1.4

REDHAWK *KVM-RT ™ User's Guide*

concurrent REAL-TIME

0898604-1.4
March 2023

Revision History:	Level:	Effective With:
July 2019	1.0	RedHawk Linux 7.5
January 2020	1.1	RedHawk Linux 8.0
February 2021	1.2	RedHawk Linux 8.2
October 2021	1.3	RedHawk Linux 8.4
March 2023	1.4	RedHawk Linux 8.4

Preface

Scope of Manual

This manual provides information and instructions for using Concurrent Real-Time's RedHawk KVM-RT™.

Structure of Manual

This manual consists of:

- Chapter 1 introduces you to KVM-RT.
- Chapter 2 explains the steps in setting up and booting virtual machines in KVM-RT.
- Chapter 3 covers how to configure KVM-RT.
- Chapter 4 summarizes all the KVM-RT tools.
- Chapter 5 discusses time synchronization.
- Chapter 6 discusses ways to analyze and debug guest VMs in KVM-RT.

Syntax Notation

The following notation is used throughout this manual:

italic	Books, reference cards, and items that the user must specify appear in *italic* type. Special terms may also appear in *italic*.
`list bold`	User input appears in **`list bold`** type and must be entered exactly as shown. Names of directories, files, commands, options and man page references also appear in **`list bold`** type.
`list`	Operating system and program output such as prompts, messages and listings of files and programs appears in `list` type.
`[]`	Brackets enclose command options and arguments that are optional. You do not type the brackets if you choose to specify these options or arguments.
hypertext links	When viewing this document online, clicking on chapter, section, figure, table and page number references will display the corresponding text. Clicking on Internet URLs provided in blue type will launch your web browser and display the web site. Clicking on publication names and numbers in red type will display the corresponding manual PDF, if accessible.

Related Publications

The following table lists Concurrent Real-Time documentation. Depending upon the document, they are available online on RedHawk Linux systems or from Concurrent Real-Time's documentation web site at **http://redhawk.concurrent-rt.com/docs**.

RedHawk KVM-RT	**Pub. Number**
RedHawk KVM-RT Release Notes	0898603
RedHawk KVM-RT User's Guide	0898604
RedHawk Architect	
RedHawk Architect Release Notes	0898600
RedHawk Architect User's Guide	0898601
RedHawk Linux	
RedHawk Linux Release Notes	0898003
RedHawk Linux User's Guide	0898004
RedHawk Linux Cluster Manager User's Guide	0898016
RedHawk Linux FAQ	N/A
NightStar RT Development Tools	
NightView User's Guide	0898395
NightTrace User's Guide	0898398
NightProbe User's Guide	0898465
NightTune User's Guide	0898515

Contents

1
Introduction to KVM-RT

This chapter provides a general overview and requirements for using RedHawk KVM-RT.

Introduction

RedHawk KVM-RT is a Real-Time Hypervisor solution that utilizes QEMU/KVM and RedHawk real-time features to extend RedHawk's real-time determinism to guest RedHawk virtual machines.

It supports multiple guests, both real-time and non-real-time, running in virtual machines on a single host system.

Host System Requirements and Installation

Refer to the *RedHawk KVM-RT Release Notes* for hardware host system requirements and software installation instructions.

Though not a requirement, it is highly recommended that the entire host system be dedicated to running the Real-Time Hypervisor. Administrators of the KVM-RT host system must be careful not to disturb CPU shielding or CPU affinities on the system, or else real-time performance of virtual machines may be compromised.

Once KVM-RT is installed, the following command can be run to test the suitability of the host system.

```
$ sudo kvmrt-validate-host
```

Host Kernel Configuration

KVM-RT requires that a RedHawk kernel is booted on the host system while KVM-RT is being used. Additional system configuration may be required.

Kernel Boot Parameters

The parameters listed in this section may be added to the RedHawk system's boot time parameters with the command **`blscfg(1)`** in RedHawk Linux versions 8.0 and later, and **`ccur-grub2(1)`** in Ubuntu releases and RedHawk Linux versions 7.x. Note that a

reboot is necessary for the parameters to take effect and, as noted, some parameters are not available in all releases.

Enabling PCI Passthrough depends on the system architecture. AMD systems do not require PCI passthrough to be enabled in the kernel while Intel systems running RedHawk releases 7.5 and 8.x require that the following boot parameter be set:

intel_iommu = on

This parameter enables device-level remapping of memory regions used for DMA in virtual machines. This parameter is required if any virtual machines will use PCI passthrough of physical PCI devices.

Most PCI-e (PCI Express) cards can be used for PCI passthrough without requiring any additional kernel boot parameter to be set. Graphic cards are the exception. The graphics card must be claimed by the VFIO driver early in the boot before it is claimed by the host's drivers.

The following boot parameters may be used to passthrough a graphics card. Note that **`lspci -nnk`** command can be used to obtain the necessary information about the device. The PCI vendor and device id codes (vendor:device) are listed at the end of the line in brackets ([]). The BUS:SLOT.FUNCTION information is listed at the very beginning of the line. If more than one device is listed for your graphics card, you must include all the devices.

vfio-pci.ids = [vendor:device,...]

This parameter can be set to a comma-separated list of pci devices that will be assigned to the VFIO driver. Each device is specified by vendor:device.

vfio-pci.addrs = [BUS:SLOT.FUNCTION,…]

This parameter is set to a comma-separated list of pci devices that will be assigned to the VFIO driver. Each device is specified by BUS:SLOT.FUNCTION.

This boot parameter should be used when you have multiple cards on the system with the same vendor and device IDs and the host wants to be able to use one or more of the cards. If you use the *vfio-pci.ids* boot parameter the host will not be able to properly initialize and use any of the devices.

Note that if you change the physical placement of cards in the system, the BUS:SLOT.FUNCTION setting will need to be re-evaluated as it may change. If it has changed, you must also change the kernel boot parameter setting.

This option is supported in RedHawk releases 8.0 and later.

These parameters are also documented in the file **`kernel-parameters.txt`** under the **`/usr/src/linux-<`***`kernel-name`***`>/Documentation`** in 7.x releases and under a subdirectory **`admin-guide`** under that path in the 8.x releases.

Migrating Managed IRQs

Per CPU interrupts can be classified as managed interrupts. Most modern NIC, RAID, and NVME devices generate managed interrupts. In the RedHawk release version 8.2 changes

were made to allow managed interrupts to migrate to another CPU. This change has been backported to releases 7.5 and later. If you have the latest release updates, you will have this change. Note that managed interrupts did not exist in releases prior to 7.5.

Migration of managed interrupts is necessary in KVM-RT because they can impact the real-time performance of VMs. Also KVM-RT attempts to take down hyperthreaded CPUs. CPUS cannot be taken down if there are IRQs still associated with a CPU. The goal is to migrate all IRQs away from CPUs responsible for vCPUs and move them to CPUs responsible for emulation and VirtIO operations.

The KVM-RT tools, **`irq-affinity`** and **`task-affinity`**, display the CPU affinities of IRQs and tasks respectively and are very useful in finding IRQs and tasks bound to specific CPUs. Use the **`--help`** option for more information and usage of these commands.

The **`shield(1)`** command cannot be used to migrate managed interrupts as they only have one CPU in their cpu affinity mask. Below are some of the ways to migrate managed IRQS off CPUs.

1. Use the **`--set`** option of the **`irq-affinity`** command to migrate managed IRQs from one set of CPUs to a different set of CPUs.

2. Setting the kernel boot parameter *`msi_affinity_mask`* will set the affinity mask for all MSI(X) managed interrupts at boot time. Note that currently the parameter *`irqaffinity`* must also be set to the same value, but in future releases only *`msi_affinity_mask`* will need to be set. Also note that this feature is only available in RedHawk 8.x releases.

   `msi_affinity_mask =` [cpulist]
   `irqaffinity =` [cpulist]

   cpulist must be set to a list of CPUs that must include CPU 0. The list can include a range i.e. 0-5, or a comma separated list i.e. 0,3,4,5.

3. The **`systemd shield`** service may be used to set shielding attributes for selected CPUs. Modifications are made to the file:
   `/etc/sysconfig/shield`.

   For example, you can assign the enp4s0f0 interrupt to CPUs 0 through 4 or assign interrupts number 55, 60 and 61 to cpus 0 and 2 by adding these lines to the **`shield`** service configuration file.

   IRQ_ASSIGN+="0-4:enp4s0f0;"
   IRQ_ASSIGN+="0,2:55; 0,2:60; 0,2:61;"

   After editing the file you can restart the service with the command:
   `systemctl restart shield`

   And you can check the status of the command with:
   `systemctl status shield`

2
Getting Started

This chapter explains the steps in setting up and booting virtual machines in KVM-RT. Also discussed are the various QEMU/KVM threads that run on the host for each virtual machine.

Building Virtual Machines

KVM-RT works with virtual machines that have been created and configured within the libvirt framework. A virtual machine may be created and configured within libvirt in several ways, including:

- with Virtual Machine Manager
- with RedHawk Architect
- by cloning another virtual machine

Detailed instructions on how to build virtual machines are beyond the scope of this book but are well documented. General instructions and references to documentation are given in the following sections.

Real-time virtual machines must contain a guest OS of RedHawk Linux 7.0 or later. The guest CPU architecture must match that of the host.

UEFI firmware in guest VMs is required for PCI passthrough of NVIDIA graphics cards. Note that Red Hat 7.x releases do not provide OVMF UEFI firmware for VMs.

Using Virtual Machine Manager to Create a Virtual Machine

The Virtual Machine Manager is a GUI tool that can be used to create, configure, and manage virtual machines within the libvirt framework.

Start Virtual Machine Manager by running:

```
$ sudo run virt-manager
```

See the **virt-manager(1)** man page for more information.

Using RedHawk Architect to Create a Virtual Machine

RedHawk Architect is an optional product offered by Concurrent Real-Time that specializes in creating, customizing and deploying RedHawk Linux disk images.

Architect can be used to create a RedHawk virtual machine and to export it to the Virtual Machine Manager. Detailed instructions can be found in the documentation that comes with RedHawk Architect. Below are the general steps required:

- run Architect
- create a new session and configure the image as desired
- build the image
- deploy the image to a virtual machine
- export the virtual machine to Virtual Machine Manager

Cloning a Virtual Machine Image

Any existing virtual machine within the libvirt framework can be cloned by using the virt-clone command. For example:

```
$ sudo virt-clone -o old_vm -n new_vm
```

See the **virt-clone(1)** man page for more information.

Importing Virtual Machines into KVM-RT

Once virtual machines have been created within the libvirt framework, they can be imported into KVM-RT.

All libvirt virtual machines can be imported into KVM-RT with the following command:

```
$ sudo kvmrt-import
```

This command may be run at any time new VMs are created. Run **kvmrt-import --help** for more information and options.

When a VM is imported into KVM-RT it inherits the VM configuration settings from libvirt. Once this is done a VM may be further configured with KVM-RT as needed. See "Configuring Virtual Machines" in Chapter 3 for more information.

Booting and Shutting Down Virtual Machines

A **systemd** service, named *kvmrt*, exists for KVM-RT. It may be enabled so that the VMs configured with autostart set, will be automatically booted during system start-up and VMs running when the system is shutting down will be shut down. The service is not enabled by default. Once enabled, the service will automatically start on the next boot. If you want it to start it immediately, you must enable the service and start it as follows:

```
systemctl enable kvmrt
```

```
systemctl start kvmrt
```

Note that the service start will fail if there are any VMs running since it invokes **kmvrt-boot** with the **--clean** option.

The following KVM-RT tools can be used to boot, shutdown, and view the status of VMs.

To start up all configured VMs:

```
$ sudo kvmrt-boot
```

To shut down all the running VMs:

```
$ sudo kvmrt-shutdown
```

To query the state of all VMs:

```
$ sudo kvmrt-stat
```

Individual VMs can be specified to all these commands. For example:

```
$ sudo kvmrt-boot RedHawk-8.4VM Windows10VM

$ sudo kvmrt-shutdown RedHawk-8.4VM Windows10VM
```

Note that by default VMs are brought down in parallel. If the **-v** (verbose) option is used, the shutdown will be serialized so that the output from the different VMs is not garbled together.

Run any of the above commands with the **--help** option for more information and options.

Understanding QEMU/KVM Threads

QEMU/KVM runs multiple threads for each virtual machine. The names and purpose of these threads are as follows:

qemu-kvm

These are emulator threads. There may be two or more of these.

qemu-system-x86

This is an alternate name for *qemu-kvm* in some distributions.

worker

These are dynamically created threads for long I/O operations being performed by the emulator.

SPICE Worker

This is a thread for a virtual console.

IO mon_ioth

This is an optional thread used for some I/O.

CPU n/KVM

These are virtual CPU (vCPU) threads. There will be one per virtual CPU, where *n* is the vCPU ID.

Use the **kvmrt-stat -t** command to display information about all currently running VM threads.

3
Configuring Virtual Machines

Virtual machines that are configured within the libvirt framework have an XML configuration file that controls all attributes of the virtual machine.

This file usually exists as "**/etc/libvirt/qemu/{DOMAIN}.xml**" for the given VM domain name and is created when the VM is created or imported into the libvirt framework. This file gets updated when VM configuration changes are made in the Virtual Machine Manager.

KVM-RT uses a simplified configuration file, explained below, to manage multiple VMs. KVM-RT updates libvirt XML configuration files as needed to keep the two files in sync.

The KVM-RT Configuration File

The default location of the KVM-RT configuration file is **/etc/kvmrt.cfg,** but all **kvmrt-*** tools that use a configuration file accept a **-f** option that allows the user to specify an alternate configuration file.

The KVM-RT configuration file uses the INI file format, where each section describes a VM. The first line of each section is the UUID, a unique VM identification number generated by libvrt. An example configuration is shown below:

```
[aeec46cc-0638-4949-ac04-146b233194a9]
name = RedHawk-8.4
title = RedHawk 8.4
description = RedHawk 8.4 VM.
nr_vcpus = 2
cpu_topology = auto
cpuset =
rt = False
rt_memory = auto
numatune = auto
hide_kvm = False
autostart = True
disabled = False
comments = This VM tends to run out of memory;
    remember to clean up

[fde74e84-0e1b-404e-90e7-72101e79c48a]
name = RedHawk-8.4-RT
title = Real-Time RedHawk 8.4
description = Configured for real-time.
nr_vcpus = 15
cpu_topology = auto
cpuset = n1-n2
```

```
rt = True
rt_memory = auto
numatune = auto
hide_kvm = False
autostart = True
disabled = False
comments = remember to change autostart to true after
    testing
```

Defined below are the field types used in the attribute description that follows:

{string}: any string

{int}: any integer

{bool}: **true** | **false** | **on** | **off** | **yes** | **no** | **1** | **0** (case-insensitive)

{ID-set}: a string that describes a set of ranges of integers in a human-readable form such as "0,2,4-7,12-15"

{CPUSET}: can be specified as a comma-separated list of CPUs or CPU ranges (eg. 0,1,16-19) but also as integers prefixed with 'n' for NUMA node, 'c' for core, 'd' for die, or 'p' for package. Additionally, the string may be prefixed with '~' to create an inverse set (eg. ~n0).

Each VM may be configured with the following attributes. Note that if an attribute is not set or it is missing from the file, the default value is used.

name = { string }

This attribute sets the VM name. This is an arbitrary, user specified name that must be unique to libvrt.
There is no default value, this attribute must be set but it can be changed.

title = {string}

This attribute sets the VM title.
The default value is "".

description = {string}

This attribute sets the VM description.
The default value is "".

nr_vcpus = {int}

This attribute defines the number of virtual CPUs in the VM.
The default value is **1**.

cpu_topology = {int}, {int}, {int} | **auto**

This attribute defines the CPU topology that is seen by the VM.

If not **auto**, the value must be a string of three positive integers separated by commas ("sockets, cores, threads"), to describe the CPU topology. sockets is

the number of CPU sockets, cores is the number of cores per socket, and threads is the number of threads per core.

When the value is **auto**, the topology is set to one socket, *nr_vcpus* cores per socket, and one thread per core.

The default value is **auto**.

NOTE

If the guest virtual machine is running a Windows operating system, the *cpu_topology* attribute may have been set to a default value that will not work well in KVM-RT. It is best to change this setting to **auto**. See the item labeleled "VMs running the Windows operating system" in the Known Issues section of the KVM-RT Release Notes document.

cpuset = {CPUSET}

This attribute defines host CPUs to which all VM threads are biased. CPUSET is defined with the other field types above. See the section "Understanding the cpuset Configuration Attribute" later in this chapter for more information.

The default value is "" (no CPU biasing).

rt_memory = {bool} | **auto**

This attribute enables memory locking of all pages used by the VM.

When the value is **auto**, this option is enabled if the *rt* attribute is enabled and disabled if *rt* is disabled.

The default value is **auto**.

numatune = {ID-set} | **auto**

This attribute sets the host NUMA node(s) to be used for memory allocation to the VM.

If not **auto**, the value must describe a set of host NUMA node IDs. The set may be empty, in which case memory will not be restricted to any host NUMA nodes.

When the value is **auto**, all NUMA nodes used by *cpuset* will be used. If *cpuset* is empty then memory will not be restricted to any host NUMA nodes.

The default value is **auto**.

hide_kvm = {bool}

This attribute hides KVM from the view of the guest OS in the VM.
The default value is **false** (do not hide KVM).

rt = {bool}

This attribute configures the VM for real-time.

The *cpuset* and *rt_memory* attributes must be configured (enabled) when this attribute is enabled. It is also recommended to configure and enable *numatune* when this attribute is enabled.

The default value is **false** (not real-time).

autostart = {bool}

This attribute enables auto-starting of the VM with **kvmrt-boot**.
The default value is **false** (do not autostart).

comments = { string }

A place for user comments. For multiple lines of comments, indent the additional line(s) with a space or TAB.

Configuration Tools

A KVM-RT configuration can be edited by running the command:

```
$ sudo kvmrt-edit-config
```

Note that KVM-RT configuration files should not be edited directly. **kvmrt-edit-config** validates and also synchronizes the configuration with **libvirt**.

A KVM-RT configuration, as interpreted by KVM-RT, can be displayed by running the command:

```
$ sudo kvmrt-show-config
```

The **kvmrt-validate-config** and **kvmrt-sync-config** commands can be run to validate and synchronize, respectively, a configuration. Users do not normally need to run these commands directly when using **kvmrt-edit-config**.

Run any of the above commands with the **--help** option for more information and options.

Advanced Libvirt Configuration

Advanced configuration that is beyond the scope of the KVM-RT configuration file may be made to the **libvirt** XML files, using Virtual Machine Manager or 'virsh edit', but additional synchronization and validation steps are required for KVM-RT. This is also true when you remove a VM from **libvrt.**

Note that some combinations of configuration may be invalid and users are encouraged to make configuration changes by editing the KVM-RT configuration file with **kvmrt-edit-config** whenever possible.

If libvirt XML files are modified by the user outside of KVM-RT, then it is necessary to run **kvmrt-sync-config -r** and **kvmrt-validate-config**, like so:

```
$ sudo kvmrt-sync-config -r
$ sudo kvmrt-validate-config
```

Also note that **kvmrt-import -u** may be used instead of **kvmrt-sync-config -r**, as in:

```
$ sudo kvmrt-import -u
$ sudo kvmrt-validate-config
```

The **kvmrt-validate-config** command will display appropriate errors or warnings for any invalid configuration.

Run any of the above commands with the **--help** option for more information and options.

Understanding the cpuset Configuration Attribute

The *cpuset* attribute controls host-CPU-biasing of the QEMU/KVM threads of a virtual machine.

The *cpuset* attribute may be used for both real-time and non-real-time VMs.

For non real-time VMs, all the CPUs in the cpuset can be allocated to any QEMU/KVM thread. Under-provisioning of host CPUs (less CPUs in *cpuset* than *nr_vcpus* +1) results in more than one vCPU being biased to a host CPU. If *cpuset* is empty then the VM will not be bound to any particular host CPUs.

For real-time VMs, host CPUs in *cpuset* are assigned to vCPUs in order, starting with the lowest numbered CPU. The rest of the CPUs (at least one more is required) are used for non-vCPU threads. Under-provisioning of host CPUs is not allowed for real-time VMs and *cpuset* is not allowed to be empty.

Understanding KVM-RT Use of RedHawk Real-Time Features

When the *rt* configuration attribute is enabled in the configuration file, the following RedHawk real-time system features are performed:

- All the CPUs in *cpuset* are shielded. See **shield(1)**.
- Hyperthreaded siblings are downed. See **cpu(1)** and "KVM-RT Use of Threaded CPUs" in the section below.
- Memory Locking is enabled. See the **-L** option of **run(1)**.

It is recommended that when the *rt* configuration attribute is enabled, that *numatune* also be enabled. When *numatune* is enabled NUMA nodes specified are to be used for memory allocation to the real-time VM. See **NUMA(7)**.

KVM-RT Use of Threaded CPUs

On host systems having a threaded-CPU architecture such as Intel's Hyper-Threading or AMD's SMT, KVM-RT gives special treatment to multi-threaded CPU cores when a real-time VM is in use.

Real-time demands that only one threaded sibling CPU be in use to avoid contention of CPU core resources (e.g. caches, etc.). To ensure this, KVM-RT shuts down all but one threaded sibling CPU for each CPU core allocated to a real-time VM. This requires some consideration when assigning VM *cpuset*s.

A real-time VM will be given ownership of all threaded sibling CPUs that are related to the CPUs specified in its *cpuset*. This may result in the VM consuming but not using more CPUs than it has specified in it's *cpuset*. Only one CPU per threaded core will be used for real-time and the others will be shutdown.

No special treatment is given to threaded cores hosting non-real-time VMs.

Configuring Real-Time Virtual Machines

Perform the following steps to configure a VM for real-time:

- enable the *rt* configuration attribute
- enable the *rt_memory* attribute (**auto** is recommended)
- consider enabling the *numatume* attribute (**auto** is recommended)
- configure the *cpuset* attribute as described below

Configuring the *cpuset* attribute for a real-time VM requires some understanding of the host system's CPU topology. Use the **hwtopo** or the **cpustat** command to see a display of the host system's CPU topology. **hwtopo** displays the layout of NUMA nodes, CPU cores, and logical CPUs. The following example shows the command output for a multi-threaded architecture with multiple NUMA nodes:

```
$ hwtopo -v --no-io
Machine 0 (Supermicro M12SWA-TF, "TEST_MACH1"):
   Package 0 (AMD Ryzen Threadripper PRO 5975WX 32-
                Cores):
      L3 Cache (32MiB):
         NUMA Node 0 (31GiB)
         Core 0:
            CPU 0
            CPU 32
         Core 1:
            CPU 1
            CPU 33
         Core 2:
            CPU 2
            CPU 34
         Core 3:
```

```
            CPU 3
            CPU 35
         Core 4:
            CPU 4
            CPU 36
         Core 5:
            CPU 5
            CPU 37
         Core 6:
            CPU 6
            CPU 38
         Core 7:
            CPU 7
            CPU 39
      L3 Cache (32MiB):
         NUMA Node 1 (31GiB)
         Core 8:
            CPU 8
            CPU 40
         Core 9:
            CPU 9
            CPU 41
         Core 10:
            CPU 10
            CPU 42
         Core 11:
            CPU 11
            CPU 43
         Core 12:
            CPU 12
            CPU 44
         Core 13:
            CPU 13
            CPU 45
         Core 14:
            CPU 14
            CPU 46
         Core 15:
            CPU 15
            CPU 47
      L3 Cache (32MiB):
         NUMA Node 2 (31GiB)

...
```

The following rules should be observed when configuring a real-time VM for optimal performance. The KVM-RT tools will display appropriate errors or warnings when any of the rules are violated. Errors must be corrected to continue, but warnings serve as reminders that your configuration may not be optimal.

- The *cpuset* of a real-time VM cannot overlap the *cpuset* of any other VM.
- The *cpuset* of a real-time VM must not be under-provisioned for the number of CPUs configured in the *nr_vcpus* attribute.

- Careful consideration should be given if the *cpuset* of a real-time VM spans multiple NUMA nodes.
- Careful consideration should be given if the *cpuset* of any other VM shares NUMA nodes with a real-time VM.
- Careful consideration should be given if *numatune* is not enabled for a real-time VM, or if the *numatune* node set is not contained within the NUMA nodes used by the *cpuset*.
- Careful consideration should be given if the *numatune* node set of any other VM overlaps with the NUMA nodes used by a real-time VM's *cpuset*.
- The *cpusets* of all real-time VMs must not consume all host CPUs. This is because some CPUs must be available for the KVM-RT host OS.

Adhering to the following recommendations will help simplify real-time VM configuration:

- Always configure *cpuset* with a least *nr_vcpus* + 1 host CPUs.
- Do not configure the *cpuset* of any other VM to conflict with this VM's *cpuset*, or to use any other CPUs in a NUMA node used by this VM.
- Do not let the *cpuset* span multiple NUMA nodes.
- Set *numatune* to **auto**.
- Do not configure the *numatune* of any other VM to include the NUMA node used by this VM.
- Use the **kvmrt-show-config** command to view the real-time policy configured for all VMs.
- Use the **kvmrt-stat -t** command to display the CPU-biasing of all currently running VM threads.

4
KVM-RT Tools

The KVM-RT tools are self-documented. Use the **`--help`** option of the command for more information. Following is a quick description of each tool arranged by function.

KVM-RT System Commands

`hwtopo:`

By default, it displays the hardware topology of the current system according to **`hwloc`** (**`lstopo --of xml`**). Optionally you can display the hardware topology of another system.

`cpustat`:

Displays all CPUs shown in a topology tree, along with various CPU, cache and memory information, as well as the state of RedHawk CPU shielding and downing. Optionally you can display the CPU affinity of IRQs and tasks and the IO bridge and device topology. You can specify a set of CPUs to prune the topology displayed.

`irq-affinity`:

Displays or changes the CPU affinity of IRQs on the current system. The **`--set`** option can be used to move IRQs from one CPU or CPU set to another. With the **`-c`** option, you can limit the search to a specific CPU or CPU set.

`task-affinity`:

Displays or changes the CPU affinity of tasks on the current system. The **`--set`** option can be used to move tasks from one CPU or CPU set to another. With the **`-c`** option, you can limit the search to a specific CPU or CPU set.

KVM-RT Start-Up Commands

`kvmrt-validate-host:`

Verifies that the current system configuration is valid for a KVM-RT host. It will provide suggestions on changes to be made if not.

`kvmrt-import:`

Imports **libvirt** virtual machines into a KVM-RT configuration file. By default, all libvirt VMs on the current system will be imported, but individual VMs may be specified instead. Any VMs already listed in the KVM-RT configuration file will be skipped, unless the **-u** option is used.

KVM-RT Configuration Commands:

kvmrt-edit-config:

Allows a user to edit, validate, and synchronize a KVM-RT configuration file. Several options are available including the **-f** option by which you can specify a configuration file other than the default.

kvmrt-show-config:

Displays the configuration of virtual machines in a KVM-RT configuration. Several options are available including the **-f** option by which you can specify a configuration file other than the default.

kvmrt-sync-config:

Synchronizes **libvirt** VM configuration **XML** files with a KVM-RT file. By default, all VMs in the KVM-RT configuration file are synchronized, but individual VMs may be specified instead. Optionally you can just query the state.

kvmrt-validate-config:

Validates a KVM-RT configuration file. The **-f** option is available to specify a config file other than the default.

KVM-RT Boot/Shutdown Commands

kvmrt-boot:

Boots virtual machines in a KVM-RT configuration, after validating the configuration. By default, all VMs in the configuration with the "autostart" configuration parameter enabled are booted, but individual VMs may be specified instead. If any VMs are running it simply re-tunes them as required for real-time and boot errors are ignored. When the option **--clean** is specified, no virtual machine may already be running and no boot errors are tolerated.

kvmrt-shutdown:

Shuts down virtual machines and removes any real-time policy used by those VMs. By default, all VMs in the configuration are shutdown in

parallel, but individual VMs may be specified instead. A **`--force`** option is available.

`kvmrt-stat:`

Displays the status of virtual machines in a KVM-RT configuration. By default, all VMs are shown but individual VMs may be specified instead.

5
Time Synchronization

Chrony is a versatile time synchronization implementation of NTP. It is designed to perform well in a wide range of conditions and can be run on virtual machines. Specific instructions are included here on how to configure and start the chrony system. See **chronyd(1)**, **chrony.conf(5)** and on-line documentation for more information.

NOTE

chronyd is supported by RedHawk releases 8.0 and later only. For earlier releases use **chrony/ntp** synchronized to a local, remote or public time server.

Complex applications may depend on the time of day to be synchronized between two or more VMs or with the host. It is also required that the time of day on the virtual guests be synchronized with the host when using RedHawk tracing to analyze performance issues or debug system problems with real-time VMs.

Instructions to run chrony

There are various techniques to synchronize the time of day clock on the virtual guests but we recommend kvm_clock synchronized with chrony via the **ptp_kvm** module.

The process of configuring **chronyd** to use **ptp_kvm** differs slightly depending on the base distribution.

If you are using Ubuntu as your base distro, use these settings:

```
service=chrony
conf=/etc/chrony/chrony.conf
drift=/var/lib/chrony/chrony.drift
```

If you are using a CentOS-compatible distro, use these settings:

```
service=chronyd
conf=/etc/chrony.conf
drift=/var/lib/chrony/drift
```

The following instructions should help in configuring chrony on a virtual guest. Substitute the variable settings below for the appropriate distro settings above.

1. If not already installed, install chrony.

   ```
   dnf install chrony
   ```

2. Stop and disable chrony.

```
systemctl stop $service
systemctl disable $service
```

3. Load the ptp_kvm module on boots.

```
echo ptp_kvm > /etc/modules-load.d/ptp_kvm.conf
```

4. Edit the appropriate chrony configuration file and comment out (place a # sign in front) any lines that reference 'refclock' 'server' 'pool' or 'peer'.

```
grep 'refclock|server|pool|peer' $conf && vi $conf
```

5. Configure 'refclock".

```
echo "refclock PHC /dev/ptp0 poll 3 dpoll -2 \
    offset 0" >> $conf
```

6. Comment (place a # at the front) any lines with PEERNTP and append PEERNTP=no to the /etc/sysconfig/network file.

```
grep PEERNTP /etc/sysconfig/network && \
    vi /etc/sysconfig/network
echo "PEERNTP=no" >> /etc/sysconfig/network
```

7. Remove the appropriate $drift file.

```
rm -f $drift
```

8. Enable the appropriate chronyd service but do not start it.

```
systemctl enable $service
```

9. Reboot for a clean start with the new configuration.

```
reboot
```

6
Analysis and Debugging

This chapter covers the system tools that can be used to analyze performance issues or debug system problems in virtualized environments.

A new multi-merge tracing feature is included in the latest release of the RedHawk operating system. It allows the merging of multiple system trace dumps into one view organized by timestamp. This new feature is crucial to debugging virtualized environments that often produce cross-VM and host interactions that can impact the performance of real-time applications.

In order to take advantage of the multi-merge tracing feature, all the guest VMs to be traced must be synchronized using the time of day clock (TOD). See the section “Instructions to run chrony” on page 5-1 to start-up chrony on each of the guest VMs to be traced.

NOTE

The time stamp counter (TSC) cannot be synchronized, therefore, only the TOD timestamp type should be used when tracing multiple systems. Be sure to select the TOD timestamp clock option in the trace tools.

In this chapter, the following information is presented:

- the KVM trace events supported in RedHawk.
- a brief description of the RedHawk tracing tools collectively known as **`xtrace`**. These tools use a simple command line interface. An example of tracing the host and one guest VM using xtrace and the new multi-merge feature is included.
- a new service named KVM-RT Guest Services. KVM-RT Guest Services is a collection of application programmer interfaces which give guest userspace applications access to functions exposed by the host hypervisor.

NightTrace is an optional product offered by Concurrent Real-Time. NightTrace is part of the NightStar family and consists of an interactive debugging and performance analysis tool, trace data collection daemons, and two Application Programming Interfaces (APIs) allowing user applications to log data values as well as analyze data collected from user or kernel.

For information on how to use NightTrace with KVM-RT see the "Kernel Tracing with KVM-RT" section in the NightTrace User's Guide.

KVM Trace Events

Following are the KVM traceable events supported by the RedHawk operating system.

KVM_ENTER_VM_PID

This is a generic catch-all event which will be triggered any time execution/control is transferred from the host kernel to the guest VM. It is produced by the KVM module on the host system, right before the host-guest transition.

KVM_EXIT_VM_PID

This is a generic catch-all event which will be triggered any time execution/control is transferred from guest VM to host kernel. It is produced by the KVM module on the host system, right after the guest-host transition.

KVM_GUEST_HC_START

This event is logged by the guest VM right before it makes a hypercall to the host.

KVM_GUEST_HC_END

This event is logged by the guest VM right after control returns from a hypercall.

KVM_HOST_HC_ENTER

This event is logged by the host system right after execution reached the generic hypercall handler.

KVM_HOST_HC_EXIT

This event is logged by the host system right before execution exits the generic hypercall handler.

Kernel Tracing with xtrace

xtrace is a command line interface used in the tracing and analysis of dumps.

xtrace comes with the RedHawk Operating system in the **ccur-xtrace** package and contains several tools named **xtrace-**<function>. To see all the commands and libraries provided by this package, on a RedHawk system execute:

```
rpm -ql ccur-xtrace
```

The following are the tools directly called in the example that follows. A brief description and only a few options are mentioned below. For more information and to see more options, use the **--help** option:

xtrace-run:

captures xtrace data during the execution of a shell command. The command must be specified in the command line. When the command exits **xtrace-run** stops. The **-o** option specifies the output directory name where the xtrace data will be saved. The **-m** overwrite option may be used when the tracing will go for long periods of time and the xtrace data will grow very large.

xtrace-multi-merge:

merges into one multi-merge directory the xtrace-data directories specified in the command line. These are the directories created when **xtrace-run** was invoked. In the command line specify one directory for the host and one for each guest VM traced. The **-o** option lets you specify the directory name of the multi-merge directory to be created. The **-t** option sets the xtrace time-stamp clock to be used. Note that only the time of day clock (TOD) can be synchronized.

xtrace-view:

merges and displays xtrace data in a user-readable format. The xtrace data directory must be specified.

xtrace-ctl:

provides control of the kernel xtrace module on one or more CPUs. In the non-interactive mode, commands such as FLUSH, PAUSE, RESUME are specified in the command line.

Example: multi-merge Tracing with xtrace

This example captures a trace dump on the host system and a guest VM simultaneously, and then merges the two trace dumps into one. The example assumes that the user application is known to fail within the first five minutes.

NOTE

Time of day synchronization must be configured and running before guest VMs are traced. Refer to the section "Instructions to run chrony" on page 5-1 to start-up chrony on each of the VMs to be traced.

In step 1 below, the host system is traced in the background and sleeps for a span of time greater than it takes the user application to fail.

In step 2 the tracing of the guest VM is started remotely from the host. When the user application fails on the guest VM, the trace buffer is flushed.

In step 3 the trace buffer is flushed and tracing is stopped on the host.

In step 4 the trace data directory on the guest VM is copied to the host system. In step 5 the two trace directories are merged into one and in step 6 the merged trace is arranged according to time stamp and viewed.

1. `rm -rf` *xtrace-host*
   `xtrace-run -m overwrite -t tod -o` *xtrace-host* `\`
   `sleep 600 &`
2. `ssh` *guest_vm* `"rm -rf` *xtrace-vm;*
   `xtrace-run -m overwrite -t tod -o` *xtrace-vm* `\`
   `bash -c '(` *userapp* `|| xtrace-ctl flush)'"`
3. `xtrace-ctl flush stop`
4. `scp -r` *guest-vm*`:`*xtrace-vm* `.`
5. `xtrace-multi-merge -o` *xtrace-merged xtrace-host xtrace-vm*
6. `xtrace-view` *xtrace-merged*

The fields displayed are controlled by options to **`xtrace-view`**. The fields in the following example output are: timestamp (TOD), hostname, CPU and event.

Note that CPUs are local to each host so in the excerpt that follows, "vm1 0" denotes virtual CPU 0 in the guest VM whose hostname is "vm1".

```
23.404455270 host 3 INTERRUPT_ENTER [apic_timer]
23.404455720 host 3 HRTIMER_CANCEL [0xffffffff8e8f84e0]
23.404455898 host 3 HRTIMER_EXPIRE [0xffffffff8e8f84e0]
23.404456627 host 3 SCHED_WAKEUP [740216]
23.404456854 host 3 HRTIMER_EXPIRE_DONE[0xffffffff8e8f84e0]
23.404456971 host 3 HRTIMER_START [0xffffffff8e8f84e0]
23.407646071 vm1 0 SYSCALL_EXIT [openat]
23.407646321 vm1 0 SYSCALL_ENTER [read]
23.407646512 vm1 0 FILE_READ [3]
23.407647171 vm1 0 SYSCALL_EXIT [read]
```

KVM-RT Guest Services

Virtualized environments can produce complex cross-VM and host interactions which can have detrimental effects on the performance of hard real-time applications running on the VMs. Some of these interactions might be infrequent and/or hard to reproduce. In these cases the standard approach of tracing may not suffice.

KVM-RT Guest Services is a collection of application programmer interfaces which give guest userspace applications access to functions exposed by the host hypervisor.

One of the ways to reduce complexity is to leverage the implied domain knowledge contained within each of the applications. Applications know the state the application should be in at any particular time and when any timing or state violations occur. In that context, KVM-RT Guest Services gives the application developer the ability to:

1. log relevant events/data from an application running on a guest-VM directly to a central logging/tracing facility (i.e. syslog, NightTrace, xtrace) on the host.
2. flush xtrace buffers on the host. This can be combined with local flushing of xtrace buffers on the guest to flush both guest and host buffers at about the same time.
3. log explicit pre-defined sequence of events, in the context of the host's clock, to establish ordering of events; "bracketing". For example the following was logged by two different guest VMs on the host:

   On VM1:

   ```
   host: "VM1 is about to start A"
   ...
   host: "VM1 just finished A"
   ...
   ```

   On VM2:

   ```
   host: "VM2 is about to start B"
   ...
   host: "VM2 just finished B"
   ```

   On the host, you will be able to see the order of events in the context of the host's clock:

   ```
   host: "VM1 is about to start A"
   ...
   host: "VM2 is about to start B"
   ...
   host: "VM2 just finished B"
   ...
   host: "VM1 just finished A
   ```

The KVM-RT Guest Services functions provided in the command line interface **kvmrt-gs** and the library **libccur_kvmrt_gs** are briefly discussed in the sections that follow.

Also discussed below are the traceable KVM-RT Guest Services events and the kernel boot parameters that must be enabled in the host and guest VMs.

KVM-RT Guest Services Library Interface

The following functions are provided via the library **libccur_kvmrt_gs**. See the **libccur_kvmrt_gs(3)** man page for more information on options and usage.

Note that the man page can be invoked using the names of any of the functions listed below. For example: **man kvmrt_gs_available**.

```
bool kvmrt_gs_available(void);
bool kvmrt_gs_ping_available(void);
bool kvmrt_gs_log_msg_available(void);
```

```
bool kvmrt_gs_xtrace_flush_available(void);
bool kvmrt_gs_xtrace_log_data_available(void);

long kvmrt_gs_ping(unsigned long cookie);
long kvmrt_gs_log_msg(char * msg);
long kvmrt_gs_xtrace_flush(unsigned long scope);
long kvmrt_gs_xtrace_log_data(void * data, long size);
```

`kvmrt_gs_available`

Returns true if the KVMRT_GS interface is present, enabled, and permitted. Smilarly, kvmrt_gs_<function>_available returns true if each individual KVMRT_GS function is present, enabled, and permitted.

Note that availability of the interface does not imply availability of any function. Further, an invocation of a function which is available may still fail due to variety of reasons.

`kvmrt_gs_ping`

Ping a hypervisor with a *`cookie`*. The purpose of this function is to provide a guest with a simple light-weight mechanism with no copying or allocation to explicitly cause a VMEXIT event on a hypervisor in a way which can easily be traced and matched from guest and host sides. This interface produces corresponding xtrace events, when xtrace is available.

`kvmrt_gs_log_msg`

Log a short ASCII text message via the standard kernel logging mechanism on a hypervisor side. *`msg`* is a pointer to a standard Zero-terminated C string. The hypervisor and any of the intermediate layers may restrict the maximum length of the string, and/or truncate the message. See also kvmrt_gs_xtrace_log_data below.

`kvmrt_gs_xtrace_flush`

Trigger FLUSH xtrace event on host OS.

`scope` controls which CPUs are affected by the FLUSH:

```
KVMRT_GS_XTRACE_CPU_CURRENT
KVMRT_GS_XTRACE_CPU_VM
KVMRT_GS_XTRACE_CPU_ALL
```

issue FLUSH to the current CPU, all the CPUs servicing current VM, and all the CPUs active on the host system respectively.

`kvmrt_gs_xtrace_log_data`

Log arbitrary binary *`data`* buffer containing *`size`* bytes as two matching xtrace events on guest and host sides. The hypervisor and any of the intermediate layers may restrict the maximum size of, and/or truncate the data logged. See also kvmrt_gs_log_msg above.

KVM-RT Guest Services Command Line Interface

The following commands are provided via the **kvmrt-gs** command line interface. See the **kvmrt-gs(1)** man page for more information on options and usage.

`kvmrt-gs [OPTIONS] [COMMAND [ARGUMENTS] ...] ...`

`available`

Return SUCCESS if KVM-RT Guest Services are available.

`ping_available`

Return SUCCESS if 'ping' command is available.

`ping` *COOKIE*

ping a hypervisor with a *COOKIE* - an arbitrary user-selected integer (unsigned long int).

`log_msg_available`

Return SUCCESS if the 'log_msg' command is available.

`log_msg` *MESSAGE*

Log a message on hypervisor. *MESSAGE* can be either a regular quoted ASCII string or a hex-encoded byte sequence.

`xtrace_flush_available`

Return SUCCESS if the 'xtrace_flush' command is available.

`xtrace_flush` *SCOPE*

Flush xtrace buffers on host OS. *SCOPE* can be one of the following: {0: the current CPU; 1: all the VM CPUs; 2: all host CPUs}

`xtrace_log_data_available`

Return SUCCESS if the 'xtrace_log_data' command is available.

`xtrace_log_data` *DATA*

Log xtrace event with binary data. *DATA* can be either a regular quoted ASCII string or a hex-encoded byte sequence.

KVM-RT Guest Services Trace Events

KVM-RT Guest Services logs various trace events. Every event type comes as a pair, where the *_GUEST part is logged on the guest side and the *_HOST is logged on the host.

The purpose behind such double-logging is to provide predictable reference points within the trace logs for cases where the host and the guest VM clocks might not be synchronized or have drifted in relation to each other.

KVMRT_GS_PING_GUEST
KVMRT_GS_PING_HOST

> These are produced by the "ping" function of KVM-RT Guest Services. See **kvmrt_gs_ping(3)** for details.

KVMRT_GS_FLUSH_GUEST
KVMRT_GS_FLUSH_HOST

> These are produced by the "xtrace_flush" function of KVM-RT Guest Services. See **kvmrt_gs_xtrace_flush(3)** for details.

KVMRT_GS_LOG_DATA_GUEST
KVMRT_GS_LOG_DATA_HOST

> These are produced by the "xtrace_log_data" function of KVM-RT Guest Services and is similar to XTRACE_EV_CUSTOM. See **kvmrt_gs_xtrace_log_data(3)** for details.

KVM-RT Guest Services Kernel Boot Parameters

KVM-RT Guest Services requires that the following kernel parameters must be enabled at boot time. Note that one is specific to the host system and the others to the guest VMs.

kvm.kvmrt_gs_hc_host_enabled=

> [KVM,x86] Enable KVM-RT Guest Services Hypercall on KVM Host. Setting this to 1 (Enabled) permits host to advertise KVMRT_GS hypercall and related GS functions to the guests. This is a host-side parameter for KVM module. Default is 0 (Disabled).

kvmrt_gs_hc_guest_enabled=

> [KVM_GUEST,x86] Enable KVM-RT Guest Services Hypercall on KVM Guest. Setting this option to 1 (Enabled) permits guest kernel to discover and use KVMRT_GS Hypercall and its functions if such is offered by the Host. This is a guest-side kernel parameter. Default is 0 (Disabled).

kvmrt_gs_syscall_enabled=

> [KVM_GUEST,x86] Enable KVM-RT Guest Services Syscall on KVM Guest. Setting this option to 1 (Enable) permits guest kernel to advertise KVMRT_GS syscall and its functions to the userspace applications running on the Guest. This is a guest-side kernel parameter. Default is 0 (Disabled).