWORK ACTIVITY WITH SYSTEMTAP** ............ 177
23.1. PROFILING NETWORK ACTIVITY WITH SYSTEMTAP .......................... 177
23.2. TRACING FUNCTIONS CALLED IN NETWORK SOCKET CODE WITH SYSTEMTAP 178
23.3. MONITORING NETWORK PACKET DROPS WITH SYSTEMTAP ................ 179

**CHAPTER 24. PROFILING KERNEL ACTIVITY WITH SYSTEMTAP** .................. 180
24.1. COUNTING FUNCTION CALLS WITH SYSTEMTAP .................................. 180
24.2. TRACING FUNCTION CALLS WITH SYSTEMTAP .................................... 181
24.3. DETERMINING TIME SPENT IN KERNEL AND USER SPACE WITH SYSTEMTAP 182
24.4. MONITORING POLLING APPLICATIONS WITH SYSTEMTAP ................ 183
24.5. TRACKING MOST FREQUENTLY USED SYSTEM CALLS WITH SYSTEMTAP ... 184
24.6. TRACKING SYSTEM CALL VOLUME PER PROCESS WITH SYSTEMTAP ........ 184

**CHAPTER 25. MONITORING DISK AND I/O ACTIVITY WITH SYSTEMTAP** ........... 186
25.1. SUMMARIZING DISK READ/WRITE TRAFFIC WITH SYSTEMTAP .............. 186
25.2. TRACKING I/O TIME FOR EACH FILE READ OR WRITE WITH SYSTEMTAP ... 187
25.3. TRACKING CUMULATIVE I/O WITH SYSTEMTAP ..................................... 187
25.4. MONITORING I/O ACTIVITY ON A SPECIFIC DEVICE WITH SYSTEMTAP .... 188
25.5. MONITORING READS AND WRITES TO A FILE WITH SYSTEMTAP .......... 189
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.

- 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 feedback via Bugzilla, create a new 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. GETTING STARTED WITH TUNED

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

1.1. THE PURPOSE OF TUNED

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

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

- High throughput
- Low latency
- Saving power

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

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

1.2. TUNED PROFILES

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

The profiles provided with TuneD are divided into the following categories:

- Power-saving profiles
- Performance-boosting profiles

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

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

**Syntax of profile configuration**

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

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

**Additional resources**

- `tuned.conf(5)` man page.
1.3. THE DEFAULT TUNED PROFILE

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

<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>

Additional resources

- tuned.conf(5) man page.

1.4. MERGED TUNED PROFILES

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

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

Example 1.1. Low power consumption in a virtual guest

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

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

**WARNING**

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

Additional resources
1.5. THE LOCATION OF TUNED PROFILES

TuneD stores profiles in the following directories:

/usr/lib/tuned/

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

/etc/tuned/

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

Additional resources

- tuned.conf(5) man page.

1.6. TUNED PROFILES DISTRIBUTED WITH RHEL

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

**NOTE**

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

**balanced**

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

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

**powersave**

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

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

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

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

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

throughput-performance

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

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

accelerator-performance

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

latency-performance

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

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

network-latency

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

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

hpc-compute

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

network-throughput

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

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

virtual-guest

A profile designed for Red Hat Enterprise Linux 9.0 virtual machines and VMWare guests based on the Red Hat Enterprise Linux 9.0 Monitoring and managing system status and performance
A profile designed for Red Hat Enterprise Linux 9 virtual machines and VMWare guests based on the **throughput-performance** profile that, among other tasks, decreases virtual memory swappiness and increases disk readahead values. It does not disable disk barriers. It inherits the **throughput-performance** profile and changes the **energy_performance_preference** and **scaling_governor** attribute to the **performance** profile.

**virtual-host**

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

**oracle**

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

**desktop**

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

**optimize-serial-console**

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

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

**mssql**

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

**intel-sst**

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

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

### 1.7. TUNED CPU-PARTITIONING PROFILE

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

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

The following figure is an example to demonstrate how to use the **cpu-partitioning** profile. This example uses the CPU and node layout.
You can configure the cpu-partitioning profile in the /etc/tuned/cpu-partitioning-variables.conf file using the following configuration options:

**Isolated CPUs with load balancing**

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

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

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

**Isolated CPUs without load balancing**

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

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

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

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

**Housekeeping CPUs**

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

**Additional resources**

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

1.8. USING THE TUNED CPU-PARTITIONING PROFILE FOR LOW-LATENCY TUNING
This procedure describes how to tune a system for low-latency using the TuneD's `cpu-partitioning` profile. It uses the example of a low-latency application that can use `cpu-partitioning` and the CPU layout as mentioned in the `cpu-partitioning` figure.

The application in this case uses:

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

Prerequisites

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

Procedure

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

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

2. Set the `cpu-partitioning` TuneD profile:

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

3. Reboot

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

Additional resources

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

1.9. CUSTOMIZING THE CPU-PARTITIONING TUNED PROFILE

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

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

Procedure

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

   ```
   # mkdir /etc/tuned/my_profile
   ```
2. Create a `tuned.conf` file in this directory, and add the following content:

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

3. Use the new profile:

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

**NOTE**

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

### Additional resources

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

### 1.10. REAL-TIME TUNED PROFILES DISTRIBUTED WITH RHEL

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

The following real-time profiles are available:

- **realtime**
  - Use on bare-metal real-time systems.
  - Provided by the `tuned-profiles-realtime` package, which is available from the RT or NFV repositories.

- **realtime-virtual-host**
  - Use in a virtualization host configured for real-time.
  - Provided by the `tuned-profiles-nfv-host` package, which is available from the NFV repository.

- **realtime-virtual-guest**
  - Use in a virtualization guest configured for real-time.
  - Provided by the `tuned-profiles-nfv-guest` package, which is available from the NFV repository.

### 1.11. STATIC AND DYNAMIC TUNING IN TUNED

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 1.2. Static and dynamic tuning on a workstation

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

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

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

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

1.12. TUNED NO-DAEMON MODE

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

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

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

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

```
daemon = 0
```

1.13. INSTALLING AND ENABLING TUNED
This procedure installs and enables the **TuneD** application, installs **TuneD** profiles, and presets a default **TuneD** profile for your system.

**Procedure**

1. Install the **tuned** package:
   
   ```bash
   # dnf install tuned
   ```

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

3. Optionally, install **TuneD** profiles for real-time systems:
   
   ```bash
   # dnf install tuned-profiles-realtime tuned-profiles-nfv
   ```

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

**1.14. LISTING AVAILABLE TUNED PROFILES**

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

**Procedure**

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

  ```bash
  $ tuned-adm list
  ```

  **Available profiles:**
  - accelerator-performance - Throughput performance based tuning with disabled higher latency STOP states
  - balanced - General non-specialized tuned profile
  - desktop - Optimize for the desktop use-case
  - latency-performance - Optimize for deterministic performance at the cost of increased power consumption
  - network-latency - Optimize for deterministic performance at the cost of increased power consumption, focused on low latency network performance
  - network-throughput - Optimize for streaming network throughput, generally only necessary on older CPUs or 40G+ networks
  - powsave - 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
- tuned-adm(8) man page.

### 1.15. SETTING A TUNED PROFILE

This procedure activates a selected Tuned profile on your system.

**Prerequisites**
- The tuned service is running. See Installing and Enabling Tuned for details.

**Procedure**

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

   ```shell
   # tuned-adm recommend
   balanced
   ```

2. Activate a profile:

   ```shell
   # tuned-adm profile selected-profile
   ```

   Alternatively, you can activate a combination of multiple profiles:

   ```shell
   # tuned-adm profile profile1 profile2
   ```

**Example 1.3. A virtual machine optimized for low power consumption**

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

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

3. View the current active Tuned profile on your system:

   ```shell
   # 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**

- *tuned-adm(8)* man page

1.16. DISABLING TUNED

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

**Procedure**

- To disable all tunings temporarily:

```bash
# tuned-adm off
```

The tunings are applied again after the `tuned` service restarts.

- Alternatively, to stop and disable the `tuned` service permanently:

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

**Additional resources**

- *tuned-adm(8)* man page
CHAPTER 2. CUSTOMIZING TUNED PROFILES

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

Prerequisites

- Install and enable TuneD as described in Installing and Enabling TuneD for details.

2.1. TUNED PROFILES

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

The profiles provided with TuneD are divided into the following categories:

- Power-saving profiles
- Performance-boosting profiles

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

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

Syntax of profile configuration

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

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

Additional resources

- tuned.conf(5) man page.

2.2. THE DEFAULT TUNED PROFILE

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

<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>
</tbody>
</table>
### 2.3. MERGED TUNED PROFILES

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

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

#### Example 2.1. Low power consumption in a virtual guest

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

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

#### WARNING

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

### Additional resources

- [tuned.conf(5) man page](#)
- [tuned-adm man page](#)

### 2.4. THE LOCATION OF TUNED PROFILES

**TuneD** stores profiles in the following directories:

```
/usr/lib/tuned/
```

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

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

Additional resources

- tuned.conf(5) man page.

2.5. INHERITANCE BETWEEN TUNED PROFILES

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

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

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

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

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

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

Example 2.2. A power-saving profile based on balanced

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

```
[main]
include=balanced

[scsi_host]
alpm=min_power
```

Additional resources

- tuned.conf(5) man page

2.6. STATIC AND DYNAMIC TUNING IN TUNED

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 syslog settings and one-shot activation of several configuration tools such as ethtool.
Dynamic tuning

Watches how various system components are used throughout the uptime of your system. TuneD adjusts system settings dynamically based on that monitoring information. For example, the hard drive is used heavily during startup and login, but is barely used later when the user might mainly work with applications such as web browsers or email clients. Similarly, the CPU and network devices are used differently at different times. TuneD monitors the activity of these components and reacts to the changes in their use.

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

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

Example 2.3. Static and dynamic tuning on a workstation

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

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

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

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

2.7. TUNED PLUG-INS

Plug-ins are modules in TuneD profiles that TuneD uses to monitor or optimize different devices on the system. TuneD uses two types of plug-ins:

Monitoring plug-ins

Monitoring plug-ins are used to get information from a running system. The output of the monitoring plug-ins can be used by tuning plug-ins for dynamic tuning. Monitoring plug-ins are automatically instantiated whenever their metrics are needed by any of the enabled tuning plug-ins. If two tuning plug-ins require the same data, only one instance of the monitoring plug-in is created and the data is shared.

Tuning plug-ins

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

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

**NAME**

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

**TYPE**

is the type of the tuning plug-in.

**DEVICES**

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

**Example 2.4. Matching block devices with a plug-in**

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

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

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

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

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

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

**Short plug-in syntax**

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

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

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

**Example 2.5. Matching block devices using the short syntax**
Conflicting plug-in definitions in a profile
If the same section is specified more than once using the `include` option, the settings are merged. If they cannot be merged due to a conflict, the last conflicting definition overrides the previous settings. If you do not know what was previously defined, you can use the `replace` Boolean option and set it to `true`. This causes all the previous definitions with the same name to be overwritten and the merge does not happen.

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

NOTE

TuneD includes the ability to run any shell command as part of enabling or disabling a tuning profile. This enables you to extend TuneD profiles with functionality that has not been integrated into TuneD yet.
You can specify arbitrary shell commands using the `script` plug-in.

Additional resources

- `tuned.conf(5)` man page

2.8. AVAILABLE TUNED PLUG-INS

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 2.6. Running a Bash script from a profile**

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

```bash
[script]
script=${i:PROFILE_DIR}/script.sh
```
sysfs
Sets various sysfs settings specified by the plug-in options.
The syntax is name=value, where name is the sysfs path to use.
Use this plugin in case you need to change some settings that are not covered by other plug-ins. Prefer specific plug-ins if they cover the required settings.

video
Sets various powersave levels on video cards. Currently, only the Radeon cards are supported.
The powersave level can be specified by using the radeon_powersave option. Supported values are:

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

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

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

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

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

The kernel options can be specified by the following syntax:

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

Example 2.7. Modifying the kernel command line
For example, to add the quiet kernel option to a TuneD profile, include the following lines in the tuned.conf file:
The following is an example of a custom profile that adds the `isolcpus=2` option to the kernel command line:

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

## 2.9. VARIABLES IN TUNED PROFILES

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

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

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

```ini
[variables]
variable_name=value
```

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

```
${variable_name}
```

### Example 2.8. 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:

```ini
[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`:

```ini
[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.
2.10. BUILT-IN FUNCTIONS IN TUNED PROFILES

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

You can:

- Use various built-in functions together with TuneD variables
- Create custom functions in Python and add them to TuneD in the form of plug-ins

To call a function, use the following syntax:

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

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

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

Example 2.9. 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:

```
[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.

2.11. BUILT-IN FUNCTIONS AVAILABLE IN TUNED PROFILES

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

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

- **exec**
  Executes a process and returns its output.

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

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

kb2s
Converts kilobytes to disk sectors.

s2kb
Converts disk sectors to kilobytes.

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

virt_check
Checks whether TuneD is running inside a virtual machine (VM) or on bare metal:
- Inside a VM, the function returns the first argument.
- On bare metal, the function returns the second argument, even in case of an error.

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

cpulist2hex
Converts a CPU list to a hexadecimal CPU mask.

cpulist2hex_invert
Converts a CPU list to a hexadecimal CPU mask and inverts it.

hex2cpulist
Converts a hexadecimal CPU mask to a CPU list.

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

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

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

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

2.12. CREATING NEW TUNED PROFILES

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

Prerequisites
- The tuned service is running. See Installing and Enabling TuneD for details.

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

   ```bash
   # 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

- `tuned.conf(5)` man page

### 2.13. MODIFYING EXISTING TUNED PROFILES

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

**Prerequisites**

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

1. In the `/etc/tuned/` directory, create a new directory named the same as the profile that you want to create:
   
   ```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 2.10. 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

- `tuned.conf(5) man page`

2.14. SETTING THE DISK SCHEDULER USING TUNED

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

In the following commands and configuration, replace:
- device with the name of the block device, for example sdf
- selected-scheduler with the disk scheduler that you want to set for the device, for example bfq

**Prerequisites**
- The tuned service is installed and enabled. For details, see Installing and enabling TuneD.

**Procedure**

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

   $ tuned-adm active

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

   # mkdir /etc/tuned/my-profile

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

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

   ID_WWN=0x5002538d00000000
   ID_SERIAL=Generic-_SD_MMC_20120501030900000-0:0
   ID_SERIAL_SHORT=20120501030900000

   **NOTE**

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

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

   a. Optional: Include an existing profile:

      [main]
      include=existing-profile

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

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

      Here:

      - Replace IDNAME with the name of the identifier being used (for example, ID_WWN).
• Replace *device system unique id* with the value of the chosen identifier (for example, 0x5002538d00000000).
  
  To match multiple devices in the `devices_udev_regex` option, enclose the identifiers in parentheses and separate them with vertical bars:

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

5. Enable your profile:

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

**Verification steps**

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

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

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

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

```# cat /sys/block/device/queue/scheduler
[mq-deadline] kyber bfq none
```

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

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

**Additional resources**

- *Customizing TuneD profiles.*
CHAPTER 3. MONITORING PERFORMANCE USING RHEL SYSTEM ROLES

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

3.1. 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 9, the interface currently consists of the following roles:

- Certificate Issuance and Renewal
- Kernel Settings
- Metrics
- Network Bound Disk Encryption client and Network Bound Disk Encryption server
- Networking
- Postfix
- SSH client
- SSH server
- System-wide Cryptographic Policies
- Terminal Session Recording

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

Additional resources

- Red Hat Enterprise Linux (RHEL) System Roles
- Documentation in the /usr/share/doc/rhel-system-roles/ directory [1]

3.2. RHEL SYSTEM ROLES TERMINOLOGY

You can find the following terms across this documentation:

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.

3.3. INSTALLING RHEL SYSTEM ROLES IN YOUR SYSTEM

To use the RHEL System Roles, install the required packages in your system.

Prerequisites

- The Ansible Core package is installed on the control machine.
- 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:

   # dnf install rhel-system-roles

2. Install the Ansible Core package:

   # dnf install ansible-core

The Ansible Core package provides the **ansible-playbook** CLI, the Ansible Vault functionality, and the basic modules and filters required by RHEL Ansible content.

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

Additional resources

- The Red Hat Enterprise Linux (RHEL) System Roles
- The **ansible-playbook** man page.

3.4. APPLYING A ROLE

The following procedure describes how to apply a particular role.

Prerequisites

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

  # dnf install rhel-system-roles

1. Install the Ansible Core package:
# dnf install ansible-core

The Ansible Core package provides the ansible-playbook CLI, the Ansible Vault functionality, and the basic modules and filters required by RHEL Ansible content.

- Ensure that you are able to create an Ansible inventory.
  Inventories represent the hosts, host groups, and some of the configuration parameters used by the Ansible playbooks.

Playbooks are typically human-readable, and are defined in ini, yaml, json, and other file formats.

- Ensure that 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 are defined in the yaml format.

**Procedure**

1. Create the required Ansible inventory containing the hosts and groups that you want to manage. Here is an example using a file called inventory.ini of a group of hosts called webservers:

   ```ini
   [webservers]
   host1
   host2
   host3
   ```

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

   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.postfix
   ```
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 `postfix`, `metrics`, `network`, `tlog`, or `ssh`.

3. To execute the playbook on specific hosts, you must perform one of the following:

- Edit the playbook to use `hosts: host1[,host2,...]`, or `hosts: all`, and execute the command:
  
  ```bash
  # ansible-playbook name.of.the.playbook
  ```

- Edit the inventory to ensure that the hosts you want to use are defined in a group, and execute the command:
  
  ```bash
  # ansible-playbook -i name.of.the.inventory name.of.the.playbook
  ```

- Specify all hosts when executing the `ansible-playbook` command:
  
  ```bash
  # ansible-playbook -i host1,host2,... name.of.the.playbook
  ```

IMPORTANT

Be aware that the `-i` flag specifies the inventory of all hosts that are available. If you have multiple targeted hosts, but want to select a host against which you want to run the playbook, you can add a variable in the playbook to be able to select a host. For example:

Ansible Playbook | example-playbook.yml:

```yaml
- hosts: "{{ target_host }}"
  roles:
  - rhel-system-roles.network
  - rhel-system-roles.postfix
```

Playbook execution command:

```bash
# ansible-playbook -i host1,..hostn -e target_host=host5 example-playbook.yml
```

Additional resources

- Ansible playbooks
- Using roles in Ansible playbook
3.5. 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 3.1. 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 <code>/var/log</code> for each host.</td>
<td><code>metrics_monitored_hosts: [&quot;webserver.example.com&quot;, &quot;database.example.com&quot;]</code></td>
</tr>
<tr>
<td>metrics_retention_days</td>
<td>Configures the number of days for performance data retention before deletion.</td>
<td><code>metrics_retention_days: 14</code></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 <code>pcp</code> and <code>grafana</code>. Set to false by default.</td>
<td><code>metrics_graph_service: no</code></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 <code>pcp</code> metrics via <code>redis</code>. Set to false by default.</td>
<td><code>metrics_query_service: no</code></td>
</tr>
<tr>
<td>metrics_provider</td>
<td>Specifies which metrics collector to use to provide metrics. Currently, <code>pcp</code> is the only supported metrics provider.</td>
<td><code>metrics_provider: &quot;pcp&quot;</code></td>
</tr>
</tbody>
</table>

**NOTE**

For details about the parameters used in `metrics_connections` and additional information about the Metrics System Role, see the `/usr/share/ansible/roles/rhel-system-roles.metrics/README.md` file.
3.6. 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

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

Procedure

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

```
localhost ansible_connection=local
```

2. Create an Ansible playbook with the following content:

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

3. Run the Ansible playbook:

```
# 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.

3.7. 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

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

Procedure

1. Add the name or IP 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]
webserver.example.com
database.example.com
```

2. Create an Ansible playbook with the following content:

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

3. Run the Ansible playbook:

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

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

3.8. 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

- The Ansible Core package is installed on the control machine.

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

Procedure

1. Create an Ansible playbook with the following content:

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

```yaml
# 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.

### 3.9. Setting up authentication while monitoring a system using the metrics system role

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

**Prerequisites**

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

**Procedure**

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

```yaml
---
vars:
  metrics_username: your_username
  metrics_password: your_password
```

2. Run the Ansible playbook:

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

**Verification steps**

- Verify the `sasl` configuration:

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

*`ip_address` should be replaced by the IP address of the host.*
3.10. USING THE METRICS SYSTEM ROLE TO CONFIGURE AND ENABLE METRICS COLLECTION FOR SQL SERVER

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

**Prerequisites**

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

**Procedure**

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

   ```
   localhost ansible_connection=local
   ```

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

   ```
   ---
   - hosts: localhost
     roles:
     - role: rhel-system-roles.metrics
       vars:
         metrics_from_mssql: yes
   ```

3. Run the Ansible playbook:

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

**Verification steps**

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

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

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

[1] This documentation is installed automatically with the `rhel-system-roles` package.
CHAPTER 4. SETTING UP PCP

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

This section describes how to install and enable PCP on your system.

4.1. OVERVIEW OF PCP

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

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

Features of PCP:

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

PCP has the following components:

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

Additional resources

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

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

This procedure describes how to install PCP using the `pcp` package. If you want to automate the PCP installation, install it using the `pcp-zeroconf` package. For more information on installing PCP by using `pcp-zeroconf`, see Setting up PCP with `pcp-zeroconf`.

**Procedure**

1. Install the `pcp` package:
   ```
   # dnf install pcp
   ```

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

**Verification steps**

- Verify if the `pmcd` process is running on the host:
  ```
  # pcp
  ```

**Performance Co-Pilot configuration on workstation:**

platform: Linux workstation 4.18.0-80.el8.x86_64 #1 SMP Wed Mar 13 12:02:46 UTC 2019 x86_64

hardware: 12 cpus, 2 disks, 1 node, 36023MB RAM
timezone: CEST-2
services: pmcd
pmcd: Version 4.3.0-1, 8 agents
pmda: root pmcd proc xfs linux mmv kvm jbd2

**Additional resources**

- `pmcd(1)` man page
- Tools distributed with PCP

### 4.3. DEPLOYING A MINIMAL PCP SETUP

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

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

**Prerequisites**

- PCP is installed. For more information, see Installing and enabling PCP.
Procedure

1. Update the pmlogger configuration:

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

2. Start the pmcd and pmlogger services:

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

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

4. Stop the pmcd and pmlogger services:

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

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

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

   Extract this file and analyze the data using PCP tools.

Additional resources

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

### 4.4. SYSTEM SERVICES DISTRIBUTED WITH PCP

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

**Table 4.1. Roles of 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>
</tbody>
</table>
pmproxy | The realtime and historical performance metrics proxy, time series query and REST API service.

4.5. TOOLS DISTRIBUTED WITH PCP

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

Table 4.2. Usage of 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-atopsar</td>
<td>Generates a system-level activity report over a variety of system resource utilization. The report is generated from a raw logfile previously recorded using pmlogger or the -w option of pcp-atop.</td>
</tr>
<tr>
<td>pcp-dmcache</td>
<td>Displays information about configured Device Mapper Cache targets, such as: device IOPs, cache and metadata device utilization, as well as hit and miss rates and ratios for both reads and writes for each cache device.</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>pcp-free</td>
<td>Reports on free and used memory in a system.</td>
</tr>
<tr>
<td>pcp-htop</td>
<td>Displays all processes running on a system along with their command line arguments in a manner similar to the top command, but allows you to scroll vertically and horizontally as well as interact using a mouse. You can also view processes in a tree format and select and act on multiple processes at once.</td>
</tr>
<tr>
<td>pcp-ipcs</td>
<td>Displays information on the inter-process communication (IPC) facilities that the calling process has read access for.</td>
</tr>
<tr>
<td>pcp-numastat</td>
<td>Displays NUMA allocation statistics from the kernel memory allocator.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>pcp-pidstat</strong></td>
<td>Displays information about individual tasks or processes running on the system such as: CPU percentage, memory and stack usage, scheduling, and priority. Reports live data for the local host by default.</td>
</tr>
<tr>
<td><strong>pcp-ss</strong></td>
<td>Displays socket statistics collected by the pmdasockets Performance Metrics Domain Agent (PMDA).</td>
</tr>
<tr>
<td><strong>pcp-uptime</strong></td>
<td>Displays how long the system has been running, how many users are currently logged on, and the system load averages for the past 1, 5, and 15 minutes.</td>
</tr>
<tr>
<td><strong>pcp-vmstat</strong></td>
<td>Provides a high-level system performance overview every 5 seconds. Displays information about processes, memory, paging, block IO, traps, and CPU activity.</td>
</tr>
<tr>
<td><strong>pmchart</strong></td>
<td>Plots performance metrics values available through the facilities of the Performance Co-Pilot.</td>
</tr>
<tr>
<td><strong>pmclient</strong></td>
<td>Displays high-level system performance metrics by using the Performance Metrics Application Programming Interface (PMAPI).</td>
</tr>
<tr>
<td><strong>pmconfig</strong></td>
<td>Displays the values of configuration parameters.</td>
</tr>
<tr>
<td><strong>pmdbg</strong></td>
<td>Displays available Performance Co-Pilot debug control flags and their values.</td>
</tr>
<tr>
<td><strong>pmdiff</strong></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><strong>pmdumplog</strong></td>
<td>Displays control, metadata, index, and state information from a Performance Co-Pilot archive file.</td>
</tr>
<tr>
<td><strong>pmdumptext</strong></td>
<td>Outputs the values of performance metrics collected live or from a Performance Co-Pilot archive.</td>
</tr>
<tr>
<td><strong>pmerr</strong></td>
<td>Displays available Performance Co-Pilot error codes and their corresponding error messages.</td>
</tr>
<tr>
<td><strong>pmfind</strong></td>
<td>Finds PCP services on the network.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</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>pmiectl</td>
<td>Manages non-primary instances of pmie.</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>pmlogctl</td>
<td>Manages non-primary instances of pmlogger.</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>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 hosts 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 PMDA.</td>
</tr>
<tr>
<td>pmval</td>
<td>Displays the current value of a performance metric.</td>
</tr>
</tbody>
</table>
4.6. PCP DEPLOYMENT ARCHITECTURES

Performance Co-Pilot (PCP) offers many options to accomplish advanced setups. From the huge variety of possible architectures, this section describes how to scale your PCP deployment based on the recommended deployment setup by Red Hat, sizing factors, and configuration options.

PCP supports multiple deployment architectures, based on the scale of the PCP deployment.

Available scaling deployment setup variants:

**Localhost**

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

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

**Decentralized**

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

![Figure 4.1. Decentralized logging](image)

**Centralized logging - pmlogger farm**

When the resource usage on the monitored hosts is constrained, another deployment option is a `pmlogger` farm, which is also known as centralized logging. In this setup, a single logger host executes multiple `pmlogger` processes, and each is configured to retrieve performance metrics from a different remote `pmcd` host. The centralized logger host is also configured to execute the `pmproxy` service, which discovers the resulting PCP archives logs and loads the metric data into a Redis instance.
Figure 4.2. Centralized logging - pmlogger farm

For large scale deployments, Red Hat recommends to deploy multiple pmlogger farms in a federated fashion. For example, one pmlogger farm per rack or data center. Each pmlogger farm loads the metrics into a central Redis instance.
By default, the deployment setup for Redis is standalone, localhost. However, Redis can optionally perform in a highly-available and highly scalable clustered fashion, where data is shared across multiple hosts. Another viable option is to deploy a Redis cluster in the cloud, or to utilize a managed Redis cluster from a cloud vendor.
Additional resources

- `pcp(1)`, `pmlogger(1)`, `pmproxy(1)`, and `pmcd(1)` man pages
- Recommended deployment architecture

4.7. RECOMMENDED DEPLOYMENT ARCHITECTURE

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

Table 4.3. Recommended deployment architecture

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

4.8. SIZING FACTORS

The following are the sizing factors required for scaling:

Remote system size

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

Logged Metrics

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

Logging Interval

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

`pmlogrewrite`

Red Hat Enterprise Linux 9.0 Monitoring and managing system status and performance
After every PCP upgrade, the `pmlogrewrite` tool is executed and rewrites old archives if there were changes in the metric metadata from the previous version and the new version of PCP. This process duration scales linear with the number of archives stored.

Additional resources

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

### 4.9. CONFIGURATION OPTIONS FOR PCP SCALING

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

#### sysctl and rlimit settings

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

#### Local Archives

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

#### Redis

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

- **stream.expire** specifies the duration when stale metrics should be removed, that is metrics which were not updated in a specified amount of time in seconds.

- **stream.maxlen** specifies the maximum number of metric values for one metric per host. This setting should be the retention time divided by the logging interval, for example 20160 for 14 days of retention and 60s logging interval (60*60*24*14/60)

Additional resources

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

### 4.10. EXAMPLE: ANALYZING THE CENTRALIZED LOGGING DEPLOYMENT

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

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

Table 4.4. Detailed utilization statistics for 10s logging interval

<table>
<thead>
<tr>
<th>Number of Hosts</th>
<th>10</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCP Archives Storage per Day</td>
<td>91 MB</td>
<td>522 MB</td>
</tr>
<tr>
<td><strong>pmlogger Memory</strong></td>
<td>160 MB</td>
<td>580 MB</td>
</tr>
<tr>
<td><strong>pmlogger Network per Day (In)</strong></td>
<td>2 MB</td>
<td>9 MB</td>
</tr>
<tr>
<td><strong>pmproxy Memory</strong></td>
<td>1.4 GB</td>
<td>6.3 GB</td>
</tr>
<tr>
<td>Redis Memory per Day</td>
<td>2.6 GB</td>
<td>12 GB</td>
</tr>
</tbody>
</table>

Table 4.5. Used resources depending on monitored hosts for 60s logging interval

<table>
<thead>
<tr>
<th>Number of Hosts</th>
<th>10</th>
<th>50</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCP Archives Storage per Day</td>
<td>20 MB</td>
<td>120 MB</td>
<td>271 MB</td>
</tr>
<tr>
<td><strong>pmlogger Memory</strong></td>
<td>104 MB</td>
<td>524 MB</td>
<td>1049 MB</td>
</tr>
<tr>
<td><strong>pmlogger Network per Day (In)</strong></td>
<td>0.38 MB</td>
<td>1.75 MB</td>
<td>3.48 MB</td>
</tr>
<tr>
<td><strong>pmproxy Memory</strong></td>
<td>2.67 GB</td>
<td>5.5 GB</td>
<td>9 GB</td>
</tr>
<tr>
<td>Redis Memory per Day</td>
<td>0.54 GB</td>
<td>2.65 GB</td>
<td>5.3 GB</td>
</tr>
</tbody>
</table>

NOTE

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

4.11. EXAMPLE: ANALYZING THE FEDERATED SETUP DEPLOYMENT

The following results were observed on a federated setup, also known as multiple pmlogger farms, consisting of three centralized logging (pmlogger farm) setups, where each pmlogger farm was monitoring 100 remote hosts, that is 300 hosts in total.
This setup of the pmlogger farms is identical to the configuration mentioned in the Example: Analyzing the centralized logging deployment for 60s logging interval, except that the Redis servers were operating in cluster mode.

Table 4.6. Used resources depending on federated hosts for 60s logging interval

<table>
<thead>
<tr>
<th>PCP Archives Storage per Day</th>
<th>pmlogger Memory</th>
<th>Network per Day (In/Out)</th>
<th>pmproxy Memory</th>
<th>Redis Memory per Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>277 MB</td>
<td>1058 MB</td>
<td>15.6 MB / 12.3 MB</td>
<td>6-8 GB</td>
<td>5.5 GB</td>
</tr>
</tbody>
</table>

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

4.12. TROUBLESHOOTING HIGH MEMORY USAGE

The following scenarios can result in high memory usage:

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

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

This section describes how to troubleshoot high memory usage issue.

Prerequisites

1. Install the pcp-pmda-redis package:

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

2. Install the redis PMDA:

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

Procedure

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

  ```
  $ pmrep :pmproxy
      backlog inflight reqs/s resp/s wait req err resp err changed throttled
      byte    count count/s count/s s/s count/s count/s count/s count/s
  14:59:08 0     0     N/A     N/A     N/A     N/A     N/A     N/A     N/A     N/A
  14:59:09 0     0     2268.9  2268.9  28     0     0     2.0     4.0
  14:59:10 0     0     0.0     0.0     0     0     0     0.0     0.0
  14:59:11 0     0     0.0     0.0     0     0     0     0.0     0.0
  ```
This column shows how many Redis requests are in-flight, which means they are queued or sent, and no reply was received so far.

A high number indicates one of the following conditions:

- The **pmproxy** process is busy processing new PCP archives and does not have spare CPU cycles to process Redis requests and responses.

- The Redis node or cluster is overloaded and cannot process incoming requests on time.

- To troubleshoot the high memory usage issue, reduce the number of **pmlogger** processes for this farm, and add another pmlogger farm. Use the federated - multiple pmlogger farms setup. If the Redis node is using 100% CPU for an extended amount of time, move it to a host with better performance or use a clustered Redis setup instead.

- To view the **pmproxy.redis.*** metrics, use the following command:

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

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

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

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

  ```
  $ pminfo -ftd redis
  redis.redis_build_id [Build ID]
  Data Type: string InDom: 24.0 0x6000000
  Semantics: discrete Units: count
  inst [0 or "localhost:6379"] value "87e335e57cfa755"
  redis.total_commands_processed [Total number of commands processed by the server]
  Data Type: 64-bit unsigned int InDom: 24.0 0x6000000
  Semantics: counter Units: count
  ```
inst [0 or "localhost:6379"] value 595627069

redis.used_memory_peak [Peak memory consumed by Redis (in bytes)]
Data Type: 32-bit unsigned int  InDom: 24.0 0x6000000
Semantics: instant  Units: count
inst [0 or "localhost:6379"] value 572234920

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

Additional resources

- pmdaredis(1), pmproxy(1), and pminfo(1) man pages
- PCP deployment architectures
CHAPTER 5. 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

5.1. MODIFYING THE PMLOGGER CONFIGURATION FILE WITH PMLOGCONF

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

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

Prerequisites

- PCP is installed. For more information, see Installing and enabling PCP.

Procedure

1. Create or modify the pmlogger configuration file:

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

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

Additional resources

- pmlogconf(1) and pmlogger(1) man pages
- Tools distributed with PCP
- System services distributed with PCP

5.2. EDITING THE PMLOGGER CONFIGURATION FILE MANUALLY

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

In manual configuration, you can:

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

Prerequisites

- PCP is installed. For more information, see Installing and enabling PCP.

Procedure

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

```plaintext
# It is safe to make additions from here on ...
#
log mandatory on every 5 seconds {
  xfs.write
  xfs.write_bytes
  xfs.read
  xfs.read_bytes
}

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

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

Additional resources

- `pmlogger(1)` man page
- Tools distributed with PCP
- System services distributed with PCP

5.3. ENABLING THE PMLOGGER SERVICE

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

This procedure describes how to enable the pmlogger service.

Prerequisites

- PCP is installed. For more information, see Installing and enabling PCP.

Procedure

- Start and enable the pmlogger service:
# systemctl start pmlogger
# systemctl enable pmlogger

Verification steps

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

```
# pcp
```

Performance Co-Pilot configuration on workstation:

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

Additional resources

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

### 5.4. SETTING UP A CLIENT SYSTEM FOR METRICS COLLECTION

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

**Prerequisites**

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

**Procedure**

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

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

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

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

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

By default, **pmcd** is listening on the localhost.
3. Configure the firewall to add the public zone permanently:

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

# firewall-cmd --reload
success
```

4. Set an SELinux boolean:

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

5. Enable the pmcd and pmlogger services:

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

**Verification steps**

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

```
# ss -t Vulkan
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))
```

**Additional resources**

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

**5.5. SETTING UP A CENTRAL SERVER TO COLLECT DATA**

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

**Prerequisites**

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

**Procedure**

1. Install the pcp-system-tools package:

```
# dnf install pcp-system-tools
```
2. Create the `/etc/pcp/pmlogger/control.d/remote` file with the following content:

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

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

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

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

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

**Verification steps**

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

```bash
# for i in /var/log/pcp/pmlogger/rhel*/*.0; do pmdumplog -L $i; done
```

Log Label (Log Format Version 2)

Performance metrics from host rhel6u10a.local


ending Mon Nov 25 22:06:04.874 2019

Archive timezone: JST-9

PID for pmlogger: 24002

Log Label (Log Format Version 2)

Performance metrics from host rhel7u4a

commencing Tue Nov 26 06:49:24.954 2019

ending Tue Nov 26 07:06:24.979 2019

Archive timezone: CET-1

PID for pmlogger: 10941

[..]

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

**Additional resources**

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

## 5.6. REPLAYING THE PCP LOG ARCHIVES WITH PMREP

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

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

**Prerequisites**

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

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

**Procedure**

- Display the data on the metric:

  ```bash
  $ pmrep --start @3:00am --archive 20211128 --interval 5seconds --samples 10 --output csv disk.dev.write
  Time,"disk.dev.write-sda","disk.dev.write-sdb"
  2021-11-28 03:00:00,4.000,5.200
  2021-11-28 03:00:05,1.600,7.600
  2021-11-28 03:00:10,0.800,7.100
  2021-11-28 03:00:15,15.600,7.200
  2021-11-28 03:00:20,21.400,7.200
  2021-11-28 03:00:25,21.200,6.800
  2021-11-28 03:00:30,21.000,27.600
  2021-11-28 03:00:35,12.400,33.800
  2021-11-28 03:00:40,9.800,20.600
  ```

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

  **NOTE**

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

**Additional resources**

- `pmlogger(1)`, `pmrep(1)`, and `pmlogsummary(1)` man pages
- Tools distributed with PCP
- System services distributed with PCP
CHAPTER 6. MONITORING PERFORMANCE WITH PERFORMANCE CO-PILOT

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

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

6.1. MONITORING POSTFIX WITH PMDA-POSTFIX

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

Prerequisites

- PCP is installed. For more information, see Installing and enabling PCP.
- The pmlogger service is enabled. For more information, see Enabling the pmlogger service.

Procedure

1. Install the following packages:
   a. Install the pcp-system-tools:
      ```
      # dnf install pcp-system-tools
      ```
   b. Install the pmda-postfix package to monitor postfix:
      ```
      # dnf install pcp-pmda-postfix postfix
      ```
   c. Install the logging daemon:
      ```
      # dnf install rsyslog
      ```
   d. Install the mail client for testing:
      ```
      # dnf install mutt
      ```
2. Enable the postfix and rsyslog services:
   ```
   # systemctl enable postfix rsyslog
   # systemctl restart postfix rsyslog
   ```
3. Enable the SELinux boolean, so that pmda-postfix can access the required log files:
   ```
   # setsebool -P pcp_read_generic_logs=on
   ```
4. Install the PMDA:
   ```
   # cd /var/lib/pcp/pmdas/postfix/
   ```
# ./Install

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

Verification steps

- Verify the `pmda-postfix` operation:

  ```
  echo testmail | mutt root
  ```

- Verify the available metrics:

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

Additional resources

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

### 6.2. VISUALLY TRACING PCP LOG ARCHIVES WITH THE PCP CHARTS APPLICATION

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

Using the PCP Charts application, you can:

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

- PCP is installed. For more information, see Installing and enabling PCP.
- Logged performance data with the pmlogger. For more information, see Logging performance data with pmlogger.
- Install the pcp-gui package:
  
  ```
  # dnf install pcp-gui
  ```

Procedure

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

  ```
  # pmchart
  ```

Figure 6.1. PCP Charts application

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

- The interval in which PCP polls the metric data
- The date and time for the metrics of historical data

2. Click File and then New Chart to select metric from both the local machine and remote machines by specifying their host name or address. Advanced configuration options include the ability to manually set the axis values for the chart, and to manually choose the color of the plots.

3. Record the views created in the PCP Charts application:

   Following are the options to take images or record the views created in the PCP Charts application:

   - Click File and then Export to save an image of the current view.
• Click **Record** and then **Start** to start a recording. Click **Record** and then **Stop** to stop the recording. After stopping the recording, the recorded metrics are archived to be viewed later.

4. Optional: In the **PCP Charts** application, the main configuration file, known as the **view**, allows the metadata associated with one or more charts to be saved. This metadata describes all chart aspects, including the metrics used and the chart columns. Save the custom **view** configuration by clicking **File** and then **Save View**, and load the **view** configuration later.

The following example of the **PCP Charts** application view configuration file describes a stacking chart graph showing the total number of bytes read and written to the given XFS file system **loop1**:

```plaintext
#kmchart
version 1

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

**Additional resources**

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

### 6.3. COLLECTING DATA FROM SQL SERVER USING PCP

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

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

**Prerequisites**

- You have installed Microsoft SQL Server for Red Hat Enterprise Linux and established a ‘trusted’ connection to an SQL server.
- You have installed the Microsoft ODBC driver for SQL Server for Red Hat Enterprise Linux.

**Procedure**

1. Install PCP:

   ```bash
   # dnf install pcp-zeroconf
   ```

2. Install packages required for the **pyodbc** driver:

   ```bash
   # dnf install python3-pyodbc
   ```

3. Install the **mssql** agent:

   a. Install the Microsoft SQL Server domain agent for PCP:

      ```bash
      # dnf install pcp-pmda-mssql
      ```
b. Edit the `/etc/pcp/mssql/mssql.conf` file to configure the SQL server account’s username and password for the `mssql` agent. Ensure that the account you configure has access rights to performance data.

   username: `user_name`
   password: `user_password`

Replace `user_name` with the SQL Server account and `user_password` with the SQL Server user password for this account.

4. Install the agent:

   # cd /var/lib/pcp/pmdas/mssql
   # ./Install
   Updating the Performance Metrics Name Space (PMNS) ...
   Terminate PMDA if already installed ...
   Updating the PMCD control file, and notifying PMCD ...
   Check mssql metrics have appeared ... 168 metrics and 598 values

**Verification steps**

- Using the `pcp` command, verify if the SQL Server PMDA (`mssql`) is loaded and running:

  
  $ pcp
  Performance Co-Pilot configuration on rhel.local:

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

  
  - View the complete list of metrics that PCP can collect from the SQL Server:

    # pminfo mssql

  - After viewing the list of metrics, you can report the rate of transactions. For example, to report on the overall transaction count per second, over a five second time window:

    # pmval -t 1 -T 5 mssql.databases.transactions

  - View the graphical chart of these metrics on your system by using the `pmchart` command. For more information, see `Visually tracing PCP log archives with the PCP Charts application`.

**Additional resources**

- `pcp(1), pminfo(1), pmval(1), pmchart(1), and pmdamssql(1)` man pages
6.4. GENERATING PCP ARCHIVES FROM SADC ARCHIVES

You can use the sadf tool provided by the sysstat package to generate PCP archives from native sadc archives.

Prerequisites

- A sadc archive has been created:

```
# /usr/lib64/sa/sadc 1 5 -
```

In this example, sadc is sampling system data 1 time in a 5 second interval. The outfile is specified as - which results in sadc writing the data to the standard system activity daily data file. This file is named saDD and is located in the /var/log/sa directory by default.

Procedure

- Generate a PCP archive from a sadc archive:

```
# sadf -l -O pcparchive=/tmp/recording -2
```

In this example, using the -2 option results in sadf generating a PCP archive from a sadc archive recorded 2 days ago.

Verification steps

You can use PCP commands to inspect and analyze the PCP archive generated from a sadc archive as you would a native PCP archive. For example:

- To show a list of metrics in the PCP archive generated from an sadc archive archive, run:

```
$ pminfo --archive /tmp/recording
Disk.dev.avactive
Disk.dev.read
Disk.dev.write
Disk.dev.blkread
[...]
```

- To show the timespace of the archive and hostname of the PCP archive, run:

```
$ pmdumplog --label /tmp/recording
Log Label (Log Format Version 2)
Performance metrics from host shard
    commencing Tue Jul 20 00:10:30.642477 2021
    ending     Wed Jul 21 00:10:30.222176 2021
```

- To plot performance metrics values into graphs, run:

```
$ pmchart --archive /tmp/recording
```
CHAPTER 7. PERFORMANCE ANALYSIS OF XFS WITH PCP

The XFS PMDA ships as part of the `pcp` package and is enabled by default during the installation. It is used to gather performance metric data of XFS file systems in Performance Co-Pilot (PCP).

This section describes how to analyze XFS file system's performance using PCP.

7.1. INSTALLING XFS PMDA MANUALLY

If the XFS PMDA is not listed in the `pcp` configuration output, install the PMDA agent manually.

This procedure describes how to manually install the PMDA agent.

Prerequisites

- PCP is installed. For more information, see Installing and enabling PCP.

Procedure

1. Navigate to the xfs directory:

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

Verification steps

- Verify that the `pmcd` process is running on the host and the XFS PMDA is listed as enabled in the configuration:

  ```bash
  # pcp
  ```

  Performance Co-Pilot configuration on workstation:

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

Additional resources

- `pmcd(1)` man page
- Tools distributed with PCP

7.2. EXAMINING XFS PERFORMANCE METRICS WITH PMINFO

PCP enables XFS PMDA to allow the reporting of certain XFS metrics per each of the mounted XFS file systems. This makes it easier to pinpoint specific mounted file system issues and evaluate performance.

The `pminfo` command provides per-device XFS metrics for each mounted XFS file system.
This procedure displays a list of all available metrics provided by the XFS PMDA.

Prerequisites

- PCP is installed. For more information, see Installing and enabling PCP.

Procedure

- Display the list of all available metrics provided by the XFS PMDA:
  ```
  # pminfo xfs
  ```

- Display information for the individual metrics. The following examples examine specific XFS read and write metrics using the pminfo tool:
  - Display a short description of the `xfs.write_bytes` metric:
    ```
    # pminfo --oneline xfs.write_bytes
    xfs.write_bytes [number of bytes written in XFS file system write operations]
    ```
  - Display a long description of the `xfs.read_bytes` metric:
    ```
    # pminfo --helptext xfs.read_bytes
    xfs.read_bytes
    Help:
    This is the number of bytes read via read(2) system calls to files in XFS file systems. It can be used in conjunction with the read_calls count to calculate the average size of the read operations to file in XFS file systems.
    ```
  - Obtain the current performance value of the `xfs.read_bytes` metric:
    ```
    # pminfo --fetch xfs.read_bytes
    xfs.read_bytes
    value 4891346238
    ```
  - Obtain per-device XFS metrics with `pminfo`:
    ```
    # pminfo --fetch --oneline xfs.perdev.read xfs.perdev.write
    xfs.perdev.read [number of XFS file system read operations]
    inst [0 or "loop1"] value 0
    inst [0 or "loop2"] value 0
    xfs.perdev.write [number of XFS file system write operations]
    inst [0 or "loop1"] value 86
    inst [0 or "loop2"] value 0
    ```

Additional resources
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.

This procedure describes how to reset XFS metrics using the `pmstore` tool.

Prerequisites

- PCP is installed. For more information, see `Installing and enabling PCP`.

Procedure

1. Display the value of a metric:

   ```
   $ pminfo -f xfs.write
   xfs.write
   value 325262
   ```

2. Reset all the XFS metrics:

   ```
   # pmstore xfs.control.reset 1
   xfs.control.reset old value=0 new value=1
   ```

Verification steps

- View the information after resetting the metric:

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

Additional resources

- `pmstore(1)` and `pminfo(1)` man pages
- Tools distributed with PCP
- PCP metric groups for XFS

7.4. PCP METRIC GROUPS FOR XFS

The following table describes the available 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_btree.*</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>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></td>
</tr>
<tr>
<td>xfs.xstrat.*</td>
<td>Counts for the number of bytes of file data flushed out by the XFS flush deamon 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>
</tbody>
</table>
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.

Metrics regarding the operations of the XFS btree.

Configuration metrics which are used to reset the metric counters for the XFS stats. Control metrics are toggled by means of the pmstore tool.

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

The following table describes the available per-device PCP metric group for XFS.

#### Table 7.2. 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>xfs.perdev.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.perdev.alloc_btree.*</td>
<td></td>
</tr>
<tr>
<td>xfs.perdev.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.perdev.bmap_btree.*</td>
<td></td>
</tr>
<tr>
<td>xfs.perdev.dir_ops.*</td>
<td>Counters for directory operations of XFS file systems for creation, entry deletions, count of “getdent” operations.</td>
</tr>
<tr>
<td>xfs.perdev.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>xfs.perdev.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>Metric Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>xfs.perdev.log.*</td>
<td>Counters for the number of log buffer writes over XFS filesystems includes the number of blocks written to disk. Metrics also for the number of log flushes and pinning.</td>
</tr>
<tr>
<td>xfs.perdev.log_tail.*</td>
<td>Counts for the number of log buffer writes over XFS filesystems. Metrics also for the number of log flushes and pinning.</td>
</tr>
<tr>
<td>xfs.perdev.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.perdev.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.perdev.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.perdev.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.perdev.btree.*</td>
<td>Metrics regarding the operations of the XFS btree.</td>
</tr>
</tbody>
</table>
CHAPTER 8. SETTING UP GRAPHICAL REPRESENTATION OF PCP METRICS

Using a combination of `pcp`, `grafana`, `pcp redis`, `pcp bpftrace`, and `pcp vector` provides graphs, based on the live data or data collected by Performance Co-Pilot (PCP).

This section describes how to set up and access the graphical representation of PCP metrics.

8.1. SETTING UP PCP WITH PCP-ZEROCONF

This procedure describes how to set up PCP on a system with the `pcp-zeroconf` package. Once the `pcp-zeroconf` package is installed, the system records the default set of metrics into archived files.

**Procedure**

- Install the `pcp-zeroconf` package:

  ```bash
  # dnf install pcp-zeroconf
  ```

**Verification steps**

- Ensure that the `pmlogger` service is active, and starts archiving the metrics:

  ```bash
  # pcp | grep pmlogger
  pmlogger: primary logger: /var/log/pcp/pmlogger/localhost.localdomain/20200401.00.12
  ```

**Additional resources**

- `pmlogger` man page
- Monitoring performance with Performance Co-Pilot

8.2. SETTING UP A GRAFANA-SERVER

Grafana generates graphs that are accessible from a browser. The `grafana-server` is a back-end server for the Grafana dashboard. It listens, by default, on all interfaces, and provides web services accessed through the web browser. The `grafana-pcp` plugin interacts with the `pmproxy` protocol in the backend.

This procedure describes how to set up a `grafana-server`.

**Prerequisites**

- PCP is configured. For more information, see Setting up PCP with pcp-zeroconf.

**Procedure**

1. Install the following packages:

   ```bash
   # dnf install grafana grafana-pcp
   ```

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

3. Open the server’s firewall for network traffic to the Grafana service.
   # firewall-cmd --permanent --add-service=grafana
   success
   # firewall-cmd --reload
   success
```

**Verification steps**

- Ensure that the `grafana-server` is listening and responding to requests:
  ```bash
  # ss -ntlp | grep 3000
  LISTEN 0 128 *:3000 *:* users:(("grafana-server",pid=19522,fd=7))
  ```
- Ensure that the `grafana-pcp` plugin is installed:
  ```bash
  # grafana-cli plugins ls | grep performancecopilot-pcp-app
  performancecopilot-pcp-app @ 3.1.0
  ```

**Additional resources**

- `pmproxy(1)` and `grafana-server` man pages

### 8.3. ACCESSING THE GRAFANA WEB UI

This procedure describes how to access the Grafana web interface.

Using the Grafana web interface, you can:

- add PCP Redis, PCP bpftrace, and PCP Vector data sources
- create dashboard
- view an overview of any useful metrics
- create alerts in PCP Redis

**Prerequisites**

1. PCP is configured. For more information, see Setting up PCP with pcp-zeroconf.
2. The `grafana-server` is configured. For more information, see Setting up a grafana-server.

**Procedure**

1. On the client system, open a browser and access the `grafana-server` on port **3000**, using `http://192.0.2.0:3000` link.
   Replace `192.0.2.0` with your machine IP.
2. For the first login, enter **admin** in both the **Email or username** and **Password** field. Grafana prompts to set a **New password** to create a secured account. If you want to set it later, click **Skip**.

3. From the menu, hover over the **Configuration** icon and then click **Plugins**.

4. In the **Plugins** tab, type performance co-pilot in the **Search by name or type** text box and then click **Performance Co-Pilot (PCP)** plugin.

5. In the **Plugins / Performance Co-Pilot** pane, click **Enable**.

6. Click Grafana **icon**. The Grafana **Home** page is displayed.

**Figure 8.1. Home Dashboard**

![Home Dashboard](image)

**NOTE**

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

7. In the Grafana **Home** page, click **Add your first data source** to add PCP Redis, PCP bpftrace, and PCP Vector data sources. For more information on adding data source, see:

- To add pcp redis data source, view default dashboard, create a panel, and an alert rule, see [Creating panels and alert in PCP Redis data source](#).

- To add pcp bpftrace data source and view the default dashboard, see [Viewing the PCP bpftrace System Analysis dashboard](#).

- To add pcp vector data source, view the default dashboard, and to view the vector checklist, see [Viewing the PCP Vector Checklist](#).

8. Optional: From the menu, hover over the **admin profile** **icon** to change the **Preferences** including **Edit Profile, Change Password**, or to **Sign out**.
8.4. CONFIGURING PCP REDIS

This section provides information for configuring PCP Redis data source.

Use the PCP Redis data source to:

- View data archives
- Query time series using pmseries language
- Analyze data across multiple hosts

Prerequisites

1. PCP is configured. For more information, see Setting up PCP with pcp-zeroconf.
2. The grafana-server is configured. For more information, see Setting up a grafana-server.

Procedure

1. Install the redis package:
   
   ```
   # dnf install redis
   ```

2. Start and enable the following services:

   ```
   # systemctl start pmproxy redis
   # systemctl enable pmproxy redis
   ```

3. Mail transfer agent, for example, sendmail or postfix is installed and configured.

4. Ensure that the allow_loading_unsigned_plugins parameter is set to PCP Redis database in the grafana.ini file:

   ```
   # vi /etc/grafana/grafana.ini
   
   allow_loading_unsigned_plugins = pcp-redis-datasource
   ```

5. Restart the grafana-server.

   ```
   # systemctl restart grafana-server
   ```

Verification steps

- Ensure that the pmproxy and redis are working:

  ```
  # pmseries disk.dev.read
  2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
  ```
This command does not return any data if the redis package is not installed.

Additional resources

- pmseries(1) man page

8.5. CREATING PANELS AND ALERT IN PCP REDIS DATA SOURCE

After adding the PCP Redis data source, you can view the dashboard with an overview of useful metrics, add a query to visualize the load graph, and create alerts that help you to view the system issues after they occur.

Prerequisites

1. The PCP Redis is configured. For more information, see Configuring PCP Redis.
2. The grafana-server is accessible. For more information, see Accessing the Grafana web UI.

Procedure

1. Log into the Grafana web UI.
2. In the Grafana Home page, click Add your first data source.
3. In the Add data source pane, type redis in the Filter by name or type text box and then click PCP Redis.
4. In the Data Sources / PCP Redis pane, perform the following:
   a. Add http://localhost:44322 in the URL field and then click Save & Test.
   b. Click Dashboards tab → Import → PCP Redis: Host Overview to see a dashboard with an overview of any useful metrics.

Figure 8.2. PCP Redis: Host Overview

5. Add a new panel:
a. From the menu, hover over the Create icon → Dashboard → Add new panel.

b. In the Query tab, select the PCP Redis from the query list instead of the selected default option and in the text field of A, enter metric, for example, kernel.all.load to visualize the kernel load graph.

c. Optional: Add Panel title and Description, and update other options from the Settings.

d. Click Save to apply changes and save the dashboard. Add Dashboard name.

e. Click Apply to apply changes and go back to the dashboard.

Figure 8.3. PCP Redis query panel

6. Create an alert rule:

a. In the PCP Redis query panel click Alert and then click Create Alert.

b. Edit the Name, Evaluate query, and For fields from the Rule, and specify the Conditions for your alert.

c. Click Save to apply changes and save the dashboard. Click Apply to apply changes and go back to the dashboard.
d. Optional: In the same panel, scroll down and click **Delete** icon to delete the created rule.

```
```
e. Optional: From the menu, click **Alerting** icon to view the created alert rules with different alert statuses, to edit the alert rule, or to pause the existing rule from the **Alert Rules** tab.

To add a notification channel for the created alert rule to receive an alert notification from Grafana, see **Adding notification channels for alerts**.

### 8.6. ADDING NOTIFICATION CHANNELS FOR ALERTS

By adding notification channels, you can receive an alert notification from Grafana whenever the alert rule conditions are met and the system needs further monitoring.

You can receive these alerts after selecting any one type from the supported list of notifiers, which includes DingDing, Discord, Email, Google Hangouts Chat, HipChat, Kafka REST Proxy, LINE, Microsoft Teams, OpsGenie, PagerDuty, Prometheus Alertmanager, Pushover, Sensu, Slack, Telegram, Threema Gateway, VictorOps, and webhook.

**Prerequisites**

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

2. An alert rule is created. For more information, see **Creating panels and alert in PCP Redis data source**.

3. Configure SMTP and add a valid sender’s email address in the **grafana/grafana.ini** file:

```bash
# vi /etc/grafana/grafana.ini

[smtph]
enabled = true
from_address = abc@gmail.com
```

Replace `abc@gmail.com` by a valid email address.

**Procedure**
1. From the menu, hover over the Alerting icon → click Notification channels → Add channel.

2. In the Add notification channel details pane, perform the following:
   
   a. Enter your name in the Name text box
   
   b. Select the communication Type, for example, Email and enter the email address. You can add multiple email addresses using the ; separator.
   
   c. Optional: Configure Optional Email settings and Notification settings.

3. Click Save.

4. Select a notification channel in the alert rule:
   
   a. From the menu, hover over the Alerting icon and then click Alert rules.
   
   b. From the Alert Rules tab, click the created alert rule.
   
   c. On the Notifications tab, select your notification channel name from the Send to option, and then add an alert message.
   
   d. Click Apply.

Additional resources

- Upstream Grafana documentation for alert notifications

8.7. SETTING UP AUTHENTICATION BETWEEN PCP COMPONENTS

You can setup authentication using the scram-sha-256 authentication mechanism, which is supported by PCP through the Simple Authentication Security Layer (SASL) framework.

Procedure

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

   # dnf install cyrus-sasl-scram cyrus-sasl-lib

2. Specify the supported authentication mechanism and the user database path in the pmcd.conf file:

   # vi /etc/sasl2/pmcd.conf

   mech_list: scram-sha-256

   sasldb_path: /etc/pcp/passwd.db

3. Create a new user:

   # useradd -r metrics
Replace *metrics* by your user name.

4. Add the created user in the user database:

```bash
# saslpasswd2 -a pmcd metrics
Password:
Again (for verification):
```

To add the created user, you are required to enter the *metrics* account password.

5. Set the permissions of the user database:

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

6. Restart the *pmcd* service:

```bash
# systemctl restart pmcd
```

**Verification steps**

- Verify the *sasl* configuration:

```bash
# pminfo -f -h "pcp://127.0.0.1?username=metrics" disk.dev.read
Password:
disk.dev.read
inst [0 or "sda"] value 19540
```

**Additional resources**

- *saslauthd(8)*, *pminfo(1)*, and *sha256* man pages
- How can I setup authentication between PCP components, like PMDAs and pmcd in RHEL 8.2?

### 8.8. INSTALLING PCP BPFTRACE

Install the PCP *bpftrace* agent to introspect a system and to gather metrics from the kernel and user-space tracepoints.

The *bpftrace* agent uses bpftrace scripts to gather the metrics. The *bpftrace* scripts use the enhanced Berkeley Packet Filter (*eBPF*).

This procedure describes how to install a *pcp bpftrace*.

**Prerequisites**

1. PCP is configured. For more information, see *Setting up PCP with pcp-zeroconf*.

2. The *grafana-server* is configured. For more information, see *Setting up a grafana-server*.

3. The *scram-sha-256* authentication mechanism is configured. For more information, see *Setting up authentication between PCP components*. 

88
Procedure

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

   ```bash
   # dnf install pcp-pmda-bpftrace
   ```

2. Edit the `bpftrace.conf` file and add the user that you have created in the `{setting-up-authentication-between-pcp-components}`:

   ```bash
   # 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 `bpftrace` PMDA:

   ```bash
   # 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. For more information, see Viewing the PCP bpftrace System Analysis dashboard.
   ```

Additional resources

- `pmdabpftrace(1)` and `bpftrace` man pages

8.9. VIEWING THE PCP BPFTRACE SYSTEM ANALYSIS DASHBOARD

Using the PCP bpftrace data source, you can access the live data from sources which are not available as normal data from the `pmlogger` or archives.

In the PCP bpftrace data source, you can view the dashboard with an overview of useful metrics.

Prerequisites

1. The PCP bpftrace is installed. For more information, see Installing PCP bpftrace.

2. The `grafana-server` is accessible. For more information, see Accessing the Grafana web UI.

Procedure

1. Log into the Grafana web UI.

2. In the Grafana Home page, click Add your first data source
3. In the Add data source pane, type bpftrace in the Filter by name or type text box and then click PCP bpftrace.

4. In the Data Sources / PCP bpftrace pane, perform the following:
   b. Toggle the Basic Auth option and add the created user credentials in the User and Password field.
   c. Click Save & Test.

   Figure 8.5. Adding PCP bpftrace in the data source

   d. Click Dashboards tab → Import → PCP bpftrace: System Analysis to see a dashboard with an overview of any useful metrics.

   Figure 8.6. PCP bpftrace: System Analysis

8.10. INSTALLING PCP VECTOR

This procedure describes how to install a pcp vector.
Prerequisites

1. PCP is configured. For more information, see Setting up PCP with pcp-zeroconf.

2. The grafana-server is configured. For more information, see Setting up a grafana-server.

Procedure

1. Install the pcp-pmda-bcc package:

```
# dnf install pcp-pmda-bcc
```

2. Install the bcc PMDA:

```
# cd /var/lib/pcp/pmdas/bcc
# ./Install

[Wed Apr  1 00:27:48] pmdabcc(22341) Info: Initializing, currently in 'notready' state.
[Wed Apr  1 00:27:48] pmdabcc(22341) Info: Enabled modules:
[Wed Apr  1 00:27:48] pmdabcc(22341) Info: ['biolatency', 'sysfork','...

Updating the Performance Metrics Name Space (PMNS) ...
Terminate PMDA if already installed ...
Updating the PMCD control file, and notifying PMCD ...
Check bcc metrics have appeared ... 1 warnings, 1 metrics and 0 values
```

Additional resources

- pmdabcc(1) man page

8.11. VIEWING THE PCP VECTOR CHECKLIST

The PCP Vector data source displays live metrics and uses the pcp metrics. It analyzes data for individual hosts.

After adding the PCP Vector data source, you can view the dashboard with an overview of useful metrics and view the related troubleshooting or reference links in the checklist.

Prerequisites

1. The PCP Vector is installed. For more information, see Installing PCP Vector.

2. The grafana-server is accessible. For more information, see Accessing the Grafana web UI.

Procedure

1. Log into the Grafana web UI.

2. In the Grafana Home page, click Add your first data source

3. In the Add data source pane, type vector in the Filter by name or type text box and then click PCP Vector.

4. In the Data Sources / PCP Vector pane, perform the following:
   a. Add http://localhost:44322 in the URL field and then click Save & Test.
b. Click Dashboards tab → Import → PCP Vector: Host Overview to see a dashboard with an overview of any useful metrics.

Figure 8.7. PCP Vector: Host Overview

5. From the menu, hover over the Performance Co-Pilot plugin and then click PCP Vector Checklist. In the PCP checklist, click help or warning icon to view the related troubleshooting or reference links.

Figure 8.8. Performance Co-Pilot / PCP Vector Checklist

8.12. TROUBLESHOOTING GRAFANA ISSUES

This section describes how to troubleshoot Grafana issues, such as, Grafana does not display any data, the dashboard is black, or similar issues.

Procedure

- Verify that the pmlogger service is up and running by executing the following command:
Verify if files were created or modified to the disk by executing the following command:

```
$ ls /var/log/pcp/pmlogger/$(hostname)/ -rlt
```

```
total 4024
-rw-r--r--. 1 pcp pcp   45996 Oct 13  2019 20191013.20.07.meta.xz
-rw-r--r--. 1 pcp pcp     412 Oct 13  2019 20191013.20.07.index
-rw-r--r--. 1 pcp pcp   32188 Oct 13  2019 20191013.20.07.0.xz
-rw-r--r--. 1 pcp pcp   44756 Oct 13  2019 20191013.20.30-00.meta.xz
[...]
```

Verify that the `pmproxy` service is running by executing the following command:

```
$ systemctl status pmproxy
```

Verify that `pmproxy` is running, time series support is enabled, and a connection to Redis is established by viewing the `/var/log/pcp/pmproxy/pmproxy.log` file and ensure that it contains the following text:

```
pmproxy(1716) Info: Redis slots, command keys, schema version setup

Here, **1716** is the PID of `pmproxy`, which will be different for every invocation of `pmproxy`.

Verify if the Redis database contains any keys by executing the following command:

```
$ redis-cli dbsize
(integer) 34837
```

Verify if any PCP metrics are in the Redis database and `pmproxy` is able to access them by executing the following commands:

```
$ pmseries disk.dev.read
2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df

$ pmseries "disk.dev.read[count:10]"
2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
70e83e88d4e1857a3a31605c6d1333755f2dd17c
[Mon Jul 26 12:21:00.087401000 2021] 117758
70e83e88d4e1857a3a31605c6d1333755f2dd17c
70e83e88d4e1857a3a31605c6d1333755f2dd17c
[...]
```

```
$ redis-cli --scan --pattern "*$(pmseries 'disk.dev.read')"*
pcp:metric.name:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:values:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:desc:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:labelvalue:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:instances:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
pcp:labelflags:series:2eb3e58d8f1e231361fb15cf1aa26fe534b4d9df
```

• Verify if there are any errors in the Grafana logs by executing the following command:

```
$ journalctl -e -u grafana-server
-- Logs begin at Mon 2021-07-26 11:55:10 IST, end at Mon 2021-07-26 12:30:15 IST. --
Jul 26 11:55:17 localhost.localdomain systemd[1]: Starting Grafana instance...
Jul 26 11:55:17 localhost.localdomain grafana-server[1171]: t=2021-07-26T11:55:17+0530 lvl=info msg="Config loaded from" logger=settings file=/etc/g
[...]
```
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 Setting the disk scheduler using TuneD
- Set the scheduler using **udev**, as described in Setting the disk scheduler using udev rules
- Temporarily change the scheduler on a running system, as described in Temporarily setting a scheduler for a specific disk

**NOTE**

In Red Hat Enterprise Linux 9, block devices support only multi-queue scheduling. This enables the block layer performance to scale well with fast solid-state drives (SSDs) and multi-core systems.

The traditional, single-queue schedulers, which were available in Red Hat Enterprise Linux 7 and earlier versions, have been removed.

9.1. AVAILABLE DISK SCHEDULERS

The following multi-queue disk schedulers are supported in Red Hat Enterprise Linux 9:

**none**

Implements a first-in first-out (FIFO) scheduling algorithm. It merges requests at the generic block layer through a simple last-hit cache.

**mq-deadline**

Attempts to provide a guaranteed latency for requests from the point at which requests reach the scheduler.

The **mq-deadline** scheduler sorts queued I/O requests into a read or write batch and then schedules them for execution in increasing logical block addressing (LBA) order. By default, read batches take precedence over write batches, because applications are more likely to block on read I/O operations. After **mq-deadline** processes a batch, it checks how long write operations have been starved of processor time and schedules the next read or write batch as appropriate.

This scheduler is suitable for most use cases, but particularly those in which the write operations are mostly asynchronous.

**bfq**

Targets desktop systems and interactive tasks.

The **bfq** scheduler ensures that a single application is never using all of the bandwidth. In effect, the storage device is always as responsive as if it was idle. In its default configuration, **bfq** focuses on delivering the lowest latency rather than achieving the maximum throughput.

**bfq** is based on **cfq** code. It does not grant the disk to each process for a fixed time slice but assigns a **budget** measured in number of sectors to the process.

This scheduler is suitable while copying large files and the system does not become unresponsive in this case.
kyber

The scheduler tunes itself to achieve a latency goal by calculating the latencies of every I/O request submitted to the block I/O layer. You can configure the target latencies for read, in the case of cache-misses, and synchronous write requests. This scheduler is suitable for fast devices, for example NVMe, SSD, or other low latency devices.

9.2. DIFFERENT DISK SCHEDULERS FOR DIFFERENT USE CASES

Depending on the task that your system performs, the following disk schedulers are recommended as a baseline prior to any analysis and tuning tasks:

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

9.3. THE DEFAULT DISK SCHEDULER

Block devices use the default disk scheduler unless you specify another scheduler.

**NOTE**

For *non-volatile Memory Express (NVMe)* block devices specifically, the default scheduler is `none` and Red Hat recommends not changing this.

The kernel selects a default disk scheduler based on the type of device. The automatically selected scheduler is typically the optimal setting. If you require a different scheduler, Red Hat recommends to use `udev` rules or the `TuneD` application to configure it. Match the selected devices and switch the scheduler only for those devices.

9.4. DETERMINING THE ACTIVE DISK SCHEDULER

This procedure determines which disk scheduler is currently active on a given block device.

**Procedure**

- Read the content of the `/sys/block/device/queue/scheduler` file:

  ```
  # cat /sys/block/device/queue/scheduler
  [mq-deadline] kyber bfq none
  ```
In the file name, replace device with the block device name, for example \texttt{sdc}.

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

**9.5. SETTING THE DISK SCHEDULER USING TUNED**

This procedure creates and enables a \texttt{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:

- \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}

**Prerequisites**

- The \texttt{tuned} service is installed and enabled. For details, see \texttt{Installing and enabling TuneD}.

**Procedure**

1. Optional: Select an existing \texttt{TuneD} profile on which your profile will be based. For a list of available profiles, see \texttt{TuneD profiles distributed with RHEL}.

   To see which profile is currently active, use:
   
   \[
   $ \texttt{tuned-adm active}
   \]

2. Create a new directory to hold your \texttt{TuneD} profile:
   
   \[
   # \texttt{mkdir /etc/tuned/my-profile}
   \]

3. Find the system unique identifier of the selected block device:
   
   \[
   $ \texttt{udevadm info --query=property --name=/dev/device | grep -E '(WWN|SERIAL)'}
   \]

   \[
   \begin{align*}
   \text{ID	extunderscore WWN} &= \text{0x5002538d00000000} \\
   \text{ID	extunderscore SERIAL} &= \text{Generic\textunderscore SD\textunderscore MMC\textunderscore 201205010309000000\textunderscore 0\textunderscore 0} \\
   \text{ID	extunderscore SERIAL\textunderscore SHORT} &= \text{20120501030900000}
   \end{align*}
   \]

   **NOTE**

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

4. Create the \texttt{/etc/tuned/my-profile/tuned.conf} configuration file. In the file, set the following options:
   
   a. Optional: Include an existing profile:
b. Set the selected disk scheduler for the device that matches the WWN identifier:

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

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

Here:

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

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

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

5. Enable your profile:

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

**Verification steps**

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

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

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

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

```
# cat /sys/block/device/queue/scheduler
[mq-deadline] kyber bfq none
```

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

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

**Additional resources**

- [Customizing TuneD profiles](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/system_usage_and_performance#customizing-tuned-profiles)
9.6. Setting the Disk Scheduler Using Udev Rules

This procedure sets a given disk scheduler for specific block devices using udev rules. The setting persists across system reboots.

In the following commands and configuration, replace:

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

**Procedure**

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

   ```bash
   $ udevadm info --name=/dev/device | grep -E '(WWN|SERIAL)'
   E: ID_WWN=0x5002538d00000000
   E: ID_SERIAL=Generic-_SD_MMC_20120501030900000-0:0
   E: ID_SERIAL_SHORT=20120501030900000
   ```

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

2. Configure the udev rule. Create the `/etc/udev/rules.d/99-scheduler.rules` file with the following content:

   ```bash
   ACTION=='add|change', SUBSYSTEM=='block', ENV{IDNAME}=='device system unique id', ATTR{queue/scheduler}=='selected-scheduler'
   ```

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

3. Reload udev rules:

   ```bash
   # udevadm control --reload-rules
   ```

4. Apply the scheduler configuration:

   ```bash
   # udevadm trigger --type=devices --action=change
   ```

**Verification steps**

- Verify the active scheduler:
9.7. TEMPORARILY SETTING A SCHEDULER FOR A SPECIFIC DISK

This procedure sets a given disk scheduler for specific block devices. The setting does not persist across system reboots.

Procedure

- Write the name of the selected scheduler to the `/sys/block/device/queue/scheduler` file:

```
# echo selected-scheduler > /sys/block/device/queue/scheduler
```

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

Verification steps

- Verify that the scheduler is active on the device:

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

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 supports the latest protocol version. If Samba also uses the latest protocol version, Windows clients connecting to Samba benefit from the performance improvements. In Samba, the default value of the server max protocol is set to the latest supported stable SMB protocol version.

NOTE

To always have the latest stable SMB protocol version enabled, do not set the server max protocol parameter. If you set the parameter manually, you will need to modify the setting with each new version of the SMB protocol, to have the latest protocol version enabled.

The following procedure explains how to use the default value in the server max protocol parameter.

Procedure

1. Remove the server max protocol parameter from the [global] section in the /etc/samba/smb.conf file.

2. Reload the Samba configuration

# smbcontrol all reload-config

10.2. TUNING SHARES WITH DIRECTORIES THAT CONTAIN A LARGE NUMBER OF FILES

Linux supports case-sensitive file names. For this reason, Samba needs to scan directories for uppercase and lowercase file names when searching or accessing a file. You can configure a share to create new files only in lowercase or uppercase, which improves the performance.

Prerequisites

- Samba is configured as a file server

Procedure
1. Rename all files on the share to lowercase.

   **NOTE**
   
   Using the settings in this procedure, files with names other than in lowercase will no longer be displayed.

2. Set the following parameters in the share’s section:

   ```plaintext
   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:

   ```plaintext
   # testparm
   ```

4. Reload the Samba configuration:

   ```plaintext
   # 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. MANAGING POWER CONSUMPTION WITH POWERTOP

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

11.1. THE PURPOSE OF POWERTOP

PowerTOP is a program that diagnoses issues related to power consumption and provides suggestions on how to extend battery lifetime.

The PowerTOP tool can provide an estimate of the total power usage of the system and also individual power usage for each process, device, kernel worker, timer, and interrupt handler. The tool can also identify specific components of kernel and user-space applications that frequently wake up the CPU.

Red Hat Enterprise Linux 9 uses version 2.x of PowerTOP.

11.2. USING POWERTOP

Prerequisites

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

```
# dnf install powertop
```

11.2.1. Starting PowerTOP

Procedure

To run PowerTOP, use the following command:

```
# powertop
```

**IMPORTANT**

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

11.2.2. 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.

### 11.2.3. 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
  ```

### 11.2.4. Additional resources

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

### 11.3. **POWERTOP** STATISTICS

While it runs, **PowerTOP** gathers statistics from the system.

**PowerTOP**'s output provides multiple tabs:

- Overview
- Idle stats
- Frequency stats
- Device stats
- Tunables
- WakeUp

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

#### 11.3.1. The Overview tab

In the **Overview** tab, you can view a list of the components that either send wakeups to the CPU most frequently or consume the most power. The items within the **Overview** tab, including processes, interrupts, devices, and other resources, are sorted according to their utilization.

The adjacent columns within the **Overview** tab provide the following pieces of information:

- Usage
Power estimation of how the resource is being used.

**Events/s**
- Wakeups per second. The number of wakeups per second indicates how efficiently the services or the devices and drivers of the kernel are performing. Less wakeups means that less power is consumed. Components are ordered by how much further their power usage can be optimized.

**Category**
- Classification of the component; such as process, device, or timer.

**Description**
- Description of the component.

If properly calibrated, a power consumption estimation for every listed item in the first column is shown as well.

Apart from this, the **Overview** tab includes the line with summary statistics such as:

- Total power consumption
- Remaining battery life (only if applicable)
- Summary of total wakeups per second, GPU operations per second, and virtual file system operations per second

### 11.3.2. 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.

### 11.3.3. The Device stats tab

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

### 11.3.4. 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.

### 11.3.5. The WakeUp tab

The **WakeUp** tab displays the device wakeup settings available for users to change as and when required.

Use the **up** and **down** keys to move through the available settings, and the **enter** key to enable or disable a setting.
11.4. WHY POWERTOP DOES NOT DISPLAY FREQUENCY STATS VALUES IN SOME INSTANCES

While using the Intel P-State driver, PowerTOP only displays values in the Frequency Stats tab if the driver is in passive mode. But, even in this case, the values may be incomplete.

In total, there are three possible modes of the Intel P-State driver:

- Active mode with Hardware P-States (HWP)
- Active mode without HWP
- Passive mode

Switching to the ACPI CPUfreq driver results in complete information being displayed by PowerTOP. However, it is recommended to keep your system on the default settings.

To see what driver is loaded and in what mode, run:

```
# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_driver
```

- `intel_pstate` is returned if the Intel P-State driver is loaded and in active mode.
- `intel_cpufreq` is returned if the Intel P-State driver is loaded and in passive mode.
- `acpi-cpufreq` is returned if the ACPI CPUfreq driver is loaded.

While using the Intel P-State driver, add the following argument to the kernel boot command line to force the driver to run in passive mode:

```
inTEL_pstate=passive
```
To disable the Intel P-State driver and use, instead, the ACPI CPUfreq driver, add the following argument to the kernel boot command line:

```
intel_pstate=disable
```

### 11.5. Generating an HTML Output

Apart from the `powertop`'s output in terminal, you can also generate an HTML report.

**Procedure**

- Run the `powertop` command with the `--html` option:

  ```
  # powertop --html=htmlfile.html
  ```

  Replace the `htmlfile.html` parameter with the required name for the output file.

### 11.6. Optimizing Power Consumption

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

#### 11.6.1. Optimizing power consumption using the powertop service

You can use the `powertop` service to automatically enable all `PowerTOP`'s suggestions from the Tunables tab on the boot:

**Procedure**

- Enable the `powertop` service:

  ```
  # systemctl enable powertop
  ```

#### 11.6.2. The powertop2tuned utility

The `powertop2tuned` utility allows you to create custom `TuneD` profiles from `PowerTOP` suggestions.

By default, `powertop2tuned` creates profiles in the `/etc/tuned/` directory, and bases the custom profile on the currently selected `TuneD` profile. For safety reasons, all `PowerTOP` tunings are initially disabled in the new profile.

To enable the tunings, you can:

- Uncomment them in the `/etc/tuned/profile_name/tuned.conf` file.

- Use the `--enable` or `-e` option to generate a new profile that enables most of the tunings suggested by `PowerTOP`.
  Certain potentially problematic tunings, such as the USB autosuspend, are disabled by default and need to be uncommented manually.

#### 11.6.3. Optimizing power consumption using the powertop2tuned utility
Prerequisites

- The `powertop2tuned` utility is installed on the system:
  
  ```bash
  # dnf install tuned-utils
  ```

Procedure

1. Create a custom profile:
   
   ```bash
   # powertop2tuned new_profile_name
   ```

2. Activate the new profile:
   
   ```bash
   # tuned-adm profile new_profile_name
   ```

Additional information

- For a complete list of options that `powertop2tuned` supports, use:
  
  ```bash
  $ powertop2tuned --help
  ```

11.6.4. Comparison of `powertop.service` and `powertop2tuned`

Optimizing power consumption with `powertop2tuned` is preferred over `powertop.service` for the following reasons:

- The `powertop2tuned` utility represents integration of PowerTOP into TuneD, which enables to benefit of advantages of both tools.

- The `powertop2tuned` utility allows for fine-grained control of enabled tuning.

- With `powertop2tuned`, potentially dangerous tuning are not automatically enabled.

- With `powertop2tuned`, rollback is possible without reboot.
CHAPTER 12. GETTING STARTED WITH PERF

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

12.1. 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.

12.2. INSTALLING PERF

This procedure installs the perf user-space tool.

Procedure

- Install the perf tool:

```bash
# dnf install perf
```

12.3. 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**

- Add the `--help` option to a subcommand to open the man page.
CHAPTER 13. CONFIGURING AN OPERATING SYSTEM TO OPTIMIZE CPU UTILIZATION

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

13.1. TOOLS FOR MONITORING AND DIAGNOSING PROCESSOR ISSUES

The following are the tools available in Red Hat Enterprise Linux 9 to monitor and diagnose processor-related performance issues:

- **turbostat** tool prints counter results at specified intervals to help administrators identify unexpected behavior in servers, such as excessive power usage, failure to enter deep sleep states, or system management interrupts (SMIs) being created unnecessarily.

- **numactl** utility provides a number of options to manage processor and memory affinity. The **numactl** package includes the **libnuma** library which offers a simple programming interface to the NUMA policy supported by the kernel, and can be used for more fine-grained tuning than the **numactl** application.

- **numastat** tool displays per-NUMA node memory statistics for the operating system and its processes, and shows administrators whether the process memory is spread throughout a system or is centralized on specific nodes. This tool is provided by the **numactl** package.

- **numad** is an automatic NUMA affinity management daemon. It monitors NUMA topology and resource usage within a system in order to dynamically improve NUMA resource allocation and management.

- **/proc/interrupts** file displays the interrupt request (IRQ) number, the number of similar interrupt requests handled by each processor in the system, the type of interrupt sent, and a comma-separated list of devices that respond to the listed interrupt request.

- **pqos** utility is available in the **intel-cmt-cat** package. It monitors CPU cache and memory bandwidth on recent Intel processors. It monitors:
  - The instructions per cycle (IPC).
  - The count of last level cache MISSES.
  - The size in kilobytes that the program executing in a given CPU occupies in the LLC.
  - The bandwidth to local memory (MBL).
  - The bandwidth to remote memory (MBR).

- **x86_energy_perf_policy** tool allows administrators to define the relative importance of performance and energy efficiency. This information can then be used to influence processors that support this feature when they select options that trade off between performance and energy efficiency.

- **taskset** tool is provided by the **util-linux** package. It allows administrators to retrieve and set the processor affinity of a running process, or launch a process with a specified processor affinity.
Additional resources

- turbostat(8), numactl(8), numastat(8), numa(7), numad(8), pqos(8),
  x86_energy_perf_policy(8), and taskset(1) man pages

13.2. TYPES OF SYSTEM TOPOLOGY

In modern computing, the idea of a CPU is a misleading one, as most modern systems have multiple processors. The topology of the system is the way these processors are connected to each other and to other system resources. This can affect system and application performance, and the tuning considerations for a system.

The following are the two primary types of topology used in modern computing:

**Symmetric Multi-Processor (SMP) topology**

SMP topology allows all processors to access memory in the same amount of time. However, because shared and equal memory access inherently forces serialized memory accesses from all the CPUs, SMP system scaling constraints are now generally viewed as unacceptable. For this reason, practically all modern server systems are NUMA machines.

**Non-Uniform Memory Access (NUMA) topology**

NUMA topology was developed more recently than SMP topology. In a NUMA system, multiple processors are physically grouped on a socket. Each socket has a dedicated area of memory and processors that have local access to that memory, these are referred to collectively as a node. Processors on the same node have high speed access to that node’s memory bank, and slower access to memory banks not on their node. Therefore, there is a performance penalty when accessing non-local memory. Thus, performance sensitive applications on a system with NUMA topology should access memory that is on the same node as the processor executing the application, and should avoid accessing remote memory wherever possible.

Multi-threaded applications that are sensitive to performance may benefit from being configured to execute on a specific NUMA node rather than a specific processor. Whether this is suitable depends on your system and the requirements of your application. If multiple application threads access the same cached data, then configuring those threads to execute on the same processor may be suitable. However, if multiple threads that access and cache different data execute on the same processor, each thread may evict cached data accessed by a previous thread. This means that each thread ‘misses’ the cache and wastes execution time fetching data from memory and replacing it in the cache. Use the `perf` tool to check for an excessive number of cache misses.

13.2.1. Displaying system topologies

There are a number of commands that help understand the topology of a system. This procedure describes how to determine the system topology.

Procedure

- To display an overview of your system topology:

```bash
$ numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 4 8 12 16 20 24 28 32 36
node 0 size: 65415 MB
node 0 free: 43971 MB
[...]
```
To gather the information about the CPU architecture, such as the number of CPUs, threads, cores, sockets, and NUMA nodes:

```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                40
On-line CPU(s) list:   0-39
Thread(s) per core:    1
Core(s) per socket:    10
Socket(s):             4
NUMA node(s):          4
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 47
Model name:            Intel(R) Xeon(R) CPU E7- 4870  @ 2.40GHz
Stepping:              2
CPU MHz:               2394.204
BogoMIPS:              4787.85
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              30720K
NUMA node0 CPU(s):     0,4,8,12,16,20,24,28,32,36
NUMA node1 CPU(s):     2,6,10,14,18,22,26,30,34,38
NUMA node2 CPU(s):     1,5,9,13,17,21,25,29,33,37
NUMA node3 CPU(s):     3,7,11,15,19,23,27,31,35,39
```

To view a graphical representation of your system:

```
# dnf install hwloc-gui
# lstopo
```
To view the detailed textual output:

```
# dnf install hwloc
# lstopo-no-graphics
Machine (15GB)
  Package L#0 + L3 L#0 (8192KB)
    L2 L#0 (256KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0
      PU L#0 (P#0)
      PU L#1 (P#4)
    HostBridge L#0
  PCI 8086:5917
    GPU L#0 "renderD128"
    GPU L#1 "controlD64"
    GPU L#2 "card0"
  PCIBridge
    PCI 8086:24fd
      Net L#3 "wlp61s0"
  PCIBridge
    PCI 8086:f1a6
    PCI 8086:15d7
      Net L#4 "enp0s31f6"
```

Additional resources

- `numactl(8)`, `lscpu(1)`, and `lstopo(1)` man pages

### 13.3. CONFIGURING KERNEL TICK TIME
By default, Red Hat Enterprise Linux 9 uses a tickless kernel, which does not interrupt idle CPUs in order to reduce power usage and allow new processors to take advantage of deep sleep states.

Red Hat Enterprise Linux 9 also offers a dynamic tickless option, which is useful for latency-sensitive workloads, such as high performance computing or real-time 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:

   ```shell
   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:

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

   On systems with UEFI firmware, execute the following command:

   ```shell
   # 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:

   ```shell
   # 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:

   ```shell
   echo 1 > /sys/bus/workqueue/devices/writeback/cpumask
   ```

**Verification steps**

- Once the system is rebooted, verify if dynticks are enabled:

  ```shell
  # journalctl -xe | grep dynticks
  ```

- Verify that the dynamic tickless configuration is working correctly:

  ```shell
  # perf stat -C 1 -e irq_vectors:local_timer_entry taskset -c 1 sleep 3
  ```
This command measures ticks on CPU 1 while telling CPU 1 to sleep for 3 seconds.

- The default kernel timer configuration shows around 3100 ticks on a regular CPU:

  ```
  # perf stat -C 0 -e irq_vectors:local_timer_entry taskset -c 0 sleep 3
  Performance counter stats for 'CPU(s) 0':
  3,107  irq_vectors:local_timer_entry
  3.001342790 seconds time elapsed
  ```

- With the dynamic tickless kernel configured, you should see around 4 ticks instead:

  ```
  # perf stat -C 1 -e irq_vectors:local_timer_entry taskset -c 1 sleep 3
  Performance counter stats for 'CPU(s) 1':
  4  irq_vectors:local_timer_entry
  3.001544078 seconds time elapsed
  ```

Additional resources

- perf(1) and cpuset(7) man pages
- All about nohz_full kernel parameter Red Hat Knowledgebase article
- How to verify the list of "isolated" and "nohz_full" CPU information from sysfs? Red Hat Knowledgebase article

13.4. OVERVIEW OF AN INTERRUPT REQUEST

An interrupt request or IRQ is a signal for immediate attention sent from a piece of hardware to a processor. Each device in a system is assigned one or more IRQ numbers which allow it to send unique interrupts. When interrupts are enabled, a processor that receives an interrupt request immediately pauses execution of the current application thread in order to address the interrupt request.

Because interrupt halts normal operation, high interrupt rates can severely degrade system performance. It is possible to reduce the amount of time taken by interrupts by configuring interrupt affinity or by sending a number of lower priority interrupts in a batch (coalescing a number of interrupts).

Interrupt requests have an associated affinity property, `smp_affinity`, which defines the processors that handle the interrupt request. To improve application performance, assign interrupt affinity and process affinity to the same processor, or processors on the same core. This allows the specified interrupt and application threads to share cache lines.

On systems that support interrupt steering, modifying the `smp_affinity` property of an interrupt request sets up the hardware so that the decision to service an interrupt with a particular processor is made at the hardware level with no intervention from the kernel.

13.4.1. Balancing interrupts manually
If your BIOS exports its NUMA topology, the *irqbalance* service can automatically serve interrupt requests on the node that is local to the hardware requesting service.

**Procedure**

1. Check which devices correspond to the interrupt requests that you want to configure.

2. Find the hardware specification for your platform. Check if the chipset on your system supports distributing interrupts.
   a. If it does, you can configure interrupt delivery as described in the following steps. Additionally, check which algorithm your chipset uses to balance interrupts. Some BIOSes have options to configure interrupt delivery.
   b. If it does not, your chipset always routes all interrupts to a single, static CPU. You cannot configure which CPU is used.

3. Check which Advanced Programmable Interrupt Controller (APIC) mode is in use on your system:
   ```shell
   $ 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 Setting the *smp_affinity mask*.

**Additional resources**

- *journalctl* and *taskset* man pages

**13.4.2. Setting the *smp_affinity mask***

The *smp_affinity* value is stored as a hexadecimal bit mask representing all processors in the system. Each bit configures a different CPU. The least significant bit is CPU 0.

The default value of the mask is $f$, which means that an interrupt request can be handled on any processor in the system. Setting this value to 1 means that only processor 0 can handle the interrupt.

**Procedure**

1. In binary, use the value 1 for CPUs that handle the interrupts. For example, to set CPU 0 and CPU 7 to handle interrupts, use 0000000010000001 as the binary code:

**Table 13.1. Binary Bits for CPUs**
2. Convert the binary code to hexadecimal:
   For example, to convert the binary code using Python:

   ```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:

   ```bash
   # echo mask > /proc/irq/irq_number/smp_affinity
   ```

Additional resources

- `journalctl(1)`, `irqbalance(1)`, and `taskset(1)` man pages
CHAPTER 14. 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.

14.1. CATEGORIES OF SCHEDULING POLICIES

Performance sensitive applications often benefit from the designer or administrator determining where threads are run. The Linux scheduler implements a number of scheduling policies which determine where and for how long a thread runs.

The following are the two major categories of scheduling policies:

Normal policies

- Normal threads are used for tasks of normal priority.

Realtime policies

- Realtime policies are used for time-sensitive tasks that must complete without interruptions. Realtime threads are not subject to time slicing. This means the thread runs until they block, exit, voluntarily yield, or are preempted by a higher priority thread.
- The lowest priority realtime thread is scheduled before any thread with a normal policy. For more information, see Static priority scheduling with SCHED_FIFO and Round robin priority scheduling with SCHED_RR.

Additional resources

- sched(7), sched_setaffinity(2), sched_getaffinity(2), sched_setscheduler(2), and sched_getscheduler(2) man pages

14.2. STATIC PRIORITY SCHEDULING WITH SCHED_FIFO

The SCHED_FIFO, also called static priority scheduling, is a realtime policy that defines a fixed priority for each thread. This policy allows administrators to improve event response time and reduce latency. It is recommended to not execute this policy for an extended period of time for time sensitive tasks.

When SCHED_FIFO is in use, the scheduler scans the list of all the SCHED_FIFO threads in order of priority and schedules the highest priority thread that is ready to run. The priority level of a SCHED_FIFO thread can be any integer from 1 to 99, where 99 is treated as the highest priority. Red Hat recommends starting with a lower number and increasing priority only when you identify latency issues.
WARNING

Because realtime threads are not subject to time slicing, Red Hat does not recommend setting a priority as 99. This keeps your process at the same priority level as migration and watchdog threads; if your thread goes into a computational loop and these threads are blocked, they will not be able to run. Systems with a single processor will eventually hang in this situation.

Administrators can limit \texttt{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:

\texttt{/proc/sys/kernel/sched\_rt\_period\_us}

This parameter defines the time period, in microseconds, that is considered to be one hundred percent of the processor bandwidth. The default value is \texttt{1000000 µs}, or 1 second.

\texttt{/proc/sys/kernel/sched\_rt\_runtime\_us}

This parameter defines the time period, in microseconds, that is devoted to running real-time threads. The default value is \texttt{950000 µs}, or 0.95 seconds.

14.3. ROUND ROBIN PRIORITY SCHEDULING WITH SCHED\_RR

The \texttt{SCHED\_RR} is a round-robin variant of the \texttt{SCHED\_FIFO}. This policy is useful when multiple threads need to run at the same priority level.

Like \texttt{SCHED\_FIFO}, \texttt{SCHED\_RR} is a realtime policy that defines a fixed priority for each thread. The scheduler scans the list of all \texttt{SCHED\_RR} threads in order of priority and schedules the highest priority thread that is ready to run. However, unlike \texttt{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 \texttt{sched\_rr\_timeslice\_ms} kernel parameter in the \texttt{/proc/sys/kernel/sched\_rr\_timeslice\_ms} file. The lowest value is \texttt{1 millisecond}.

14.4. NORMAL SCHEDULING WITH SCHED\_OTHER

The \texttt{SCHED\_OTHER} is the default scheduling policy in Red Hat Enterprise Linux 9. This policy uses the Completely Fair Scheduler (CFS) to allow fair processor access to all threads scheduled with this policy. This policy is most useful when there are a large number of threads or when data throughput is a priority, as it allows more efficient scheduling of threads over time.

When this policy is in use, the scheduler creates a dynamic priority list based partly on the niceness value of each process thread. Administrators can change the niceness value of a process, but cannot change the scheduler’s dynamic priority list directly.

14.5. SETTING SCHEDULER POLICIES

Check and adjust scheduler policies and priorities by using the \texttt{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:

   ```bash
   # 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:

   ```bash
   # 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:
      ```bash
      # chrt -f -p 50 1000
      ```
   b. For example, to set the process with PID 1000 to SCHED_OTHER, with a priority of 0:
      ```bash
      # chrt -o -p 0 1000
      ```
   c. For example, to set the process with PID 1000 to SCHED_RR, with a priority of 10:
      ```bash
      # chrt -r -p 10 1000
      ```
   d. To start a new application with a particular policy and priority, specify the name of the application:
      ```bash
      # chrt -f 36 /bin/my-app
      ```

Additional resources

- [chrt(1) man page](#)
- Policy Options for the chrt command
- Changing the priority of services during the boot process

14.6. POLICY OPTIONS FOR THE CHRT COMMAND

Using the `chrt` command, you can view and set the scheduling policy of a process.

The following table describes the appropriate policy options, which can be used to set the scheduling policy of a process.
### Table 14.1. 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>-f</td>
<td>--fifo</td>
<td>Set schedule to <strong>SCHED_FIFO</strong></td>
</tr>
<tr>
<td>-o</td>
<td>--other</td>
<td>Set schedule to <strong>SCHED_OTHER</strong></td>
</tr>
<tr>
<td>-r</td>
<td>--rr</td>
<td>Set schedule to <strong>SCHED_RR</strong></td>
</tr>
</tbody>
</table>

### 14.7. CHANGING THE PRIORITY OF SERVICES DURING THE BOOT PROCESS

Using the **systemd** service, it is possible to set up real-time priorities for services launched during the boot process. The **unit configuration directives** are used to change the priority of a service during the boot process.

The boot process priority change is done by using the following directives in the service section:

- **CPUSchedulingPolicy=**
  Sets the CPU scheduling policy for executed processes. It is used to set **other**, **fifo**, and **rr** policies.

- **CPUSchedulingPriority=**
  Sets the CPU scheduling priority for executed processes. The available priority range depends on the selected CPU scheduling policy. For real-time scheduling policies, an integer between 1 (lowest priority) and 99 (highest priority) can be used.

The following procedure describes how to change the priority of a service, during the boot process, using the **mcelog** service.

**Prerequisites**

1. Install the tuned package:
   ```
   # dnf install tuned
   ```

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

**Procedure**

1. View the scheduling priorities of running threads:
   ```
   # tuna --show_threads
<table>
<thead>
<tr>
<th>thread</th>
<th>ctxt_switches</th>
<th>pid</th>
<th>SCHED_rtpri</th>
<th>affinity voluntary nonvoluntary</th>
<th>cmd</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTHER</td>
<td>0 0xff 3181</td>
<td>292</td>
<td>OTHER</td>
<td>0</td>
<td>systemd</td>
</tr>
<tr>
<td>OTHER</td>
<td>2 0xff 254</td>
<td>0</td>
<td>OTHER</td>
<td>0</td>
<td>kthread</td>
</tr>
<tr>
<td>OTHER</td>
<td>0 0xff 2</td>
<td>0</td>
<td>OTHER</td>
<td>0</td>
<td>rcu_gp</td>
</tr>
<tr>
<td>OTHER</td>
<td>0 0xff 2</td>
<td>0</td>
<td>OTHER</td>
<td>0</td>
<td>rcu_par_gp</td>
</tr>
</tbody>
</table>
   ```

Red Hat Enterprise Linux 9.0 Monitoring and managing system status and performance
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
thread  ctxt_switches
pid SCHED_ rtpri affinity voluntary nonvoluntary     cmd
826    FIFO  20 0,1,2,3   13   0     mcelog
```

**Additional resources**

- `systemd(1)` and `tuna(8)` man pages
- Description of the priority range

### 14.8. PRIORITY MAP

Priorities are defined in groups, with some groups dedicated to certain kernel functions. For real-time scheduling policies, an integer between 1 (lowest priority) and 99 (highest priority) can be used.

The following table describes the priority range, which can be used while setting the scheduling policy of a process.

**Table 14.2. Description of the priority range**
<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>

### 14.9. TUNED CPU-PARTITIONING PROFILE

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

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

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

**Figure 14.1. Figure cpu-partitioning**

![Figure cpu-partitioning](image)

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

- **Isolated CPUs with load balancing**

---

Number of CPUs: 24

- 8 CPUs per node
- 3 nodes

<table>
<thead>
<tr>
<th>Numa node</th>
<th>Housekeeping CPUs</th>
<th>Isolated CPUs with no scheduler load balancing</th>
<th>Isolated CPUs with scheduler load balancing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In the cpu-partitioning figure, the blocks numbered from 4 to 23, are the default isolated CPUs. The kernel scheduler’s process load balancing is enabled on these CPUs. It is designed for low-latency processes with multiple threads that need the kernel scheduler load balancing. You can configure the cpu-partitioning profile in the `/etc/tuned/cpu-partitioning-variables.conf` file using the `isolated_cores=cpu-list` option, which lists CPUs to isolate that will use the kernel scheduler load balancing.

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

**Isolated CPUs without load balancing**

In the cpu-partitioning figure, the blocks numbered 2 and 3, are the isolated CPUs that do not provide any additional kernel scheduler process load balancing. You can configure the cpu-partitioning profile in the `/etc/tuned/cpu-partitioning-variables.conf` file using the `no_balance_cores=cpu-list` option, which lists CPUs to isolate that will not use the kernel scheduler load balancing.

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

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

**Housekeeping CPUs**

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

**Additional resources**

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

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

This procedure describes how to tune a system for low-latency using the TuneD’s `cpu-partitioning` profile. It uses the example of a low-latency application that can use `cpu-partitioning` and the CPU layout as mentioned in the `cpu-partitioning` figure.

The application in this case uses:

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

**Prerequisites**

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

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

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

2. Set the `cpu-partitioning` TuneD profile:

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

3. Reboot

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

**Additional resources**

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

**14.11. CUSTOMIZING THE CPU-PARTITIONING TUNED PROFILE**

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

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

**Procedure**

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

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

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

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

3. Use the new profile:

   ```bash
   # tuned-adm profile my_profile
   ```
NOTE

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

Additional resources

- `tuned-profiles-cpu-partitioning(7)` man page
CHAPTER 15. USING SYSTEMD TO MANAGE RESOURCES USED BY APPLICATIONS

RHEL 9 moves the resource management settings from the process level to the application level by binding the system of cgroup hierarchies with the systemd unit tree. Therefore, you can manage the system resources with the systemctl command, or by modifying the systemd unit files.

To achieve this, systemd takes various configuration options from the unit files or directly via the systemctl command. Then systemd applies those options to specific process groups by utilizing the Linux kernel system calls and features like cgroups and namespaces.

NOTE
You can review the full set of configuration options for systemd in the following manual pages:

- systemd.resource-control(5)
- systemd.exec(5)

15.1. ALLOCATING SYSTEM RESOURCES USING SYSTEMD

To modify the distribution of system resources, you can apply one or more of the following distribution models:

Weights
You can distribute the resource by adding up the weights of all sub-groups and giving each sub-group the fraction matching its ratio against the sum.
For example, if you have 10 cgroups, each with weight of value 100, the sum is 1000. Each cgroup receives one tenth of the resource.

Weight is usually used to distribute stateless resources. For example the CPUWeight= option is an implementation of this resource distribution model.

Limits
A cgroup can consume up to the configured amount of the resource. The sum of sub-group limits can exceed the limit of the parent cgroup. Therefore it is possible to overcommit resources in this model.
For example the MemoryMax= option is an implementation of this resource distribution model.

Protections
You can set up a protected amount of a resource for a cgroup. If the resource usage is below the protection boundary, the kernel will try not to penalize this cgroup in favor of other cgroups that compete for the same resource. An overcommit is also possible.
For example the MemoryLow= option is an implementation of this resource distribution model.

Allocations
Exclusive allocations of an absolute amount of a finite resource. An overcommit is not possible. An example of this resource type in Linux is the real-time budget.

unit file option
A setting for resource control configuration.
For example, you can configure CPU resource with options like `CPUAccounting=`, or `CPUQuota=`. Similarly, you can configure memory or I/O resources with options like `AllowedMemoryNodes=` and `IOAccounting=`.

**Procedure**

To change the required value of the unit file option of your service, you can adjust the value in the unit file, or use `systemctl` command:

1. Check the assigned values for the service of your choice.
   
   ```shell
   # systemctl show --property <unit file option> <service name>
   ```

2. Set the required value of the CPU time allocation policy option:
   
   ```shell
   # systemctl set-property <service name> <unit file option>=<value>
   ```

**Verification steps**

- Check the newly assigned values for the service of your choice.
  
  ```shell
  # systemctl show --property <unit file option> <service name>
  ```

**Additional resources**

- `systemd.resource-control(5)`, `systemd.exec(5)` manual pages

### 15.2. ROLE OF SYSTEMD IN RESOURCE MANAGEMENT

The core function of `systemd` is service management and supervision. The `systemd` system and service manager ensures that managed services start at the right time and in the correct order during the boot process. The services have to run smoothly to use the underlying hardware platform optimally. Therefore, `systemd` also provides capabilities to define resource management policies, and to tune various options, which can improve the performance of the service.

**IMPORTANT**

In general, Red Hat recommends you use `systemd` for controlling the usage of system resources. You should manually configure the `cgroups` virtual file system only in special cases. For example, when you need to use `cgroup-v1` controllers that have no equivalents in `cgroup-v2` hierarchy.

### 15.3. OVERVIEW OF SYSTEMD HIERARCHY FOR CGROUPS

On the backend, the `systemd` system and service manager makes use of the `slice`, the `scope` and the `service` units to organize and structure processes in the control groups. You can further modify this hierarchy by creating custom unit files or using the `systemctl` command. Also, `systemd` automatically mounts hierarchies for important kernel resource controllers at the `/sys/fs/cgroup/` directory.

Three `systemd` unit types are used for resource control:
- **Service** - A process or a group of processes, which **systemd** started according to a unit configuration file. Services encapsulate the specified processes so that they can be started and stopped as one set. Services are named in the following way:

  `<name>.service`  

- **Scope** - A group of externally created processes. Scopes encapsulate processes that are started and stopped by the arbitrary processes through the `fork()` function and then registered by **systemd** at runtime. For example, user sessions, containers, and virtual machines are treated as scopes. Scopes are named as follows:

  `<name>.scope`  

- **Slice** - A group of hierarchically organized units. Slices organize a hierarchy in which scopes and services are placed. The actual processes are contained in scopes or in services. Every name of a slice unit corresponds to the path to a location in the hierarchy. The dash (“-”) character acts as a separator of the path components to a slice from the `-slice` root slice. In the following example:

  `<parent-name>.slice`  

  

  `parent-name.slice` is a sub-slice of `parent.slice`, which is a sub-slice of the `-slice` root slice. `parent-name.slice` can have its own sub-slice named `parent-name-name2.slice`, and so on.

The **service**, the **scope**, and the **slice** units directly map to objects in the control group hierarchy. When these units are activated, they map directly to control group paths built from the unit names.

The following is an abbreviated example of a control group hierarchy:

```
Control group /:  
  -.slice  
    ├─ user.slice  
    ├─ user-42.slice  
    │   └─ session-c1.scope  
    │       └─ 967 gdm-session-worker [pam/gdm-launch-environment]  
    │               └─ 1035 /usr/libexec/gdm-x-session gnome-session --autostart  
    |                  /usr/share/gdm/greeter/autostart  
    |                  └─ 1054 /usr/libexec/Xorg vt1 -displayfd 3 -auth /run/user/42/gdm/Xauthority -background none  
    |                          -noreset -keeptty -verbose 3  
    |                          └─ 1212 /usr/libexec/gnome-session-binary --autostart /usr/share/gdm/greeter/autostart  
    |                          └─ 1369 /usr/bin/gnome-shell  
    |                              └─ 1732 ibus-daemon --xim --panel disable  
    |                                          └─ 1752 /usr/libexec/ibus-dconf  
    |                                          └─ 1762 /usr/libexec/ibus-x11 --kill-daemon  
    |                                              └─ 1912 /usr/libexec/gsd-xsettings  
    |                                              └─ 1917 /usr/libexec/gsd-a11y-settings  
    |                                                  └─ 1920 /usr/libexec/gsd-clipboard  
    |                                                      ...  
    |                                                      └─ init.scope  
    |                                                          └─ 1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18  
    |                                                          └─ systemd.slice  
    |                                                              └─ mgd.service  
    |                                                                  └─ 800 /sbin/mgd -f  
    |                                                                      └─ systemd-udevd.service  
    |                                                                  └─ 659 /usr/lib/systemd/systemd-udevd  
```
The example above shows that services and scopes contain processes and are placed in slices that do not contain processes of their own.

Additional resources

- Configuring basic system settings in Red Hat Enterprise Linux
- What are kernel resource controllers
- systemd.resource-control(5), systemd.exec(5), cgroups(7), fork(), fork(2) manual pages
- Understanding cgroups

15.4. LISTING SYSTEMD UNITS

The following procedure describes how to use the systemd system and service manager to list its units.

Procedure

- To list all active units on the system, execute the `# systemctl` command and the terminal will return an output similar to the following example:

  ```
  # systemctl
  UNIT                              LOAD   ACTIVE SUB       DESCRIPTION
  ...
  init.scope                       loaded active running   System and Service Manager
  session-2.scope                 loaded active running   Session 2 of user jdoe
  abrt-ccpp.service               loaded active exited    Install ABRT coredump hook
  abrt-oops.service               loaded active running   ABRT kernel log watcher
  abrt-vmcore.service             loaded active exited    Harvest vmcores for ABRT
  abrt-xorg.service               loaded active running   ABRT Xorg log watcher
  ...
  -.slice                         loaded active active   Root Slice
  machine.slice                   loaded active active   Virtual Machine and Container
  Slice system-getty.slice        loaded active active   loaded active active
  system-getty.slice
  system-lvma2x2dpvscan.slice     loaded active active   system-
  lvm2a2x2dpvscan.slice
  system-sshdx2dkeygen.slice     loaded active active   system-
  sshdx2dkeygen.slice
  system-systemd\x2dhibernate\x2dresume.slice loaded active active   system-
  systemd\x2dhibernate\x2dresume>
  system-user\x2druntime\x2ddir.slice loaded active active   system-
  ```
UNIT - a name of a unit that also reflects the unit position in a control group hierarchy. The units relevant for resource control are a slice, a scope, and a service.

LOAD - indicates whether the unit configuration file was properly loaded. If the unit file failed to load, the field contains the state error instead of loaded. Other unit load states are: stub, merged, and masked.

ACTIVE - the high-level unit activation state, which is a generalization of SUB.

SUB - the low-level unit activation state. The range of possible values depends on the unit type.

DESCRIPTION - the description of the unit content and functionality.

To list inactive units, execute:

```
# systemctl --all
```

To limit the amount of information in the output, execute:

```
# systemctl --type service,masked
```

The --type option requires a comma-separated list of unit types such as a service and a slice, or unit load states such as loaded and masked.

Additional resources

- Configuring basic system settings in RHEL
- systemd.resource-control(5), systemd.exec(5) manual pages

15.5. VIEWING SYSTEMD CONTROL GROUP HIERARCHY

The following procedure describes how to display control groups (cgroups) hierarchy and processes running in specific cgroups.

Procedure

- To display the whole cgroups hierarchy on your system, execute # systemctl-cgls:

```
# systemctl-cgls
Control group /:
  -.slice
    └─user.slice
      └─user-42.slice
        └─session-c1.scope
          └─965 gdm-session-worker [pam/gdm-launch-environment]
          └─1040 /usr/libexec/gdm-x-session gnome-session --autostart
```
The example output returns the entire cgroups hierarchy, where the highest level is formed by slices.

- To display the cgroups hierarchy filtered by a resource controller, execute `# systemd-cgls <resource_controller>`:

```
# systemd-cgls memory
Controller memory; Control group /:
├─ /usr/lib/systemd/systemd --switched-root --system --deserialize 18
│  └─ user.slice
│       └─ session-c1.scope
│           └─ 965 gdm-session-worker [pam/gdm-launch-environment]
└─ system.slice
```

The example output of the above command lists the services that interact with the selected controller.

- To display detailed information about a certain unit and its part of the cgroups hierarchy, execute `# systemctl status <system_unit>`:

```
# systemctl status example.service
● example.service - My example service
   Loaded: loaded (/usr/lib/systemd/system/example.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2019-04-16 12:12:39 CEST; 3s ago
   Main PID: 17737 (bash)
   Tasks: 2 (limit: 11522)
   Memory: 496.0K (limit: 1.5M)
   CGroup: /system.slice/example.service
       └─ 17737 /bin/bash /home/jdoe/example.sh
```

CHAPTER 15. USING SYSTEMD TO MANAGE RESOURCES USED BY APPLICATIONS
15.6. VIEWING CGROUPS OF PROCESSES

The following procedure describes how to learn which control group (cgroup) a process belongs to. Then you can check the cgroup to learn which controllers and controller-specific configurations it uses.

Procedure

1. To view which cgroup a process belongs to, run the `# cat /proc/<PID>/cgroup` command:

   ```
   # cat /proc/2467/cgroup
   0::/system.slice/example.service
   ```

   The example output relates to a process of interest. In this case, it is a process identified by PID 2467, which belongs to the example.service unit. You can determine whether the process was placed in a correct control group as defined by the systemd unit file specifications.

2. To display what controllers the cgroup utilizes and the respective configuration files, check the cgroup directory:

   ```
   # cat /sys/fs/cgroup/system.slice/example.service/cgroup.controllers
   memory pids
   
   # ls /sys/fs/cgroup/system.slice/example.service/
   cgroup.controllers
   cgroup.events
   ...
   cpu.pressure
   cpu.stat
   io.pressure
   memory.current
   memory.events
   ...
   pids.current
   pids.events
   pids.max
   ```

   **NOTE**

   The version 1 hierarchy of cgroups uses a per-controller model. Therefore the output from the `/proc/<PID>cgroup` file shows, which cgroups under each controller the PID belongs to. You can find the respective cgroups under the controller directories at /sys/fs/cgroup/<controller_name>.
Additional resources

- cgroups(7) manual page
- What are kernel resource controllers
- Documentation in the /usr/share/doc/kernel-doc-<kernel_version>/Documentation/admin-guide/cgroup-v2.rst file (after installing the kernel-doc package)

15.7. MONITORING RESOURCE CONSUMPTION

The following procedure describes how to view a list of currently running control groups (cgroups) and their resource consumption in real-time.

Procedure

1. To see a dynamic account of currently running cgroups, execute the # systemd-cgtop command:

```
# systemd-cgtop
```

<table>
<thead>
<tr>
<th>Control Group</th>
<th>Tasks</th>
<th>%CPU</th>
<th>Memory</th>
<th>Input/s</th>
<th>Output/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>607</td>
<td>29.8</td>
<td>1.5G</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice</td>
<td>125</td>
<td>-</td>
<td>428.7M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/ModemManager.service</td>
<td>3</td>
<td>-</td>
<td>8.6M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/NetworkManager.service</td>
<td>3</td>
<td>-</td>
<td>12.8M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/accounts-daemon.service</td>
<td>3</td>
<td>-</td>
<td>1.8M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/boot.mount</td>
<td>-</td>
<td>-</td>
<td>48.0K</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/chronyd.service</td>
<td>1</td>
<td>-</td>
<td>2.0M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/cockpit.socket</td>
<td>-</td>
<td>-</td>
<td>1.3M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/color.service</td>
<td>3</td>
<td>-</td>
<td>3.5M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/crond.service</td>
<td>1</td>
<td>-</td>
<td>1.8M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/cups.service</td>
<td>1</td>
<td>-</td>
<td>3.1M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/dev-hugepages.mount</td>
<td>-</td>
<td>-</td>
<td>244.0K</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/swap</td>
<td>-</td>
<td>-</td>
<td>912.0K</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/dev-mqueue.mount</td>
<td>-</td>
<td>-</td>
<td>48.0K</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/example.service</td>
<td>2</td>
<td>-</td>
<td>2.0M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/firewalld.service</td>
<td>2</td>
<td>-</td>
<td>28.8M</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

The example output displays currently running cgroups ordered by their resource usage (CPU, memory, disk I/O load). The list refreshes every 1 second by default. Therefore, it offers a dynamic insight into the actual resource usage of each control group.

Additional resources

- systemd-cgtop(1) manual page

15.8. USING SYSTEMD UNIT FILES TO SET LIMITS FOR APPLICATIONS

Each existing or running unit is supervised by the systemd, which also creates control groups for them. The units have configuration files in the /usr/lib/systemd/system/ directory. You can manually modify the unit files to set limits, prioritize, or control access to hardware resources for groups of processes.

Prerequisites
You have the root privileges.

Procedure

1. Modify the /usr/lib/systemd/system/example.service file to limit the memory usage of a service:

```
[Service]
MemoryMax=1500K
```

The configuration above places a maximum memory limit, which the processes in a control group cannot exceed. The example.service service is part of such a control group which has imposed limitations. You can use suffixes K, M, G, or T to identify Kilobyte, Megabyte, Gigabyte, or Terabyte as a unit of measurement.

2. Reload all unit configuration files:

```
# systemctl daemon-reload
```

3. Restart the service:

```
# systemctl restart example.service
```

NOTE

You can review the full set of configuration options for systemd in the following manual pages:

- systemd.resource-control(5)
- systemd.exec(5)

Verification

1. Check that the changes took effect:

```
# cat /sys/fs/cgroup/system.slice/example.service/memory.max
1536000
```

The example output shows that the memory consumption was limited at around 1,500 KB.

Additional resources

- Understanding cgroups
- Configuring basic system settings in Red Hat Enterprise Linux
- systemd.resource-control(5), systemd.exec(5), cgroups(7) manual pages

15.9. USING SYSTEMCTL COMMAND TO SET LIMITS TO APPLICATIONS
CPU affinity settings help you restrict the access of a particular process to some CPUs. Effectively, the CPU scheduler never schedules the process to run on the CPU that is not in the affinity mask of the process.

The default CPU affinity mask applies to all services managed by systemd.

To configure CPU affinity mask for a particular systemd service, systemd provides CPUAffinity= both as a unit file option and a manager configuration option in the /etc/systemd/system.conf file.

The CPUAffinity= unit file option sets a list of CPUs or CPU ranges that are merged and used as the affinity mask.

After configuring CPU affinity mask for a particular systemd service, you must restart the service to apply the changes.

Procedure

To set CPU affinity mask for a particular systemd service using the CPUAffinity unit file option:

1. Check the values of the CPUAffinity unit file option in the service of your choice:

   ```
   $ systemctl show --property <CPU affinity configuration option> <service name>
   ```

2. As a root, set the required value of the CPUAffinity unit file option for the CPU ranges used as the affinity mask:

   ```
   # systemctl set-property <service name> CPUAffinity=<value>
   ```

3. Restart the service to apply the changes.

   ```
   # systemctl restart <service name>
   ```

   **NOTE**

   You can review the full set of configuration options for systemd in the following manual pages:

   - systemd.resource-control(5)
   - systemd.exec(5)

**15.10. SETTING GLOBAL DEFAULT CPU AFFINITY THROUGH MANAGER CONFIGURATION**

The CPUAffinity option in the /etc/systemd/system.conf file defines an affinity mask for the process identification number (PID) 1 and all processes forked off of PID1. You can then override the CPUAffinity on a per-service basis.

To set default CPU affinity mask for all systemd services using the manager configuration option:

1. Set the CPU numbers for the CPUAffinity= option in the /etc/systemd/system.conf file.

2. Save the edited file and reload the systemd service:
3. Reboot the server to apply the changes.

NOTE
You can review the full set of configuration options for systemd in the following manual pages:

- systemd.resource-control(5)
- systemd.exec(5)

15.11. CONFIGURING NUMA POLICIES USING SYSTEMD

Non-uniform memory access (NUMA) is a computer memory subsystem design, in which the memory access time depends on the physical memory location relative to the processor.

Memory close to the CPU has lower latency (local memory) than memory that is local for a different CPU (foreign memory) or is shared between a set of CPUs.

In terms of the Linux kernel, NUMA policy governs where (for example, on which NUMA nodes) the kernel allocates physical memory pages for the process.

systemd provides unit file options NUMAPolicy and NUMAMask to control memory allocation policies for services.

Procedure
To set the NUMA memory policy through the NUMAPolicy unit file option:

1. Check the values of the NUMAPolicy unit file option in the service of your choice:

   ```shell
   $ systemctl show --property <NUMA policy configuration option> <service name>
   ```

2. As a root, set the required policy type of the NUMAPolicy unit file option:

   ```shell
   # systemctl set-property <service name> NUMAPolicy=<value>
   ```

3. Restart the service to apply the changes.

   ```shell
   # systemctl restart <service name>
   ```

To set a global NUMAPolicy setting through the manager configuration option:

1. Search in the /etc/systemd/system.conf file for the NUMAPolicy option.

2. Edit the policy type and save the file.

3. Reload the systemd configuration:

   ```shell
   # systemctl daemon-reload
   ```

4. Reboot the server.
IMPORTANT

When you configure a strict NUMA policy, for example bound, make sure that you also appropriately set the CPUAffinity unit file option.

Additional resources

- Using systemctl command to set limits to applications

15.12. NUMA POLICY CONFIGURATION OPTIONS FOR SYSTEMD

Systemd provides the following options to configure the NUMA policy:

**NUMAPolicy**

Controls the NUMA memory policy of the executed processes. The following policy types are possible:

- default
- preferred
- bind
- interleave
- local

**NUMAMask**

Controls the NUMA node list which is associated with the selected NUMA policy. Note that the NUMAMask option is not required to be specified for the following policies:

- default
- local

For the preferred policy, the list specifies only a single NUMA node.

Additional resources

- systemd.resource-control(5), systemd.exec(5), and set_mempolicy(2) manual pages

15.13. CREATING TRANSIENT CGROUPS USING SYSTEMD-RUN COMMAND

The transient cgroups set limits on resources consumed by a unit (service or scope) during its runtime.

**Procedure**

- To create a transient control group, use the systemd-run command in the following format:

  ```
  # systemd-run --unit=<name> --slice=<name>.slice <command>
  ```
This command creates and starts a transient service or a scope unit and runs a custom command in such a unit.

- The `--unit=<name>` option gives a name to the unit. If `--unit` is not specified, the name is generated automatically.
- The `--slice=<name>.slice` option makes your service or scope unit a member of a specified slice. Replace `<name>.slice` with the name of an existing slice (as shown in the output of `systemctl -t slice`), or create a new slice by passing a unique name. By default, services and scopes are created as members of the `system.slice`.
- Replace `<command>` with the command you wish to execute in the service or the scope unit.

The following message is displayed to confirm that you created and started the service or the scope successfully:

```
# Running as unit <name>.service
```

- Optionally, keep the unit running after its processes finished to collect run-time information:

```
# systemd-run --unit=<name> --slice=<name>.slice --remain-after-exit <command>
```

The command creates and starts a transient service unit and runs a custom command in such a unit. The `--remain-after-exit` option ensures that the service keeps running after its processes have finished.

Additional resources
- Understanding control groups
- Configuring basic system settings in RHEL
- `systemd-run(1)` manual page

### 15.14. REMOVING TRANSIENT CONTROL GROUPS

You can use the systemd system and service manager to remove transient control groups (cgroups) if you no longer need to limit, prioritize, or control access to hardware resources for groups of processes.

Transient cgroups are automatically released once all the processes that a service or a scope unit contains, finish.

**Procedure**

- To stop the service unit with all its processes, execute:
  ```
  # systemctl stop name.service
  ```

- To terminate one or more of the unit processes, execute:
  ```
  # systemctl kill name.service --kill-who=PID,... --signal=<signal>
  ```

The command above uses the `--kill-who` option to select process(es) from the control group.
you wish to terminate. To kill multiple processes at the same time, pass a comma-separated list of PIDs. The \texttt{--signal} option determines the type of POSIX signal to be sent to the specified processes. The default signal is \texttt{SIGTERM}.

Additional resources

- Understanding control groups
- What are kernel resource controllers
- \texttt{systemd.resource-control(5), cgroups(7)} manual pages
- Role of systemd in control groups
- \texttt{Configuring basic system settings} in RHEL
CHAPTER 16. UNDERSTANDING CGROUPS

You can use the control groups (cgroups) kernel functionality to set limits, prioritize or isolate the hardware resources of processes. This allows you to granularly control resource usage of applications to utilize them more efficiently.

16.1. UNDERSTANDING CONTROL GROUPS

Control groups is a Linux kernel feature that enables you to organize processes into hierarchically ordered groups - cgroups. The hierarchy (control groups tree) is defined by providing structure to cgroups virtual file system, mounted by default on the /sys/fs/cgroup/ directory. The systemd system and service manager utilizes cgroups to organize all units and services that it governs. Alternatively, you can manage cgroups hierarchies manually by creating and removing sub-directories in the /sys/fs/cgroup/ directory.

The resource controllers (a kernel component) then modify the behavior of processes in cgroups by limiting, prioritizing or allocating system resources, (such as CPU time, memory, network bandwidth, or various combinations) of those processes.

The added value of cgroups is process aggregation which enables division of hardware resources among applications and users. Thereby an increase in overall efficiency, stability and security of users' environment can be achieved.

Control groups version 1

Control groups version 1 (cgroups-v1) provide a per-resource controller hierarchy. It means that each resource, such as CPU, memory, I/O, and so on, has its own control group hierarchy. It is possible to combine different control group hierarchies in a way that one controller can coordinate with another one in managing their respective resources. However, the two controllers may belong to different process hierarchies, which does not permit their proper coordination.

The cgroups-v1 controllers were developed across a large time span and as a result, the behavior and naming of their control files is not uniform.

Control groups version 2

The problems with controller coordination, which stemmed from hierarchy flexibility, led to the development of control groups version 2.

Control groups version 2 (cgroups-v2) provides a single control group hierarchy against which all resource controllers are mounted.

The control file behavior and naming is consistent among different controllers.

IMPORTANT

RHEL 9, by default, mounts and utilizes cgroups-v2.

This sub-section was based on a Devconf.cz 2019 presentation.[2]

Additional resources

- What are kernel resource controllers
- cgroups(7) manual page
16.2. WHAT ARE KERNEL RESOURCE CONTROLLERS

The functionality of control groups is enabled by kernel resource controllers. RHEL 9 supports various controllers for control groups version 1 (cgroups-v1) and control groups version 2 (cgroups-v2).

A resource controller, also called a control group subsystem, is a kernel subsystem that represents a single resource, such as CPU time, memory, network bandwidth or disk I/O. The Linux kernel provides a range of resource controllers that are mounted automatically by the systemd system and service manager. Find a list of currently mounted resource controllers in the /proc/cgroups file.

The following controllers are available for cgroups-v1:

- **blkio** - can set limits on input/output access to and from block devices.
- **cpu** - can adjust the parameters of the Completely Fair Scheduler (CFS) scheduler for control group’s tasks. It is mounted together with the cpuacct controller on the same mount.
- **cpuacct** - creates automatic reports on CPU resources used by tasks in a control group. It is mounted together with the cpu controller on the same mount.
- **cpuset** - can be used to restrict control group tasks to run only on a specified subset of CPUs and to direct the tasks to use memory only on specified memory nodes.
- **devices** - can control access to devices for tasks in a control group.
- **freezer** - can be used to suspend or resume tasks in a control group.
- **memory** - can be used to set limits on memory use by tasks in a control group and generates automatic reports on memory resources used by those tasks.
- **net_cls** - tags network packets with a class identifier (classid) that enables the Linux traffic controller (the tc command) to identify packets that originate from a particular control group task. A subsystem of net_cls, the net_filter (iptables), can also use this tag to perform actions on such packets. The net_filter tags network sockets with a firewall identifier (fwid) that allows the Linux firewall (through iptables command) to identify packets originating from a particular control group task.
- **net_prio** - sets the priority of network traffic.
- **pids** - can set limits for a number of processes and their children in a control group.
- **perf_event** - can group tasks for monitoring by the perf performance monitoring and reporting utility.
- **rdma** - can set limits on Remote Direct Memory Access/InfiniBand specific resources in a control group.
- **hugetlb** - can be used to limit the usage of large size virtual memory pages by tasks in a control group.

The following controllers are available for cgroups-v2:

- **io** - A follow-up to blkio of cgroups-v1.
• **memory** - A follow-up to memory of cgroups-v1.

• **pids** - Same as pids in cgroups-v1.

• **rdma** - Same as rdma in cgroups-v1.

• **cpu** - A follow-up to cpu and cpuacct of cgroups-v1.

• **cpuset** - Supports only the core functionality (`cpus{,.effective}, mems{,.effective}`) with a new partition feature.

• **perf_event** - Support is inherent, no explicit control file. You can specify a `v2 cgroup` as a parameter to the `perf` command that will profile all the tasks within that `cgroup`.

**IMPORTANT**

A resource controller can be used either in a cgroups-v1 hierarchy or a cgroups-v2 hierarchy, not simultaneously in both.

**Additional resources**

- cgroups(7) manual page


### 16.3. WHAT ARE NAMESPACES

Namespaces are one of the most important methods for organizing and identifying software objects.

A namespace wraps a global system resource (for example a mount point, a network device, or a hostname) in an abstraction that makes it appear to processes within the namespace that they have their own isolated instance of the global resource. One of the most common technologies that utilize namespaces are containers.

Changes to a particular global resource are visible only to processes in that namespace and do not affect the rest of the system or other namespaces.

To inspect which namespaces a process is a member of, you can check the symbolic links in the `/proc/<PID>/ns/` directory.

The following table shows supported namespaces and resources which they isolate:

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Isolates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mount</td>
<td>Mount points</td>
</tr>
<tr>
<td>UTS</td>
<td>Hostname and NIS domain name</td>
</tr>
<tr>
<td>IPC</td>
<td>System V IPC, POSIX message queues</td>
</tr>
<tr>
<td>PID</td>
<td>Process IDs</td>
</tr>
</tbody>
</table>
### Namespace Isolates

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Isolates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
<td>Network devices, stacks, ports, etc</td>
</tr>
<tr>
<td>User</td>
<td>User and group IDs</td>
</tr>
<tr>
<td>Control groups</td>
<td>Control group root directory</td>
</tr>
</tbody>
</table>

**Additional resources**

- namespaces(7) and cgroup_namespaces(7) manual pages
- Understanding control groups

CHAPTER 17. IMPROVING SYSTEM PERFORMANCE WITH ZSWAP

You can improve system performance by enabling the `zswap` kernel feature.

17.1. WHAT IS ZSWAP

This section explains what `zswap` is and how it can lead to system performance improvement.

`zswap` is a kernel feature that provides a compressed RAM cache for swap pages. The mechanism works as follows: `zswap` takes pages that are in the process of being swapped out and attempts to compress them into a dynamically allocated RAM-based memory pool. When the pool becomes full or the RAM becomes exhausted, `zswap` evicts pages from compressed cache on an LRU basis (least recently used) to the backing swap device. After the page has been decompressed into the swap cache, `zswap` frees the compressed version in the pool.

The benefits of `zswap`

- significant I/O reduction
- significant improvement of workload performance

In Red Hat Enterprise Linux 9, `zswap` is enabled by default.

Additional resources

- What is Zswap?

17.2. ENABLING ZSWAP AT RUNTIME

You can enable the `zswap` feature at system runtime using the `sysfs` interface.

Prerequisites

- You have root permissions.

Procedure

- Enable `zswap`:
  
  ```bash
  # echo 1 > /sys/module/zswap/parameters/enabled
  ```

Verification step

- Verify that `zswap` is enabled:
  
  ```bash
  # grep -r . /sys/kernel/debug/zswap
  duplicate_entry:0
  pool_limit_hit:13422200
  pool_total_size:6184960 (pool size in total in pages)
  reject_alloc_fail:5
  ```
reject_compress_poor:0
reject_kmemcache_fail:0
reject_reclaim_fail:13422200
stored_pages:4251 (pool size after compression)
written_back_pages:0

Additional resources

- How to enable Zswap feature?

17.3. ENABLING ZSWAP PERMANENTLY

You can enable the **zswap** feature permanently by providing the `zswap.enabled=1` kernel command-line parameter.

**Prerequisites**

- You have root permissions.
- The **grubby** or **zipl** utility is installed on your system.

**Procedure**

1. Enable **zswap** permanently:

   ```
   # grubby --update-kernel=/boot/vmlinuz-$(uname -r) --args="zswap.enabled=1"
   ```

2. Reboot the system for the changes to take effect.

**Verification steps**

- Verify that **zswap** is enabled:

  ```
  # cat /proc/cmdline
  ```

  ```
  BOOT_IMAGE=(hd0,msdos1)/vmlinuz-5.14.0-70.5.1.el9_0.x86_64
  root=/dev/mapper/rhel-root ro crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M
  resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
  rd.lvm.lv=rhel/swap rhgb quiet
  zswap.enabled=1
  ```

**Additional resources**

- How to enable Zswap feature?
- Configuring kernel command-line parameters
CHAPTER 18. USING CGROUPFS TO MANUALLY MANAGE CGROUPS

You can manage `cgroup` hierarchies on your system by creating directories on the `cgroupfs` virtual file system. The file system is mounted by default on the `/sys/fs/cgroup/` directory and you can specify desired configurations in dedicated control files.

**IMPORTANT**

In general, Red Hat recommends you use `systemd` for controlling the usage of system resources. You should manually configure the `cgroups` virtual file system only in special cases. For example, when you need to use `cgroup-v1` controllers that have no equivalents in `cgroup-v2` hierarchy.

18.1. CREATING CGROUPS AND ENABLING CONTROLLERS IN CGROUPS-V2 FILE SYSTEM

You can manage the control groups (`cgroups`) by creating or removing directories and by writing to files in the `cgroups` virtual file system. The file system is by default mounted on the `/sys/fs/cgroup/` directory. To use settings from the `cgroups` controllers, you also need to enable the desired controllers for child `cgroups`. The root `cgroup` has, by default, enabled the `memory` and `pids` controllers for its child `cgroups`. Therefore, Red Hat recommends to create at least two levels of child `cgroups` inside the `/sys/fs/cgroup/` root `cgroup`. This way you optionally remove the `memory` and `pids` controllers from the child `cgroups` and maintain better organizational clarity of `cgroup` files.

**Prerequisites**

- You have root permissions.

**Procedure**

1. Create the `/sys/fs/cgroup/Example/` directory:

   ```bash
   # mkdir /sys/fs/cgroup/Example/
   ```

   The `/sys/fs/cgroup/Example/` directory defines a child group. When you create the `/sys/fs/cgroup/Example/` directory, some `cgroups-v2` interface files are automatically created in the directory. The `/sys/fs/cgroup/Example/` directory contains also controller-specific files for the `memory` and `pids` controllers.

2. Optionally, inspect the newly created child control group:

   ```bash
   # ll /sys/fs/cgroup/Example/
   -r---r--r-- 1 root root 0 Jun  1 10:33 cgroup.controllers
   -r---r--r-- 1 root root 0 Jun  1 10:33 cgroup.events
   -rw-r--r-- 1 root root 0 Jun  1 10:33 cgroup.freeze
   -rw-r--r-- 1 root root 0 Jun  1 10:33 cgroup.procs
   ...
   -rw-r--r-- 1 root root 0 Jun  1 10:33 cgroup.subtree_control
   -r---r--r-- 1 root root 0 Jun  1 10:33 memory.events.local
   -rw-r--r-- 1 root root 0 Jun  1 10:33 memory.high
   -rw-r--r-- 1 root root 0 Jun  1 10:33 memory.low
   ...
   ```
The example output shows general cgroup control interface files such as cgroup.procs or cgroup.controllers. These files are common to all control groups, regardless of enabled controllers.

The files such as memory.high and pids.max relate to the memory and pids controllers, which are in the root control group (/sys/fs/cgroup/), and are enabled by default by systemd.

By default, the newly created child group inherits all settings from the parent cgroup. In this case, there are no limits from the root cgroup.

3. Verify that the desired controllers are available in the /sys/fs/cgroup/cgroup.controllers file:

```bash
# cat /sys/fs/cgroup/cgroup.controllers
cpuset cpu io memory hugetlb pids rdma
```

4. Enable the desired controllers. In this example it is cpu and cpuset controllers:

```bash
# echo "+cpu" >> /sys/fs/cgroup/cgroup.subtree_control
# echo "+cpuset" >> /sys/fs/cgroup/cgroup.subtree_control
```

These commands enable the cpu and cpuset controllers for the immediate child groups of the /sys/fs/cgroup/ root control group. Including the newly created Example control group. A child group is where you can specify processes and apply control checks to each of the processes based on your criteria.

Users can read the contents of the cgroup.subtree_control file at any level to get an idea of what controllers are going to be available for enablement in the immediate child group.

**NOTE**

By default, the /sys/fs/cgroup/cgroup.subtree_control file in the root control group contains memory and pids controllers.

5. Enable the desired controllers for child cgroups of the Example control group:

```bash
# echo "+cpu +cpuset" >> /sys/fs/cgroup/Example/cgroup.subtree_control
```

This command ensures that the immediate child control group will only have controllers relevant to regulate the CPU time distribution - not to memory or pids controllers.

6. Create the /sys/fs/cgroup/Example/tasks/ directory:

```bash
# mkdir /sys/fs/cgroup/Example/tasks/
```

The /sys/fs/cgroup/Example/tasks/ directory defines a child group with files that relate purely to cpu and cpuset controllers. You can now assign processes to this control group and utilize cpu and cpuset controller options for your processes.

7. Optionally, inspect the child control group:
# ll /sys/fs/cgroup/Example/tasks
-rt----- 1 root root 0 Jun 1 11:45 cgroup.controllers
-rt----- 1 root root 0 Jun 1 11:45 cgroup.events
-rw-r----- 1 root root 0 Jun 1 11:45 cgroup.freeze
-rw-r----- 1 root root 0 Jun 1 11:45 cgroup.max.depth
-rw-r----- 1 root root 0 Jun 1 11:45 cgroup.max.descendants
-rw-r----- 1 root root 0 Jun 1 11:45 cgroup.procs
-r-----    1 root root 0 Jun 1 11:45 cgroup.stat
-rw-r----- 1 root root 0 Jun 1 11:45 cgroup.subtree_control
-rw-r----- 1 root root 0 Jun 1 11:45 cgroup.type
-rw-r----- 1 root root 0 Jun 1 11:45 cpu.max
-rw-r----- 1 root root 0 Jun 1 11:45 cpu.pressure
-rw-r----- 1 root root 0 Jun 1 11:45 cpuset.cpus
-r-----    1 root root 0 Jun 1 11:45 cpuset.cpus.effective
-rw-r----- 1 root root 0 Jun 1 11:45 cpuset.cpus.partition
-rw-r----- 1 root root 0 Jun 1 11:45 cpuset.mems
-r-----    1 root root 0 Jun 1 11:45 cpuset.mems.effective
-r-----    1 root root 0 Jun 1 11:45 cpu.stat
-rw-r----- 1 root root 0 Jun 1 11:45 cpu.weight
-rw-r----- 1 root root 0 Jun 1 11:45 cpu.weight.nice
-rw-r----- 1 root root 0 Jun 1 11:45 io.pressure
-rw-r----- 1 root root 0 Jun 1 11:45 memory.pressure

IMPORTANT

The **cpu** controller is only activated if the relevant child control group has at least 2 processes which compete for time on a single CPU.

Verification steps

- Optional: confirm that you have created a new **cgroup** with only the desired controllers active:

```
# cat /sys/fs/cgroup/Example/tasks/cgroup.controllers
cpuset cpu
```

Additional resources

- Understanding control groups
- What are kernel resource controllers
- Mounting cgroups-v1
- cgroups(7), sysfs(5) manual pages

18.2. CONTROLLING DISTRIBUTION OF CPU TIME FOR APPLICATIONS BY ADJUSTING CPU WEIGHT

You need to assign values to the relevant files of the **cpu** controller to regulate distribution of the CPU time to applications under the specific cgroup tree.

Prerequisites
You have root permissions.

You have applications for which you want to control distribution of CPU time.

You created a two level hierarchy of child control groups inside the `/sys/fs/cgroup/ root control group as in the following example:

```
... Example
    ├── g1
    │    └── g2
    │         └── g3
    └── g2
        └── g3
...
```

You enabled the `cpu` controller in the parent control group and in child control groups similarly as described in Creating cgroups and enabling controllers in cgroups-v2 file system.

Procedure

1. Configure desired CPU weights to achieve resource restrictions within the control groups:

   ```
   # echo "150" > /sys/fs/cgroup/Example/g1/cpu.weight
   # echo "100" > /sys/fs/cgroup/Example/g2/cpu.weight
   # echo "50" > /sys/fs/cgroup/Example/g3/cpu.weight
   ```

2. Add the applications’ PIDs to the `g1`, `g2`, and `g3` child groups:

   ```
   # echo "33373" > /sys/fs/cgroup/Example/g1/cgroup.procs
   # echo "33374" > /sys/fs/cgroup/Example/g2/cgroup.procs
   # echo "33377" > /sys/fs/cgroup/Example/g3/cgroup.procs
   ```

   The example commands ensure that desired applications become members of the `Example/g*/` child cgroups and will get their CPU time distributed as per the configuration of those cgroups.

   The weights of the children cgroups (g1, g2, g3) that have running processes are summed up at the level of the parent cgroup (Example). The CPU resource is then distributed proportionally based on the respective weights.

   As a result, when all processes run at the same time, the kernel allocates to each of them the proportionate CPU time based on their respective cgroup’s `cpu.weight` file:

<table>
<thead>
<tr>
<th>Child cgroup</th>
<th>cpu.weight file</th>
<th>CPU time allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>g1</td>
<td>150</td>
<td>~50% (150/300)</td>
</tr>
<tr>
<td>g2</td>
<td>100</td>
<td>~33% (100/300)</td>
</tr>
<tr>
<td>g3</td>
<td>50</td>
<td>~16% (50/300)</td>
</tr>
</tbody>
</table>

   The value of the `cpu.weight` controller file is not a percentage.
If one process stopped running, leaving cgroup g2 with no running processes, the calculation would omit the cgroup g2 and only account weights of cgroups g1 and g3:

<table>
<thead>
<tr>
<th>Child cgroup</th>
<th>cpu.weight file</th>
<th>CPU time allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>g1</td>
<td>150</td>
<td>~75% (150/200)</td>
</tr>
<tr>
<td>g3</td>
<td>50</td>
<td>~25% (50/200)</td>
</tr>
</tbody>
</table>

**IMPORTANT**

If a child cgroup had multiple running processes, the CPU time allocated to the respective cgroup would be distributed equally to the member processes of that cgroup.

**Verification**

1. Verify that the applications run in the specified control groups:

```bash
# cat /proc/33373/cgroup /proc/33374/cgroup /proc/33377/cgroup
0::/Example/g1
0::/Example/g2
0::/Example/g3
```

The command output shows the processes of the specified applications that run in the Example/g*/ child cgroups.

2. Inspect the current CPU consumption of the throttled applications:

```bash
# top
top - 05:17:18 up 1 day, 18:25, 1 user, load average: 3.03, 3.03, 3.00
Tasks: 95 total, 4 running, 91 sleeping, 0 stopped, 0 zombie
%Cpu(s): 18.1 us, 81.6 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.3 hi, 0.0 si, 0.0 st
MiB Mem : 3737.0 total, 3233.7 free, 132.8 used, 370.5 buff/cache
MiB Swap: 4060.0 total, 4060.0 free, 0.0 used. 3373.1 avail Mem

PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
33373 root      20   0  18720  1748  1460 R  49.5  0.0   415:05.87 sha1sum
33374 root      20   0  18720  1756  1464 R  32.9  0.0   412:58.33 sha1sum
33377 root      20   0  18720  1860  1568 R  16.3  0.0   411:03.12 sha1sum
760 root      20   0 416620  28540 15296 S  0.3    0.7  0:10.23 tuned
  1 root      20   0  186328  14108  9484 S  0.0    0.4  0:02.00 systemd
  2 root      20   0    0    0    0 S  0.0    0.0  0:00.01 kthread
...
```

**NOTE**

We forced all the example processes to run on a single CPU for clearer illustration. The CPU weight applies the same principles also when used on multiple CPUs.
Notice that the CPU resource for the **PID 33373**, **PID 33374**, and **PID 33377** was allocated based on the weights, 150, 100, 50, you assigned to the respective child cgroups. The weights correspond to around 50%, 33%, and 16% allocation of CPU time for each application.

**Additional resources**

- Understanding control groups
- What are kernel resource controllers
- Creating cgroups and enabling controllers in cgroups-v2 file system
- Resource Distribution Models
- `cgroups(7), sysfs(5)` manual pages

### 18.3. MOUNTING CGROUPS-V1

During the boot process, RHEL 9 mounts the **cgroup-v2** virtual filesystem by default. To utilize **cgroup-v1** functionality in limiting resources for your applications, manually configure the system.

**NOTE**

Both **cgroup-v1** and **cgroup-v2** are fully enabled in the kernel. There is no default control group version from the kernel point of view, and is decided by **systemd** to mount at startup.

**Prerequisites**

- You have root permissions.

**Procedure**

1. Configure the system to mount **cgroups-v1** by default during system boot by the **systemd** system and service manager:

   ```
   # grubby --update-kernel=/boot/vmlinuz-$(uname -r) --
   args="systemd.unified_cgroup_hierarchy=0
   systemd.legacy_systemd_cgroup_controller"
   ```

   This adds the necessary kernel command-line parameters to the current boot entry.

   To add the same parameters to all kernel boot entries:

   ```
   # grubby --update-kernel=ALL --args="systemd.unified_cgroup_hierarchy=0
   systemd.legacy_systemd_cgroup_controller"
   ```

2. Reboot the system for the changes to take effect.

**Verification**

1. Optionally, verify that the **cgroups-v1** filesystem was mounted:

   ```
   # mount -l | grep cgroup
   ```
The cgroups-v1 filesystems that correspond to various cgroup-v1 controllers, were successfully mounted on the /sys/fs/cgroup/ directory.

2. Optionally, inspect the contents of the /sys/fs/cgroup/ directory:

```
# ll /sys/fs/cgroup/
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 blkio
lrwxrwxrwx.  1 root root 11 Mar 16 09:34 cpu -> cpu,cpuacct
lrwxrwxrwx.  1 root root 11 Mar 16 09:34 cpuacct -> cpu,cpuacct
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 cpu,cpuacct
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 cpuset
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 devices
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 freezer
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 hugetlb
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 memory
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 misc
lrwxrwxrwx.  1 root root 16 Mar 16 09:34 net_cls -> net_cls,net_prio
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 net_cls,net_prio
lrwxrwxrwx.  1 root root 16 Mar 16 09:34 net_prio -> net_cls,net_prio
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 perf_event
dr-xr-xr-x. 10 root root  0 Mar 16 09:34 pids
dr-xr-xr-x.  2 root root  0 Mar 16 09:34 rdma
dr-xr-xr-x. 11 root root  0 Mar 16 09:34 systemd
```

The /sys/fs/cgroup/ directory, also called the root control group, by default, contains controller-specific directories such as cpuset. In addition, there are some directories related to systemd.

Additional resources

- Understanding control groups
18.4. SETTING CPU LIMITS TO APPLICATIONS USING CGROUPS-V1

Sometimes an application consumes a lot of CPU time, which may negatively impact the overall health of your environment. Use the `/sys/fs/` virtual file system to configure CPU limits to an application using control groups version 1 (cgroups-v1).

**Prerequisites**

- You have root permissions.
- You have an application whose CPU consumption you want to restrict.
- You configured the system to mount cgroups-v1 by default during system boot by the `systemd` system and service manager:

```bash
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --
args="systemd.unified_cgroup_hierarchy=0
systemd.legacy_systemd_cgroup_controller"
```

This adds the necessary kernel command-line parameters to the current boot entry.

**Procedure**

1. Identify the process ID (PID) of the application you want to restrict in CPU consumption:

```bash
# top
```

The example output of the `top` program reveals that PID 6955 (illustrative application `sha1sum`) consumes a lot of CPU resources.

2. Create a sub-directory in the `cpu` resource controller directory:

```bash
# mkdir /sys/fs/cgroup/cpu/Example/
```
The directory above represents a control group, where you can place specific processes and apply certain CPU limits to the processes. At the same time, some `cgroups-v1` interface files and `cpu` controller-specific files will be created in the directory.

3. Optionally, inspect the newly created control group:

```
# ll /sys/fs/cgroup/cpu/Example/
-rw-r--r-- 1 root root 0 Mar 11 11:42 cgroup.clone_children
-rw-r--r-- 1 root root 0 Mar 11 11:42 cgroup.procs
-r--r--r-- 1 root root 0 Mar 11 11:42 cpuacct.stat
-rw-r--r-- 1 root root 0 Mar 11 11:42 cpuacct.usage
-r--r--r-- 1 root root 0 Mar 11 11:42 cpuacct.usage_all
-r--r--r-- 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu
-r--r--r-- 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu_sys
-r--r--r-- 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu_user
-r--r--r-- 1 root root 0 Mar 11 11:42 cpuacct.usage_sys
-r--r--r-- 1 root root 0 Mar 11 11:42 cpuacct.usage_user
-rw-r--r-- 1 root root 0 Mar 11 11:42 cpu.cfs_period_us
-rw-r--r-- 1 root root 0 Mar 11 11:42 cpu.cfs_quota_us
-rw-r--r-- 1 root root 0 Mar 11 11:42 cpu.rt_period_us
-rw-r--r-- 1 root root 0 Mar 11 11:42 cpu.rt_runtime_us
-rw-r--r-- 1 root root 0 Mar 11 11:42 cpu.shares
-r--r--r-- 1 root root 0 Mar 11 11:42 cpu.stat
-rw-r--r-- 1 root root 0 Mar 11 11:42 notify_on_release
-rw-r--r-- 1 root root 0 Mar 11 11:42 tasks
```

The example output shows files, such as `cpuacct.usage`, `cpu.cfs_period_us`, that represent specific configurations and/or limits, which can be set for processes in the Example control group. Notice that the respective file names are prefixed with the name of the control group controller to which they belong.

By default, the newly created control group inherits access to the system’s entire CPU resources without a limit.

4. Configure CPU limits for the control group:

```
# echo "1000000" > /sys/fs/cgroup/cpu/Example/cpu.cfs_period_us
# echo "200000" > /sys/fs/cgroup/cpu/Example/cpu.cfs_quota_us
```

The `cpu.cfs_period_us` file represents a period of time in microseconds (µs, represented here as "us") for how frequently a control group’s access to CPU resources should be reallocated. The upper limit is 1 second and the lower limit is 1000 microseconds.

The `cpu.cfs_quota_us` file represents the total amount of time in microseconds for which all processes collectively in a control group can run during one period (as defined by `cpu.cfs_period_us`). As soon as processes in a control group, during a single period, use up all the time specified by the quota, they are throttled for the remainder of the period and not allowed to run until the next period. The lower limit is 1000 microseconds.

The example commands above set the CPU time limits so that all processes collectively in the Example control group will be able to run only for 0.2 seconds (defined by `cpu.cfs_quota_us`) out of every 1 second (defined by `cpu.cfs_period_us`).

5. Optionally, verify the limits:

```
# cat /sys/fs/cgroup/cpu/Example/cpu.cfs_period_us
```
6. Add the application’s PID to the Example control group:

```
# echo "6955" > /sys/fs/cgroup/cpu/Example/cgroup.procs
```

or

```
# echo "6955" > /sys/fs/cgroup/cpu/Example/tasks
```

The previous command ensures that a desired application becomes a member of the Example control group and hence does not exceed the CPU limits configured for the Example control group. The PID should represent an existing process in the system. The PID 6955 here was assigned to process `sha1sum /dev/zero &`, used to illustrate the use-case of the cpu controller.

7. Verify that the application runs in the specified control group:

```
# cat /proc/6955/cgroup
12:cpuset:/
11:hugetlb:/
10:net_cls,net_prio:/
9:memory:/user.slice/user-1000.slice/user@1000.service
8:devices:/user.slice
7:blkio:/
6:freezer:/
5:rdma:/
4:pids:/user.slice/user-1000.slice/user@1000.service
3:perf_event:/
2:cpu,cpuacct:/Example
1:name=systemd:/user.slice/user-1000.slice/user@1000.service/gnome-terminal-server.service
```

The example output above shows that the process of the desired application runs in the Example control group, which applies CPU limits to the application’s process.

8. Identify the current CPU consumption of your throttled application:

```
# top
```

```
PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
6955 root      20   0  228440   1752   1472 R 20.6  0.1  47:11.43 sha1sum
5760 jdoe      20   0 3604956 208832  65316 R   2.3  11.2   0:43.50 gnome-shell
6448 jdoe      20   0  743836  31736  19488 S   0.7   1.7   0:08.25 gnome-terminal
505 root      20   0       0      0      0 I   0.3   0.0   0:03.39 kworker/u4:4-events_unbound
4217 root      20   0  74192  1612   1320 S   0.3   0.1   0:01.19 spice-vdagentd
...```

Notice that the CPU consumption of the PID 6955 has decreased from 99% to 20%.
IMPORTANT

The cgroups-v2 counterpart for cpu.cfs_period_us and cpu.cfs_quota_us is the cpu.max file. The cpu.max file is available through the cpu controller.

Additional resources

- Understanding control groups
- What kernel resource controllers are
- cgroups(7), sysfs(5) manual pages
CHAPTER 19. 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.

19.1. 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.

19.2. 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.

Procedure

1. Install bcc-tools:

   ```
   # dnf install bcc-tools
   ```

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

2. Optionally, inspect the tools:

   ```
   # ll /usr/share/bcc/tools/
   ...
   -rwxr-xr-x. 1 root root  4198 Dec 14 17:53 dcsnoop
   -rwxr-xr-x. 1 root root  3931 Dec 14 17:53 dcstat
   -rwxr-xr-x. 1 root root 20040 Dec 14 17:53 deadlock_detector
   -rw-r--r--. 1 root root  7105 Dec 14 17:53 deadlock_detector.c
   drwxr-xr-x. 3 root root  8192 Mar 11 10:28 doc
   -rwxr-xr-x. 1 root root  7588 Dec 14 17:53 execsnoop
   -rwxr-xr-x. 1 root root  6373 Dec 14 17:53 ext4dist
   -rwxr-xr-x. 1 root root 10401 Dec 14 17:53 ext4slower
   ...
   ```

   The doc directory in the listing above contains documentation for each tool.

19.3. 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-tools package
- 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/
   ...
   ```

   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:

```
<table>
<thead>
<tr>
<th>PID</th>
<th>COMM</th>
<th>D</th>
<th>MAJ</th>
<th>MIN</th>
<th>DISK</th>
<th>I/O</th>
<th>Kbytes</th>
<th>AVGms</th>
</tr>
</thead>
<tbody>
<tr>
<td>9568</td>
<td>dd</td>
<td>R</td>
<td>252</td>
<td>0</td>
<td>vda</td>
<td>16294</td>
<td>14440636.0</td>
<td>3.69</td>
</tr>
<tr>
<td>48</td>
<td>kswapd0</td>
<td>W</td>
<td>252</td>
<td>0</td>
<td>vda</td>
<td>1763</td>
<td>120696.0</td>
<td>1.65</td>
</tr>
<tr>
<td>7571</td>
<td>gnome-shell</td>
<td>R</td>
<td>252</td>
<td>0</td>
<td>vda</td>
<td>834</td>
<td>83612.0</td>
<td>0.33</td>
</tr>
<tr>
<td>1891</td>
<td>gnome-shell</td>
<td>R</td>
<td>252</td>
<td>0</td>
<td>vda</td>
<td>1379</td>
<td>19792.0</td>
<td>0.15</td>
</tr>
<tr>
<td>7515</td>
<td>Xorg</td>
<td>R</td>
<td>252</td>
<td>0</td>
<td>vda</td>
<td>280</td>
<td>9940.0</td>
<td>0.28</td>
</tr>
<tr>
<td>7579</td>
<td>llvmpipe-1</td>
<td>R</td>
<td>252</td>
<td>0</td>
<td>vda</td>
<td>228</td>
<td>6928.0</td>
<td>0.19</td>
</tr>
<tr>
<td>9515</td>
<td>gnome-control-c</td>
<td>R</td>
<td>252</td>
<td>0</td>
<td>vda</td>
<td>62</td>
<td>6444.0</td>
<td>0.43</td>
</tr>
<tr>
<td>8112</td>
<td>gnome-terminal</td>
<td>R</td>
<td>252</td>
<td>0</td>
<td>vda</td>
<td>67</td>
<td>2572.0</td>
<td>1.54</td>
</tr>
<tr>
<td>7807</td>
<td>gnome-software</td>
<td>R</td>
<td>252</td>
<td>0</td>
<td>vda</td>
<td>31</td>
<td>2336.0</td>
<td>0.73</td>
</tr>
<tr>
<td>9578</td>
<td>awk</td>
<td>R</td>
<td>252</td>
<td>0</td>
<td>vda</td>
<td>17</td>
<td>2228.0</td>
<td>0.66</td>
</tr>
<tr>
<td>7578</td>
<td>llvmpipe-0</td>
<td>R</td>
<td>252</td>
<td>0</td>
<td>vda</td>
<td>156</td>
<td>2204.0</td>
<td>0.07</td>
</tr>
<tr>
<td>9581</td>
<td>pgrep</td>
<td>R</td>
<td>252</td>
<td>0</td>
<td>vda</td>
<td>58</td>
<td>1748.0</td>
<td>0.42</td>
</tr>
<tr>
<td>7531</td>
<td>InputThread</td>
<td>R</td>
<td>252</td>
<td>0</td>
<td>vda</td>
<td>30</td>
<td>1200.0</td>
<td>0.48</td>
</tr>
<tr>
<td>7504</td>
<td>gdbus</td>
<td>R</td>
<td>252</td>
<td>0</td>
<td>vda</td>
<td>3</td>
<td>1164.0</td>
<td>0.30</td>
</tr>
<tr>
<td>1982</td>
<td>llvmpipe-0</td>
<td>R</td>
<td>252</td>
<td>0</td>
<td>vda</td>
<td>36</td>
<td>652.0</td>
<td>0.06</td>
</tr>
</tbody>
</table>

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:

```
# /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:

$ vim text

The command above creates a text file in the vim editor to initiate certain interaction with the XFS file system.

3. The terminal running xfsslower shows something similar upon saving the file from the previous step:

```
TIME  COMM           PID    T  BYTES   OFF_KB   LAT(ms)  FILENAME
13:07:14 b'bash'        4754   R 256     0           7.11 b'vim'
13:07:14 b'vim'         4754   R 832     0           4.03 b'libgpm.so.2.1.0'
13:07:14 b'vim'         4754   R 32      20          1.04 b'libgpm.so.2.1.0'
13:07:14 b'vim'         4754   R 1982    0           2.30 b'vimrc'
13:07:14 b'vim'         4754   R 1393    0           2.52 b'getscriptPlugin.vim'
13:07:45 b'vim'         4754   S 0       0           6.71 b'text'
13:07:45 b'pool'        2588   R 16      0           5.58 b'text'
...  
```

Each line above represents an operation in the file system, which took more time than a certain threshold. xfsslower is good at exposing possible file system problems, which can take form of unexpectedly slow operations.

The xfsslower output displays the following fields:

- **COMM** - The process name. (b'bash')
- **T** - The operation type. (R)
  - Read
  - Write
  - Sync
- **OFF_KB** - The file offset in KB. (0)
- **FILENAME** - The file being read, written, or synced.

To see more details, examples, and options for xfsslower, refer to the /usr/share/bcc/tools/doc/xfsslower_example.txt file.

For more information about fsync, see fsync(2) manual pages.
CHAPTER 20. CONFIGURING HUGE PAGES

Physical memory is managed in fixed-size chunks called pages. On the x86_64 architecture, supported by Red Hat Enterprise Linux 9, the default size of a memory page is 4 KB. This default page size has proved to be suitable for general-purpose operating systems, such as Red Hat Enterprise Linux, which supports many different kinds of workloads.

However, specific applications can benefit from using larger page sizes in certain cases. For example, an application that works with a large and relatively fixed data set of hundreds of megabytes or even dozens of gigabytes can have performance issues when using 4 KB pages. Such data sets can require a huge amount of 4 KB pages, which can lead to overhead in the operating system and the CPU.

This section provides information about huge pages available in RHEL 9 and how you can configure them.

20.1. AVAILABLE HUGE PAGE FEATURES

With Red Hat Enterprise Linux 9, you can use huge pages for applications that work with big data sets, and improve the performance of such applications.

The following are the huge page methods, which are supported in RHEL 9:

HugeTLB pages

HugeTLB pages are also called static huge pages. There are two ways of reserving HugeTLB pages:

- At boot time: It increases the possibility of success because the memory has not yet been significantly fragmented. However, on NUMA machines, the number of pages is automatically split among the NUMA nodes. For more information on parameters that influence HugeTLB page behavior at boot time, see Parameters for reserving HugeTLB pages at boot time and how to use these parameters to configure HugeTLB pages at boot time, see Configuring HugeTLB at boot time.

- At run time: It allows you to reserve the huge pages per NUMA node. If the run-time reservation is done as early as possible in the boot process, the probability of memory fragmentation is lower. For more information on parameters that influence HugeTLB page behavior at run time, see Parameters for reserving HugeTLB pages at run time and how to use these parameters to configure HugeTLB pages at run time, see Configuring HugeTLB at run time.

Transparent HugePages (THP)

With THP, the kernel automatically assigns huge pages to processes, and therefore there is no need to manually reserve the static huge pages. The following are the two modes of operation in THP:

- **system-wide**: Here, the kernel tries to assign huge pages to a process whenever it is possible to allocate the huge pages and the process is using a large contiguous virtual memory area.

- **per-process**: Here, the kernel only assigns huge pages to the memory areas of individual processes which you can specify using the **madvise**() system call.

**NOTE**

The THP feature only supports 2 MB pages.
For more information on parameters that influence HugeTLB page behavior at boot time, see Enabling transparent hugepages and Disabling transparent hugepages.

### 20.2. PARAMETERS FOR RESERVING HUGETLB PAGES AT BOOT TIME

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

For more information on how to use these parameters to configure HugeTLB pages at boot time, see Configuring HugeTLB at boot time.

<table>
<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>hugepagesz</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_hugepagesz</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>

### 20.3. CONFIGURING HUGETLB AT BOOT TIME

The page size, which the HugeTLB subsystem supports, depends on the architecture. The x86_64 architecture supports 2 MB huge pages and 1 GB gigantic pages.

This procedure describes how to reserve a 1 GB page at boot time.

**Procedure**

1. 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:
      
      ```bash
      # grub2-mkconfig -o /boot/grub2/grub.cfg
      ```
   
   b. If your system uses UEFI framework, execute the following command:
      
      ```bash
      # 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:

   ```ini
   [Unit]
   Description=HugeTLB Gigantic Pages Reservation
   DefaultDependencies=no
   Before=dev-hugepages.mount
   ConditionPathExists=/sys/devices/system/node
   ConditionKernelCommandLine=hugepagesz=1G

   [Service]
   Type=oneshot
   RemainAfterExit=yes
   ExecStart=/usr/lib/systemd/hugetlb-reserve-pages.sh

   [Install]
   WantedBy=sysinit.target
   ```

4. Create a new file called `hugetlb-reserve-pages.sh` in the `/usr/lib/systemd/` directory and add the following content:

   ```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 1 GB pages on node0 and one 1GB page on node1, 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
   reserve_pages 2 node0
   reserve_pages 1 node1
   ```
5. Create an executable script:
   
   ```
   # chmod +x /usr/lib/systemd/hugetlb-reserve-pages.sh
   ```

6. Enable early boot reservation:
   
   ```
   # 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 1 GB 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

- `systemd.service(5)` man page
- `/usr/share/doc/kernel-doc-kernel_version/Documentation/vm/hugetlbpage.txt` file

### 20.4. PARAMETERS FOR RESERVING HUGETLB PAGES AT RUN TIME

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

For more information on how to use these parameters to configure HugeTLB pages at run time, see Configuring HugeTLB at run time.

Table 20.2. 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><code>nr_hugepages</code></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>
### 20.5. CONFIGURING HUGETLB AT RUN TIME

This procedure describes how to add 20 2048 kB huge pages to node2.

To reserve pages based on your requirements, replace:

- `20` with the number of huge pages you wish to reserve,
- `2048kB` with the size of the huge pages,
- `node2` with the node on which you wish to reserve the pages.

**Procedure**

1. Display the memory statistics:

   ```
   # numastat -cm | egrep 'Node|Huge'
   
   Node 0 Node 1 Node 2 Node 3 Total add
   Anon HugePages     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
   
   # echo 20 > /sys/devices/system/node/node2/hugepages/hugepages-2048kB/nr_hugepages
   
   # numastat -cm | egrep 'Node|Huge'
   
   Node 0 Node 1 Node 2 Node 3 Total
   Anon HugePages     0  2  0  8  10
   
   # echo 20 > /sys/devices/system/node/node2/hugepages/hugepages-2048kB/nr_hugepages
   ```

2. Add the number of huge pages of a specified size to the node:

   ```
   # echo 20 > /sys/devices/system/node/node2/hugepages/hugepages-2048kB/nr_hugepages
   ```

**Verification steps**

- Ensure that the number of huge pages are added:

  ```
  # numastat -cm | egrep 'Node|Huge'
  
  Node 0 Node 1 Node 2 Node 3 Total
  Anon HugePages     0  2  0  8  10
  ```

---

<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>
CHAPTER 20. CONFIGURING HUGE PAGES

HugePages_Total  0  0  40  0  40
HugePages_Free   0  0  40  0  40
HugePages_Surp   0  0  0  0  0

Additional resources

- numastat(8) man page

20.6. ENABLING TRANSPARENT HUGEPAGES

THP is enabled by default in Red Hat Enterprise Linux 9. However, you can enable or disable THP.

This procedure describes how to enable THP.

Procedure

1. Check the current status of THP:

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

2. Enable THP:

   # echo always > /sys/kernel/mm/transparent_hugepage/enabled

3. To prevent applications from allocating more memory resources than necessary, disable the
   system-wide transparent huge pages and only enable them for the applications that explicitly
   request it through the madvise:

   # echo madvise > /sys/kernel/mm/transparent_hugepage/enabled

   NOTE

Sometimes, providing low latency to short-lived allocations has higher priority than immediately achieving the best performance with long-lived allocations. In such cases, you can disable direct compaction while leaving THP enabled.

Direct compaction is a synchronous memory compaction during the huge page allocation. Disabling direct compaction provides no guarantee of saving memory, but can decrease the risk of higher latencies during frequent page faults. Note that if the workload benefits significantly from THP, the performance decreases. Disable direct compaction:

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

Additional resources

- madvise(2) man page

Disabling transparent hugepages.

20.7. DISABLING TRANSPARENT HUGEPAGES

THP is enabled by default in Red Hat Enterprise Linux 9. However, you can enable or disable THP.
This procedure describes how to 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
   ```

20.8. 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 21. 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.

21.1. 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.

21.2. INSTALLING SYSTEMTAP

To begin using SystemTap, install the required packages. To use SystemTap on more than one kernel where a system has multiple kernels installed, install the corresponding required kernel packages for each kernel version.

**Prerequisites**
- You have enabled debug repositories as described in Enabling debug and source repositories.

**Procedure**

1. Install the required SystemTap packages:

   ```
   # dnf install systemtap
   ```

2. Install the required kernel packages:

   a. Using `stap-prep`:

   ```
   # stap-prep
   ```

   b. If `stap-prep` does not work, install the required kernel packages manually:
# dnf install kernel-debuginfo-$(uname -r) kernel-debuginfo-common-$(uname -i)-
$(uname -r) kernel-devel-$(uname -r)

$(uname -i) is automatically replaced with the hardware platform of your system and
$(uname -r) is automatically replaced with the version of your running kernel.

Verification steps

- If the kernel to be probed with SystemTap is currently in use, test if your installation was successful:

```
# stap -v -e 'probe kernel.function("vfs_read") {printf("read performed\n"); exit()}'
```

A successful SystemTap deployment results in an output similar to the following:

```
Pass 1: parsed user script and 45 library script(s) in 340usr/0sys/358real ms.
Pass 2: analyzed script: 1 probe(s), 1 function(s), 0 embed(s), 0 global(s) in 290usr/260sys/568real ms.
Pass 3: translated to C into
"/tmp/stapiArgLX/stap_e5886fa50499994e6a87aacdc43cd392_399.c" in 490usr/430sys/938real ms.
Pass 4: compiled C into "stap_e5886fa50499994e6a87aacdc43cd392_399.ko" in 3310usr/430sys/3714real ms.
Pass 5: starting run.  
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).

21.3. PRIVILEGES TO RUN SYSTEMTAP

Running SystemTap scripts requires elevated system privileges but, in some instances, non-privileged users might need to run SystemTap instrumentation on their machine.

To allow users to run SystemTap without root access, add users to both of these user groups:

**stapdev**

Members of this group can use **stap** to run SystemTap scripts, or **staprun** to run SystemTap instrumentation modules. Running **stap** involves compiling SystemTap scripts into kernel modules and loading them into the kernel. This requires elevated privileges to the system, which are granted to **stapdev** members. Unfortunately, such privileges also grant effective root access to **stapdev** members. As such, only grant **stapdev** group membership to users who can be trusted with root access.

**stapusr**
Members of this group can only use `staprun` to run SystemTap instrumentation modules. In addition, they can only run those modules from the `/lib/modules/kernel_version/systemtap` directory. This directory must be owned only by the root user, and must only be writable by the root user.

### 21.4. RUNNING SYSTEMTAP SCRIPTS

You can run SystemTap scripts from standard input or from a file.

Sample scripts that are distributed with the installation of SystemTap can be found in the `/usr/share/systemtap/examples` directory.

**Prerequisites**

1. SystemTap and the associated required kernel packages are installed as described in Installing Systemtap.

2. To run SystemTap scripts as a normal user, add the user to the SystemTap groups:

   ```bash
   # usermod --append --groups stapdev,stapusr user-name
   ```

**Procedure**

- Run the SystemTap script:
  - From standard input:
    ```bash
    # 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:
    ```bash
    # echo "probe timer.s(1) {exit()}" | stap -v -
    ```

  - From a file:
    ```bash
    # stap file_name
    ```
CHAPTER 22. CROSS-INSTRUMENTATION OF SYSTEMTAP

Cross-instrumentation of SystemTap is creating SystemTap instrumentation modules from a SystemTap script on one system to be used on another system that does not have SystemTap fully deployed.

22.1. 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.

    IMPORTANT

    Kernel packaging bugs may prevent the installation. In such cases, the kernel-debuginfo and kernel-devel packages for the host system and target system must match. If a bug occurs, report the bug at https://bugzilla.redhat.com/.

- Each target machine needs only one package to be installed to use the generated SystemTap instrumentation module: systemtap-runtime.

    IMPORTANT

    The host system must be the same architecture and running the same distribution of Linux as the target system in order for the built instrumentation module to work.
TERMINOLOGY

**instrumentation module**
The kernel module built from a SystemTap script; the SystemTap module is built on
the *host system*, and will be loaded on the *target kernel* of the *target system*.

**host system**
The system on which the instrumentation modules (from SystemTap scripts) are
compiled, to be loaded on *target systems*.

**target system**
The system in which the *instrumentation module* is being built (from SystemTap
scripts).

**target kernel**
The kernel of the *target system*. This is the kernel that loads and runs the
*instrumentation module*.

22.2. INITIALIZING CROSS-INSTRUMENTATION OF SYSTEMTAP

Initialize cross-instrumentation of SystemTap to build SystemTap instrumentation modules from a
SystemTap script on one system and use them on another system that does not have SystemTap fully
deployed.

Prerequisites

- SystemTap is installed on the *host system* as described in Installing Systemtap.
- The *systemtap-runtime* package is installed on each *target system*:

  ```
  # dnf install systemtap-runtime
  ```

- Both the *host system* and *target system* are the same architecture.
- Both the *host system* and *target system* are running the same major version of Red Hat
  Enterprise Linux (such as Red Hat Enterprise Linux 9).

IMPORTANT

Kernel packaging bugs may prevent multiple *kernel-debuginfo* and *kernel-devel*
packages from being installed on one system. In such cases, the minor version for the *host
system* and *target system* must match. If a bug occurs, report it at
https://bugzilla.redhat.com/.

Procedure

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

   ```
   $ uname -r
   ```

   Repeat this step for each *target system*.

2. On the *host system*, install the *target kernel* and related packages for each *target system* by the
   method described in Installing Systemtap.
3. Build an instrumentation module on the host system, copy this module to and run this module on 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 -p4 option tells SystemTap to not load and run the compiled module.

ii. Once the instrumentation module is compiled, copy it to the target system and load it using the following command:

```bash
# staprun module_name.ko
```
You can use helpful example SystemTap scripts available in the `/usr/share/systemtap/testsuite/systemtap.examples/` directory, upon installing the `systemtap-testsuite` package, to monitor and investigate the network activity of your system.

### 23.1. PROFILING NETWORK ACTIVITY WITH SYSTEMTAP

You can use the `nettop.stp` example SystemTap script to profile network activity. The script tracks which processes are generating network traffic on the system, and provides the following information about each process:

- **PID**
  - The ID of the listed process.
- **UID**
  - User ID. A user ID of 0 refers to the root user.
- **DEV**
  - Which ethernet device the process used to send or receive data (for example, eth0, eth1).
- **XMIT_PK**
  - The number of packets transmitted by the process.
- **RECV_PK**
  - The number of packets received by the process.
- **XMIT_KB**
  - The amount of data sent by the process, in kilobytes.
- **RECV_KB**
  - The amount of data received by the service, in kilobytes.

**Prerequisites**

- You have installed SystemTap as described in [Installing SystemTap](#).

**Procedure**

- Run the `nettop.stp` script:
  ```bash
  # stap --example nettop.stp
  ```

  The `nettop.stp` script provides network profile sampling every 5 seconds.

  Output of the `nettop.stp` script looks similar to the following:

  ```
  [...] , UID DEV  XMIT_PK RECV_PK XMIT_KB RECV_KB COMMAND
  0      0  eth0   0       5       0       0  swapper
  11178  0  eth0   2       0       0       0  synergyc
  2886   4  eth0   79      0       5       0  cups-polld
  11362  0  eth0   0       61      0       0  firefox
  ```
### 23.2. TRACING FUNCTIONS CALLED IN NETWORK SOCKET CODE WITH SYSTEMTAP

You can use the `socket-trace.stp` example SystemTap script to trace functions called from the kernel’s `net/socket.c` file. This helps you identify, in finer detail, how each process interacts with the network at the kernel level.

**Prerequisites**

- You have installed SystemTap as described in [Installing SystemTap](#).

**Procedure**

- Run the `socket-trace.stp` script:

  ```
  # stap --example socket-trace.stp
  ```

  A 3-second excerpt of the output of the `socket-trace.stp` script looks similar to the following:

  ```
  [...]  
  0 Xorg(3611): -> sock_poll  
  3 Xorg(3611): <- sock_poll  
  0 Xorg(3611): -> sock_poll  
  3 Xorg(3611): <- sock_poll  
  0 gnome-terminal(11106): -> sock_poll  
  5 gnome-terminal(11106): <- sock_poll  
  0 scim-bridge(3883): -> sock_poll  
  3 scim-bridge(3883): <- sock_poll  
  0 scim-bridge(3883): -> sys_socketcall  
  4 scim-bridge(3883): -> sys_recv  
  8 scim-bridge(3883): -> sys_recvfrom  
  12 scim-bridge(3883): -> sock_from_file  
  16 scim-bridge(3883): <- sock_from_file  
  20 scim-bridge(3883): -> sock_recvmsg  
  24 scim-bridge(3883): <- sock_recvmsg  
  28 scim-bridge(3883): <- sys_recvfrom  
  31 scim-bridge(3883): <- sys_recv  
  35 scim-bridge(3883): <- sys_socketcall  
  [...]  
  ```
23.3. MONITORING NETWORK PACKET DROPS WITH SYSTEMTAP

The network stack in Linux can discard packets for various reasons. Some Linux kernels include a tracepoint, `kernel.trace("kfree_skb")`, which tracks where packets are discarded.

The `dropwatch.stp` SystemTap script uses `kernel.trace("kfree_skb")` to trace packet discards; the script summarizes what locations discard packets in every 5-second interval.

Prerequisites

- You have installed SystemTap as described in Installing SystemTap.

Procedure

- Run the `dropwatch.stp` script:
  
  ```
  # stap --example dropwatch.stp
  ```

Running the `dropwatch.stp` script for 15 seconds results in output similar to the following:

```
Monitoring for dropped packets
51 packets dropped at location 0xffffffff8024cd0f
2 packets dropped at location 0xffffffff8044b472
51 packets dropped at location 0xffffffff8024cd0f
1 packets dropped at location 0xffffffff8044b472
97 packets dropped at location 0xffffffff8024cd0f
1 packets dropped at location 0xffffffff8044b472
Stopping dropped packet monitor
```

NOTE

To make the location of packet drops more meaningful, see the `/boot/System.map-$(uname -r)` file. This file lists the starting addresses for each function, enabling you to map the addresses in the output of the `dropwatch.stp` script to a specific function name. Given the following snippet of the `/boot/System.map-$(uname -r)` file, the address `0xffffffff8024cd0f` maps to the function `unix_stream_recvmsg` and the address `0xffffffff8044b472` maps to the function `arp_rcv`:

```
[...]
ffffffff8024c5cd T unlock_new_inode
ffffffff8024c5da t unix_stream_sendmsg
ffffffff8024c920 t unix_stream_recvmsg
ffffffff8024cea1 t udp_v4_lookup_longway
[...]
ffffffff8044addc t arp_process
ffffffff8044b360 t arp_rcv
ffffffff8044b487 t parp_redo
ffffffff8044b48c t arp_solicit
[...]```
CHAPTER 24. PROFILING KERNEL ACTIVITY WITH SYSTEMTAP

The following sections showcase scripts that profile kernel activity by monitoring function calls.

24.1. COUNTING FUNCTION CALLS WITH SYSTEMTAP

You can use the `functioncallcount.stp` SystemTap script to count specific kernel function calls. You can also use this script to target multiple kernel functions.

Prerequisites

- You have installed SystemTap as described in Installing Systemtap.

Procedure

- Run the `functioncallcount.stp` script:

  ```
  # stap --example functioncallcount.stp 'argument'
  ```

  This script takes the targeted kernel function as an argument. You can use the argument wildcards to target multiple kernel functions up to a certain extent.

  The output of the script, in alphabetical order, contains the names of the functions called and how many times it was called during the sample time.

  Consider the following example:

  ```
  # stap -w -v --example functioncallcount.stp "*@mm*.c" -c /bin/true
  ```

  where:

  - `-w`: Suppresses warnings.
  - `-v`: Makes the output of starting kernel visible.
  - `-c command`: Tells SystemTap to count function calls during the execution of a command, in this example being `/bin/true`.

  The output should look similar to the following:

  ```
  [...]
  __vma_link 97
  __vma_link_file 66
  __vma_link_list 97
  __vma_link_rb 97
  __xchg 103
  add_page_to_active_list 102
  add_page_to_inactive_list 19
  add_to_page_cache 19
  add_to_page_cache_lru 7
  all_vm_events 6
  alloc_pages_node 4630
  alloc_slabmgmt 67
  ```
24.2. TRACING FUNCTION CALLS WITH SYSTEMTAP

You can use the `para-callgraph.stp` SystemTap script to trace function calls and function returns.

**Prerequisites**

- You have installed SystemTap as described in [Installing Systemtap](#).

**Procedure**

- Run the `para-callgraph.stp` script.

```
# stap --example para-callgraph.stp 'argument1' 'argument2'
```

The script `para-callgraph.stp` takes two command-line arguments:

1. The name of the function(s) whose entry/exit you’d like to trace.
2. An optional trigger function, which enables or disables tracing on a per-thread basis. Tracing in each thread will continue as long as the trigger function has not exited yet.

Consider the following example:

```
# stap -wv --example para-callgraph.stp 'kernel.function("@fs/proc.c")' 'kernel.function("vfs_read")' -c "cat /proc/sys/vm/" || true"
```

where:

- `-w` : Suppresses warnings.
- `-v` : Makes the output of starting kernel visible.
- `-c command` : Tells SystemTap to count function calls during the execution of a command, in this example being `/bin/true`.

The output should look similar to the following:

```
[...]
```
24.3. DETERMINING TIME SPENT IN KERNEL AND USER SPACE WITH SYSTEMTAP

You can use the thread-times.stp SystemTap script to determine the amount of time any given thread is spending in either the kernel or user-space.

Prerequisites

- You have installed SystemTap as described in Installing Systemtap.

Procedure

- Run the thread-times.stp script:

```
# stap --example thread-times.stp
```

This script will display the top 20 processes taking up CPU time during a 5-second period, along with the total number of CPU ticks made during the sample. The output of this script also notes the percentage of CPU time each process used, as well as whether that time was spent in kernel space or user space.

```
tid  %user %kernel (of 20002 ticks)
   0 0.00%  87.88%
  32169 5.24%  0.03%
   9815 3.33%  0.36%
   9859 0.95%  0.00%
   3611 0.56%  0.12%
   9861 0.62%  0.01%
  11106 0.37%  0.02%
  32167 0.08%  0.08%
   3897 0.01%  0.08%
   3800 0.03%  0.00%
   2886 0.02%  0.00%
   3243 0.00%  0.01%
   3862 0.01%  0.00%
   3782 0.00%  0.00%
  21767 0.00%  0.00%
```
24.4. MONITORING POLLING APPLICATIONS WITH SYSTMEMTAP

You can use **timeout.stp** SystemTap script to identify and monitor which applications are polling. Doing so allows you to track unnecessary or excessive polling, which helps you pinpoint areas for improvement in terms of CPU usage and power savings.

**Prerequisites**

- You have installed SystemTap as described in **Installing Systemtap**.

**Procedure**

- Run the **timeout.stp** script:

```bash
# stap --example timeout.stp
```

This script will track how many times each application uses the following system calls over time:

- **poll**
- **select**
- **epoll**
- **itimer**
- **futex**
- **nanosleep**
- **signal**

In this example output you can see which process used which system call and how many times.

```
uid | poll  select  epoll  itimer  futex  nanosleep  signal | process
28937 | 148793 0 0 4727 37288 0 0 | firefox
22945 | 0 56949 0 1 0 0 0 | scim-bridge
22941 | 7908 1 0 62 0 0 0 | gnome-terminal
22941 | 0 14405 0 0 0 0 0 | scim-launcher
4275 | 23140 0 0 1 0 0 0 | mixer_applet2
4191 | 7267 0 0 0 0 0 0 | dhcdbus
3695 | 5767 0 0 0 0 0 0 | gdm-binary
1863 | 5767 0 0 0 0 0 0 | iscsid
```
24.5. TRACKING MOST FREQUENTLY USED SYSTEM CALLS WITH SYSTEMTAP

You can use the `topsys.stp` SystemTap script to list the top 20 system calls used by the system per 5-second interval. It also lists how many times each system call was used during that period.

**Prerequisites**

- You have installed SystemTap as described in Installing Systemtap.

**Procedure**

- Run the `topsys.stp` script:
  
  ```
  # stap --example topsys.stp
  ```

  Consider the following example:

  ```
  # stap -v --example topsys.stp
  ```

  where `-v` makes the output of starting kernel visible.

  The output should look similar to the following:

<table>
<thead>
<tr>
<th>SYSCALL</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>gettimeofday</td>
<td>1857</td>
</tr>
<tr>
<td>read</td>
<td>1821</td>
</tr>
<tr>
<td>ioctl</td>
<td>1568</td>
</tr>
<tr>
<td>poll</td>
<td>1033</td>
</tr>
<tr>
<td>close</td>
<td>638</td>
</tr>
<tr>
<td>open</td>
<td>503</td>
</tr>
<tr>
<td>select</td>
<td>455</td>
</tr>
<tr>
<td>write</td>
<td>391</td>
</tr>
<tr>
<td>writev</td>
<td>335</td>
</tr>
<tr>
<td>futex</td>
<td>303</td>
</tr>
<tr>
<td>recvmsg</td>
<td>251</td>
</tr>
<tr>
<td>socket</td>
<td>137</td>
</tr>
<tr>
<td>clock_gettime</td>
<td>124</td>
</tr>
<tr>
<td>rt_sigprocmask</td>
<td>121</td>
</tr>
<tr>
<td>sendto</td>
<td>120</td>
</tr>
<tr>
<td>setitimer</td>
<td>106</td>
</tr>
<tr>
<td>stat</td>
<td>90</td>
</tr>
<tr>
<td>time</td>
<td>81</td>
</tr>
<tr>
<td>sigreturn</td>
<td>72</td>
</tr>
<tr>
<td>fstat</td>
<td>66</td>
</tr>
</tbody>
</table>

24.6. TRACKING SYSTEM CALL VOLUME PER PROCESS WITH SYSTEMTAP

You can use the `syscalls_by_proc.stp` SystemTap script to see which processes are performing the highest volume of system calls. It displays 20 processes performing the most of system calls.
Prerequisites

- You have installed SystemTap as described in Installing Systemtap.

Procedure

- Run the `syscalls_by_proc.stp` script:

```
# stap --example syscalls_by_proc.stp
```

Output of the `syscalls_by_proc.stp` script looks similar to the following:

```
Collecting data... Type Ctrl-C to exit and display results
#SysCalls Process Name
1577   multiload-apple
692    synergyc
408    pcscd
376    mixer_applet2
299    gnome-terminal
293    Xorg
206    scim-panel-gtk
95     gnome-power-man
90     arts
85     dhcddb
84     scim-bridge
78     gnome-screensav
66     scim-launcher
[...]```
CHAPTER 25. MONITORING DISK AND I/O ACTIVITY WITH SYSTEMTAP

The following sections showcase scripts that monitor disk and I/O activity.

25.1. SUMMARIZING DISK READ/WRITE TRAFFIC WITH SYSTEMTAP

You can use the `disktop.stp` SystemTap script to identify which processes are performing the heaviest disk reads and writes to the system.

Prerequisites

- You have installed SystemTap as described in Installing Systemtap.

Procedure

- Run the `disktop.stp` script:

```
# stap --example disktop.stp
```

The script displays the top ten processes responsible for the heaviest reads or writes to a disk.

The output includes the following data per listed process:

- **UID**: User ID. A user ID of 0 refers to the root user.
- **PID**: The ID of the listed process.
- **PPID**: The process ID of the listed process’s parent process.
- **CMD**: The name of the listed process.
- **DEVICE**: Which storage device the listed process is reading from or writing to.
- **T**: The type of action performed by the listed process, where W refers to write, and R refers to read.
- **BYTES**: The amount of data read to or written from disk.

Output of the `disktop.stp` script looks similar to the following:

```
[...]
Mon Sep 29 03:38:28 2008 , Average: 19Kb/sec, Read: 7Kb, Write: 89Kb
UID   PID   PPID              CMD     DEVICE    T    BYTES
0    26319  26294              firefox  sda5    W    90229
0    2758   2757              pam_timestamp_c  sda5    R    8064
0    2885   1                 cupsd    sda5    W    1678
Mon Sep 29 03:38:38 2008 , Average: 1Kb/sec, Read: 7Kb, Write: 1Kb
```
25.2. TRACKING I/O TIME FOR EACH FILE READ OR WRITE WITH SYSTEMTAP

You can use the iotime.stp SystemTap script to monitor the amount of time it takes for each process to read from or write to any file. This helps you to determine what files are slow to load on a system.

Prerequisites

- You have installed SystemTap as described in Installing Systemtap.

Procedure

- Run the iotime.stp script:

```
# stap --example iotime.stp
```

The script tracks each time a system call opens, closes, reads from, and writes to a file. For each file any system call accesses, it counts the number of microseconds it takes for any reads or writes to finish and tracks the amount of data, in bytes, read from or written to the file.

The output contains:

- A timestamp, in microseconds
- Process ID and process name
- An access or iotime flag
- The file accessed

If a process was able to read or write any data, a pair of access and iotime lines should appear together. The access line refers to the time that a given process started accessing a file. The end of the access line will show the amount of data read or written. The iotime line will show the amount of time, in microseconds, that the process took in order to perform the read or write.

Output of the iotime.stp script looks similar to the following:

```
[...]
825946 3364 (NetworkManager) access /sys/class/net/eth0/carrier read: 8190 write: 0
825955 3364 (NetworkManager) iotime /sys/class/net/eth0/carrier time: 9
[...]
117061 2460 (pcscd) access /dev/bus/usb/003/001 read: 43 write: 0
117065 2460 (pcscd) iotime /dev/bus/usb/003/001 time: 7
[...]
3973737 2886 (sendmail) access /proc/loadavg read: 4096 write: 0
3973744 2886 (sendmail) iotime /proc/loadavg time: 11
[...]
```

25.3. TRACKING CUMULATIVE I/O WITH SYSTEMTAP
You can use the `traceio.stp` SystemTap script to track the cumulative amount of I/O to the system.

**Prerequisites**

- You have installed SystemTap as described in [Installing Systemtap](#).

**Procedure**

- Run the `traceio.stp` script:

  ```bash
  # stap --example traceio.stp
  ```

  The script prints the top ten executables generating I/O traffic over time. It also tracks the cumulative amount of I/O reads and writes done by those executables. This information is tracked and printed out in 1-second intervals, and in descending order.

  Output of the `traceio.stp` script looks similar to the following:

  ```plaintext
  [...]  
  Xorg r:  583401 KiB w:   0 KiB  
  floaters r:  96 KiB w:  7130 KiB  
  multiload-apple r:  538 KiB w:  537 KiB  
  sshd r:  71 KiB w:  72 KiB  
  pam_timestamp_c r:  138 KiB w:   0 KiB  
  staprun r:  51 KiB w:  51 KiB  
  snmpd r:  46 KiB w:   0 KiB  
  pcscd r:  28 KiB w:   0 KiB  
  irqbalance r:  27 KiB w:   4 KiB  
  cupsd r:  4 KiB w:  18 KiB  
  Xorg r:  588140 KiB w:  0 KiB  
  floaters r:  97 KiB w:  7143 KiB  
  multiload-apple r:  543 KiB w:  542 KiB  
  sshd r:  72 KiB w:  72 KiB  
  pam_timestamp_c r:  138 KiB w:   0 KiB  
  staprun r:  51 KiB w:  51 KiB  
  snmpd r:  46 KiB w:   0 KiB  
  pcscd r:  28 KiB w:   0 KiB  
  irqbalance r:  27 KiB w:   4 KiB  
  cupsd r:  4 KiB w:  18 KiB  
  ```

**25.4. MONITORING I/O ACTIVITY ON A SPECIFIC DEVICE WITH SYSTEMTAP**

You can use the `traceio2.stp` SystemTap script to monitor I/O activity on a specific device.

**Prerequisites**

- You have installed SystemTap as described in [Installing Systemtap](#).

**Procedure**

- Run the `traceio2.stp` script.
This script takes the whole device number as an argument. To find this number you can use:

```
# stat -c "0x%D" directory
```

Where `directory` is located on the device you want to monitor.

The output contains following:

- The name and ID of any process performing a read or write
- The function it is performing (`vfs_read` or `vfs_write`)
- The kernel device number

Consider following output of `# stap traceio2.stp 0x805`

```
[...]
synergyc(3722) vfs_read 0x800005
synergyc(3722) vfs_read 0x800005
cupsd(2889) vfs_write 0x800005
cupsd(2889) vfs_write 0x800005
cupsd(2889) vfs_write 0x800005
[...]
```

**25.5. MONITORING READS AND WRITES TO A FILE WITH SYSTEMTAP**

You can use the `inodewatch.stp` SystemTap script to monitor reads from and writes to a file in real time.

**Prerequisites**

- You have installed SystemTap as described in *Installing Systemtap*.

**Procedure**

- Run the `inodewatch.stp` script.

```
# stap --example inodewatch.stp 'argument1' 'argument2' 'argument3'
```

The script `inodewatch.stp` takes three command-line arguments:

1. The file’s major device number.
2. The file’s minor device number.
3. The file’s inode number.

You can get these numbers using:

```
# stat -c '%D %i' filename
```
Where filename is an absolute path.

Consider following example:

```
# stat -c '%D %i' /etc/crontab
```

The output should look like:

```
805 1078319
```

where:

- **805** is the base-16 (hexadecimal) device number. The last two digits are the minor device number, and the remaining digits are the major number.
- **1078319** is the inode number.

To start monitoring /etc/crontab, run:

```
# stap inodewatch.stp 0x8 0x05 1078319
```

In the first two arguments you must use 0x prefixes for base-16 numbers.

The output contains following:

- The name and ID of any process performing a read or write
- The function it is performing (**vfs_read** or **vfs_write**)
- The kernel device number

The output of this example should look like:

```
cat(16437) vfs_read 0x800005/1078319
cat(16437) vfs_read 0x800005/1078319
```