1.2
REDHAWK KVM-RT™ User's Guide

concurrent REAL-TIME

0898604-1.2
February 2021

本書は当社製品を利用する社員、顧客、エンドユーザーを対象とします。
本書に含まれる情報は、本書発行時点での正確な情報ですが、予告なく変更されることがあります。
当社は、明示的、暗示的に関わらず本書に含まれる情報に対して保障できかねます。

誤字・誤記の報告または本書の特定部分への意見は、当該ページをコピーし、コピーに修正またはコメントを記述してコンカレント日本株式会社まで郵送またはメールしてください。

http://www.concurrent-rt.co.jp/company/

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

注意事項:

本書は、Concurrent Real-Time, Inc.より発行された「RedHawk KVM-RT User's Guide」を日本語に翻訳した資料です。英文と表現が異なる文章については英文の内容が優先されます。

前書き

マニュアルの範囲

本書はCocurrent Real-TimeのRedHawk KVM-RT™を利用するための情報と取扱説明について提供します。

マニュアルの構成

本書は以下のセクションで構成されます:

- 1章: KVM-RTを紹介します。
- 2章: KVM-RTで仮想マシーンをセットアップおよび起動する手順について説明します。
- 3章: KVM-RTの構成について説明します。
- 4章: 既知の問題の一覧を提示します。

構文記法

本書を通して使用される表記法は以下のとおりとなります。

斜体
: ユーザーが特定する書類、参照カード、参照項目は、*斜体*にて表記します。特殊用語も*斜体*にて表記します。

太字
: ユーザー入力は**太字**形式にて表記され、指示されたとおりに入力する必要があります。ディレクトリ名、ファイル名、コマンド、オプション、manページの引用も**太字**形式にて表記します。

`list`
: プロンプト、メッセージ、ファイルやプログラムのリストのようなオペレーティング・システムおよびプログラムの出力は`list`形式にて表記します。

`[]`
: ブラケット(大括弧)はコマンドオプションやオプションの引数を囲みます。もし、これらのオプションまたは引数を入力する場合、ブラケットをタイプする必要はありません。

ハイパーテキスト・リンク
: 本資料を見ている時に項、図、テーブル・ページ番号照会をクリックすると対応する本文を表示します。青字で提供されるインターネットURLをクリックするとWebブラウザを起動してそのWebサイトを表示します。赤字の出版名称および番号をクリックすると(アクセス可能であれば)対応するPDFのマニュアルを表示します。

関連図書

下の表にConcurrent Real-Timeの文書を記載します。文書にもよりますがRedHawk Linuxシステム上、またはConcurrent Real-Timeの文書Webサイト**http://redhawk.concurrent-rt.com/docs** からでオンラインで利用可能です。

RedHawk KVM-RT	**文書番号**
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 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

目次

1
KVM-RTの概要

本章は、RedHawk KVM-RTを使用に関する一般的な概要と要件を提供します。

概要

RedHawk KVM-RTは、ゲストのRedHawk仮想マシーンに対してRedHawkのリアルタイム・デターミニズムを拡張するためにQEMU/KVMとRedHwkのリアルタイム特性を利用するリアルタイム・ハイパーバイザー・ソリューションです。

これはホスト・システム上の仮想マシーン内で実行中の複数のゲスト、リアルタイムと非リアルタイムの両方をサポートします。

ホスト・システムの要件とインストール

ハードウェア・ホスト・システムの要件とソフトウェアのインストール手順については *RedHawk KVM-RT Release Notes*を参照して下さい。

要件ではありませんが、ホスト・システム全体をリアルタイム・ハイパーバイザーの実行に専念させることを強く推奨します。KVM-RTホスト・システムの管理者はシステム上のCPUシールディングまたはCPUアフィニティを阻害しないように注意する必要があります。さもないと仮想マシーンのリアルタイム性能が犠牲になる可能性があります。

KVM-RTがインストールされたら、ホスト・システムの適合性を分析するために次のコマンドを実行することが可能です。

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

ホスト・カーネル構成

KVM-RTが使用される間はRedHawkカーネルがホスト・システムで起動されていることをKVM-RTは要求します。追加のシステム構成が必要になる可能性があります。

カーネル起動パラメータ

これらのパラメータは、7.xリリースでは**/usr/src/linux-<***kernel-name***>/Documentation**以下の**kernel-parameters.txt**ファイル内、8.xリリースではそのパスの下のサブディレクトリ**admin-guide**以下にも記載されています。

これらのパラメータは、RedHawk Version 8.0以降では**blscfg(1)**コマンド、UbuntuリリースおよびRedHawk Version 7.xでは**ccur-grub2(1)**を使いRedHawkシステムの起動時パラメータに追加することが可能です。パラメータを有効にするには再起動が必要であること、お気付きのように全てのリリースで利用可能ではないことに注意して下さい。

intel_iommu = on / *amd_iommu* = on

本パラメータは仮想マシーンのDMAで使用されるメモリ領域のデバイス・レベルの再マッピングを有効にします。本パラメータはいずれの仮想マシーンが物理PCIデバイスのPCIパススルーを使用する場合には必要となります。

workqueue.pri = 3

リアルタイム仮想マシーンとして有効になっている仮想マシーンは物理CPUを事実上「所有」します。ホストに仮想マシーンの制御を保持させるために本パラメータはホストのデーモンおよびプロセスをより高い優先度で実行することを許可します。本パラメータはリアルタイム仮想マシーンにリアルタイム性能を保証するために必要となります。

一部のPCI-eカードはintel_iommuとは互換性がありません。ユーザーがIOMMUを有効にした時に適切に動作しないデバイスを持っている場合、次の2つのオプションが役立つ可能性があります。これらのオプションはRedHawkリリース7.5以降で利用可能であることに注意して下さい。同様にRedHawkリリース7.5では、以下に記載されている起動パラメータ*intel_iommu.exception_ids*は*intel_iommu.blacklist_ids*という名前になっていることにも注意して下さい。これは8.0以降のリリースで名称が変更されました。

intel_iommu = plx_off

PLXブリッジの後ろにある全てのデバイスをIOMMUが再マッピングするのを防ぐにはこれを有効にして下さい。これらのデバイスはPCIパススルーで使用することは出来ません。

workqueue.exception_ids = [vendor:device, …]

IOMMU再マッピングから除外されるカンマ区切りのデバイスのリスト。これらのデバイスはPCIパススルーで使用することは出来ません。ベンダーとデバイスは0xの接頭語なしの16進数として指定されます。

ユーザーはシステム内の重複するデバイスの一部だけを仮想マシーンに専念してもらいたいと望んでいるかもしれません。起動時にVFIOドライバーに対して個々のデバイスを予約するには本オプションを使用して下さい。これは起動中の初期に他の動的モジュールがデバイスを獲得するのを防止します。本オプションはRedHawkリリース8.0以降でサポートされます。

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

VFIOドライバーに割り当てられるカンマ区切りのPCIデバイスのリスト。

管理IRQの移動

CPU毎の割込みは管理割込みに分類することが可能です。最新のNIC、RAID、NVMEデバイスは管理割込みを生成します。RedHawkリリースVersion 8.2では、管理割込みを他のCPUへ移動することが可能となるように変更されました。本変更はリリース7.5以降に対してバックポートされています。最新のリリース・アップデートをお持ちである場合は本変更を得ています。管理割込みは7.5以前のリリースでは存在していなかったことに注意して下さい。

管理割込みの移動はVMのリアルタイム性能に影響を及ぼす可能性がありますのでKVM-RTでは必要です。また、KVM-RTはハイパースレッド化されたCPUを停止しようとします。IRQがCPUと関連付けられている場合はCPUを停止させることが出来ません。目標は全てのIRQをvCPUに対する責任を持つCPUから移動すること、およびエミュレーションとVirtIO操作を担当するCPUにそれらを移動する事です。

管理割込みはCPUアフィニティ・マスクに1つのCPUしか持てないため、それらを移動するのに**shield(1)**コマンドを使用することは出来ません。以下は管理IRQをCPUから移動するための方法の一部です。

1. **/proc/irq/**<*irq-no*>**/smp_affinity_list**ファイルに書き込むことでIRQのアフィニティ・マスクをリセットします。変更はブートを超えて持続しないことに注意して下さい。従って後述のカーネル起動パラメータ***msi_affinity_mask***を設定することを推奨します。次の例では、IRQ番号11はCPU 0-11と15-25で実行するように設定されます。

   echo “0-11,15-25” > /proc/irq/11/smp_affinity_list

2. カーネル起動パラメータ***msi_affinity_mask***に設定すると、起動時に全てのMSI(X)管理割込みにアフィニティ・マスクを設定します。現在、パラメータ***irqaffinity***も同じ値が設定される必要がありますが、将来のリリースでは***msi_affinity_mask***だけが設定される必要があることに注意して下さい。また、本機能はRedHawk 8.Xリリースでのみ利用可能です。

   msi_affinity_mask = [cpulist]
   irqaffinity = [cpulist]

   cpulistはCPU 0を含む必要のあるCPUのリストが設定される必要があります。リストは0-5のような範囲、または0,3,4,5のようなカンマ区切りのリストを含めることが可能です。

3. **system shield**サービスは選択したCPUのシールド属性を設定するために使用することが可能です。変更はファイル**/etc/sysconfig/shield**に対して行います。

   例えば、enp4s0f0の割込みをCPU 0から4に割り当て、または割込み番号55, 60, 61をCPU 0と2に割り当てるにはこれらの行を**shield**サービス構成ファイルに追加することで可能となります。

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

   ファイルを編集後、次のコマンドを使ってサービスを再開することが可能です：
   # systemctl restart shield

   続いてコマンドのステータスを確認することが可能です：
   # systemctl status shield

2
はじめに

本章ではKVM-RTで仮想マシーンをセットアップおよび起動する手順について説明します。また、各仮想マシーンのホスト上で実行する様々なQEMU/KVMスレッドについても説明します。

仮想マシーンの構築

KVM-RTはlibvirtフレームワーク内で生成および構成された仮想マシーンと連動します。仮想マシーンは次を含むいくつかの方法でlibvirt内に生成し構成することが可能です：

- 仮想マシーン・マネージャーを使用
- RedHawk Architectを使用
- 他の仮想マシーンのクローン作成

仮想マシーンを構築する詳細な手順は本書の範囲を超えますが、十分に文書化されています。汎用的な手順書および文書の参照は次項で提供されます。

リアルタイム仮想マシーンはRedHawk Linux 7.0以降のゲストOSが含まれている必要があります。ゲストCPUアーキテクチャはホストと一致している必要があります。

仮想マシーンを生成するために仮想マシーン・マネージャーを使用

仮想マシーン・マネージャーはlibvirtフレームワーク内の仮想マシーンを生成、構成、管理するために使用することが可能なGUIツールです。

次を実行して仮想マシーン・マネージャーを開始して下さい：

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

詳細についてはのmanページを参照して下さい。

仮想マシーンを生成するために**RedHawk Architect**を使用

RedHawk Architectは、RedHawk Linuxのディスク・イメージを生成、カスタマイズ、展開することに特化したConcurrent Real-Timeが提供するオプション製品です。

ArchitectはRedHawkの仮想マシーンを生成し、それを仮想マシーン・マネージャーにエクスポートするために使用することが可能です。詳細な手順についてはRedHawk Architectに付属する文書内で見ることが可能です。以下は必要となる一般的な手順となります：

- Architectを実行
- 新しいセッションを生成し、望むようなイメージを構成
- イメージを構築
- 仮想マシーンにイメージを展開
- 仮想マシーン・マネージャーに仮想マシーンをエクスポート

仮想マシーン・イメージのクローン作成

libvirtフレームワーク内の既存の仮想マシーンはvirt-cloneコマンドを使用することでクローンを作成することが可能です。実行例：

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

詳細については**`virt-clone(1)`**のmanページを参照して下さい。

仮想マシーンを**KVM-RT**にインポート

仮想マシーンがlibvirtフレームワーク内に生成されたら、KVM-RTにインポートすることが可能です。

全てのlibvirtの仮想マシーンは次のコマンドを使ってKVM-RTにインポートすることが可能です：

```
$ sudo kvmrt-import
```

本コマンドは新しいVMが生成された時にいつでも実行することが可能です。詳細およびオプションについては**`kvmrt-import --help`**を実行して下さい。

VMがKVM-RTにインポートされた時、VMの構成設定はlibvirtから継承します。これが終了するとVMは必要に応じてKVM-RTを使って更に構成することが可能です。詳細については3章の「仮想マシーンの構成」を参照して下さい。

仮想マシーンの起動とシャットダウン

仮想マシーンがKVM-RTにインポートされると次のKVM-RTツールがVMを起動、シャットダウン、ステータス表示するために使用することが可能です。

構成されている全てのVMを開始するには次のコマンドを実行して下さい：

```
$ sudo kvmrt-boot
```

構成されている全てのVMをシャットダウンするには次のコマンドを実行して下さい：

```
$ sudo kvmrt-shutdown
```

全てのVMのステータスを問合せるには次のコマンドを実行して下さい：

```
$ sudo kvmrt-stat
```

個々のVMは**kvmrt-boot**および**kvmrt-shutdown**コマンドを使って起動またはシャットダウンすることが可能です。実行例：

```
$ sudo kvmrt-boot RedHawk-8.2
$ sudo kvmrt-shutdown RedHawk-8.2
```

詳細およびオプションについては上記のコマンドのいずれかに**--help**オプションを付けて実行して下さい。

QEMU/KVMスレッドの理解

QEMU/KVMは各仮想マシーンに対して複数のスレッドを実行します。これらのスレッドの名称と目的は次のとおりです：

qemu-kvm

エミュレータ・スレッドです。これらは2つ以上になる可能性があります。

qemu-system-x86

一部のディストリビューションでは*qemu-kvm*の別名です。

worker

エミュレータにより実行される長いI/O操作用に動的に生成されたスレッドです。

SPICE Worker

仮想コンソール用のスレッドです。

IO mon_ioth

一部のI/Oで使用されるオプションのスレッドです。

CPU n/KVM

仮想CPU(vCPU)スレッドです。仮想CPUごとに1個あり、*n*はvCPU IDです。

現在実行中の全てのスレッドに関する情報を表示するには**kvmrt-stat -t**コマンドを使用して下さい。

3
仮想マシーンの構成

libvirtフレームワーク内に構成された仮想マシーンは、仮想マシーンの全ての属性を制御するXML構成ファイルを持っています。

本ファイルは通常、任意のVMドメイン名を表す**/etc/libvirt/qemu/{DOMAIN}.xml**として存在し、VMがlibvirtフレームワーク内に生成またはインポートされた時に生成されます。本ファイルはVM構成の変更が仮想マシーン・マネージャーで行われた時に更新されます。

KVM-RTは複数のVMを管理するために後述する簡易化された構成ファイルを使用します。KVM-RTは2つのファイルの同期を維持するために必要に応じてlivvirt XML構成ファイルを更新します。

KVM-RT構成ファイル

KVM-RT構成ファイルのデフォルトの保管場所は**/etc/kvmrt.cfg**ですが、全ての**kvmrt-***ツールはユーザーが代替の構成ファイルを指定することを許可する**-f**オプションを受け付けます。

KVM-RT構成ファイルはINIファイルの書式を使用しており、各セクションでVMについて説明します。各セクションの最初の行は、libvirtにより生成された一意的なVMの識別番号であるUUIDです。構成の実例を以下に示します：

```
[aeec46cc-0638-4949-ac04-146b233194a9]
name = RedHawk-8.2
title = RedHawk 8.2
description = A RedHawk 8.2 VM.
nr_vcpus = 2
cpu_topology = auto
cpuset =
rt = False
rt_memory = auto
numatune = auto
hide_kvm = False
autostart = True

[fde74e84-0e1b-404e-90e7-72101e79c48a]
name = RedHawk-8.2-RT
title = Real-Time RedHawk 8.2
description = A RedHawk 8.2 VM configured real-time.
nr_vcpus = 4
cpu_topology = auto
cpuset = 1-5
rt = True
rt_memory = auto
numatune = auto
```

```
hide_kvm = False
autostart = True
```

以下に定義されているのは後述する属性の説明内で使用されているフィールドの型です：

{ string }: 任意の文字列

{ int }: 任意の整数

{ bool }: **`true | false | on | off | yes | no | 1 | 0`**
(大文字と小文字の区別なし)

{ ID-set }: 「0,2,4-7,12-15」などの形式で人間が解読可能な整数の範囲のセットを説明する文字列

各VMは次の属性を使って構成されます。属性が設定されていないもしくはファイルからなくなっている場合、デフォルト値が使用されることに注意して下さい。

`name` = { string }

本属性はVMの名称を設定します。これは任意ですが、libvirtに対して一意である必要があるユーザーが指定する名称です。
デフォルトの値はなく、本属性は設定されている必要はありますが変更することが可能です。

`title` = { string }

本属性はVMのタイトルを設定します。
デフォルトの値は””です。

`description` = { string }

本属性はVMの説明を設定します。
デフォルトの値は””です。

`nr_vcpus` = { int }

本属性はVM内の仮想CPUの数を定義します。
デフォルトの値は**`1`**です。

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

本属性はVMに認識されるCPUトポロジーを定義します。

`auto`ではない場合、値はCPUトポロジーを表現するためにカンマで区切られた3つの正の整数の文字列(ソケット、コア、スレッド)である必要があります。ソケットはCPUソケットの数、コアはソケットあたりのコアの数、スレッドはコアあたりのスレッドの数となります。

値が**`auto`**である場合、トポロジーは1個のソケット、ソケットあたり*`nr_vcpus`*個のコア、コアあたり1個のスレッドが設定されます。

デフォルトの値は**`auto`**です。

NOTE

ゲストの仮想マシーンがWindowsオペレーティング・システムを実行しようとしている場合、*cpu_topology*属性はKVM-RTで正常に動作しないデフォルト値に設定されている可能性があります。これは本設定を**auto**に変更する必要があります。詳細については4章の「Windowsオペレーティング・システムを実行するVM」のラベルの付いた項目を参照して下さい。

cpuset = { ID-set }

本属性は全てのVMスレッドがバイアスされるホストのCPU IDを定義します。デフォルトの値は""(CPUバイアスなし)です。詳細については本章で後述する「cpuset構成属性の理解」を参照して下さい。

rt_memory = { bool } | **auto**

本属性はVMで使用される全ページのメモリ・ロックを有効にします。

値が**auto**である場合、本オプションは*rt*属性が有効化されていれば有効、*rt*属性が無効化されていれば無効となります。

デフォルトの値は**auto**です。

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

本属性はVMへのメモリ割り当てで使用されるホストのNUMAノードを設定します。

autoではない場合、この値はホストのNUMAノードのセットを記述する必要があります。設定が空である場合、メモリはいずれのホストのNUMAノードにも制限されません。*cpuset*が空であった場合もメモリはいずれのホストのNUMAノードにも制限されません。

値が**auto**である場合、*cpuset*で使用される全てのNUMAノードが使用されます。

デフォルトの値は**auto**です。

hide_kvm = { bool }

本属性はVM内のゲストOSの表示からKVMを隠します。
デフォルトの値は**false**(無効)です。

rt = { bool }

本属性はリアルタイム用にVMを構成します。
デフォルトの値は**false**(無効)です。

本属性が有効である場合は*cpuset*と*rt_memory*の属性は(有効に)構成されている必要があります。本属性が有効である場合は*numatune*も有効に構成することを推奨します。

autostart = { bool }

本属性は**kvmrt-boot**を使ったVMの自動起動を有効にします。
デフォルトの値は**true**(有効)です。

構成ツール

KVM-RTの構成は次のコマンドを実行することで編集することが可能です：

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

KVM-RT構成ファイルは直接編集すべきではないことに注意して下さい。**kvmrt-edit-config**は妥当性を検証し、また、**libvirt**と構成を同期させます。

KVM-RTによって解釈されるKVM-RT構成は、次のコマンドの実行により表示することが可能です：

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

kvmrt-validate-configと**kvmrt-sync-config**のコマンドはそれぞれ構成の妥当性の検証および同期させるために実行することが可能です。**kvmrt-edit-config**を使用する場合、ユーザーは通常これらのコマンドを直接実行する必要はありません。

詳細およびオプションについては上記のコマンドのいずれかに**--help**オプションを付けて実行して下さい。

高度なLibvirt構成

KVM-RT構成ファイルの範囲を超える高度な構成は、仮想マシーン・マネージャーまたは「virsh edit」を使って**libvirt** XMLファイルに対して行うことが可能ですが、追加の同期および妥当性の検証がKVM-RTに必要となります。これは**libvirt**からVMを削除する場合も当てはまります。

一部の構成の組み合わせは無効である可能性があることに注意し、いつであろうともユーザーは**kvmrt-edit-config**を使いKVM-RT構成ファイルを編集して構成を変更することを推奨します。

libvirt XMLファイルをKVM-RTの外でユーザーが変更した場合、次のように**kvmrt-sync-config**および**kvmrt-validate-config**を実行する必要があります：

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

また、次のように**kvmrt-import -u**を**kvmrt-sync-config -r**の代わりに使用することも可能であることに注意して下さい：

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

kvmrt-validate-configコマンドはどの無効な構成に関しても適切なエラーまたは警告を表示します。

詳細およびオプションについては上記のコマンドのいずれかに**--help**オプションを付けて実行して下さい。

cpuset構成属性の理解

*cpuset*属性は仮想マシーンのQEMU/KVMスレッドのためのホストCPUバイアスを制御します。

*cpuset*属性はリアルタイムと非リアルタイムVMの両方で使用することが可能です。*cpuset*が空の場合、VMはどの特定のホストCPUに対しても固定されません。

*cpuset*の最初のCPUは全ての非vCPUスレッドに割り当てられます。*cpuset*の残りのCPUは次のようにvCPUスレッドにより使用されます：

- 全ての仮想CPUスレッドのCPU配列ポリシーは、(1つのCPUに全ての非vCPU VMスレッドを割り当てた後に)*cpuset*により定義されたホストCPU上でラウンド・ロビンとなります。
- ホストCPUの供給過多(*cpuset*内のCPUが*nr_vcpus* + 1以上)は結果的に各仮想CPUが1個以上のホストCPUに固定されます。
- ホストCPUの供給不足(*cpuset*内のCPUが*nr_vcpus* + 1以下)は結果的に各ホストCPUに1個以上の仮想CPUが固定されます。

リアルタイム仮想マシーンの構成

リアルタイム用VMを構成するには次の手順を実行して下さい：

- *rt*構成属性を有効化
- *rt_memory*属性を有効化 (**auto**を推奨)
- *numatune*属性の有効化を検討 (**auto**を推奨)
- 以下で説明するように*cpuset*属性を構成

リアルタイムVMに関する*cpuset*属性を構成するには、ホスト・システムのCPUトポロジーを多少理解していることが求められます。ホスト・システムのCPUトポロジーの表示を見るには**cpu-topology**コマンドを使用して下さい：

```
$ cpu-topology
```

詳細およびオプションについては**cpu-topology --help**を実行して下さい。

cpu-topologyはCPUソケット、NUMAノード、CPUコア、論理CPUのレイアウトを表示します。出力例は次のようになります：

```
NUMA Node 0:
     Core 0:   [Socket 0]
         CPU 0
     Core 1:   [Socket 0]
         CPU 1
     Core 2:   [Socket 0]
         CPU 2
     Core 3:   [Socket 0]
         CPU 3
```

システムがIntel Hyper-Threadingなどのスレッド化CPUアーキテクチャを搭載している場合、出力は次のようになります：

```
NUMA Node 0:
     Core 0:   [Socket 0]
         CPU 0
         CPU 4
     Core 1:   [Socket 0]
         CPU 1
         CPU 5
     Core 2:   [Socket 0]
         CPU 2
         CPU 6
     Core 3:   [Socket 0]
         CPU 3
         CPU 7
```

NUMAシステムが1つ以上のNUMAノードを搭載している場合、次のようになります：

```
NUMA Node 0:
     Core 0:   [Socket 0]
         CPU 0
         CPU 16
     Core 1:   [Socket 0]
         CPU 1
         CPU 17
     Core 2:   [Socket 0]
         CPU 2
         CPU 18
     ...
NUMA Node 1:
     Core 8:   [Socket 1]
         CPU 8
         CPU 24
     Core 9:   [Socket 1]
         CPU 9
         CPU 25
     Core 10:  [Socket 1]
         CPU 10
         CPU 26
     ...
```

リアルタイムVMを構成する場合は次のルールを守る必要があります：

- リアルタイムVMの*cpuset*は、他のどのVMの*cpuset*と重複することも出来ません。
- リアルタイムVMの*cpuset*は、*nr_vcpus*属性で構成されたCPUの数に対して供給不足となってはなりません。
- リアルタイムVMの*cpuset*が複数のNUMAノードに広がる場合、慎重な考慮が必要となります。
- 他のどのVMの*cpuset*がリアルタイムVMとNUMAノードを共有する場合、慎重な考慮が必要となります。

- リアルタイムVMに対して*numatune*が有効ではない場合、または*numatune*ノード・セットが*cpuset*で使用されるNUMAノードの中に含まれていない場合、慎重な考慮が必要となります。
- 他のどのVMの*numatune*ノード・セットがリアルタイムVMの*cpuset*で使用されるNUMAノードと重複している場合、慎重な考慮が必要となります。
- 全てのリアルタイムVMの*cpuset*はホストCPU全てを消費してはなりません。これは一部のCPUはKVM-RTのホストOS用に利用可能である必要があるためです。

KVM-RTツールは上記ルールのいずれかに違反した場合、適切なエラーまたは警告を表示します。

次の推奨事項を忠実に守ることはリアルタイムVM構成の簡略化に役立ちます：

- 常時、最小で*nr_vcpus* +1のホストCPUを*cpuset*に構成して下さい。
- 他のいずれのVMと対象のVMの*cpuset*が競合する、または対象のVMで使用するNUMAノード内の他のCPUを使用するような構成をしないで下さい。
- *cpuset*が複数のNUMAノードに広がらないようにして下さい。
- *cpuset*を構成する場合、各ソケット内の最高値のコアを出発点にカウント・ダウンしてVMのCPUを割当てる、必ず各ソケット内の最初のコアはKVM-RTホスト専用とするようにして下さい。例えば、2ソケット、ソケットあたり10コアのCPU構成において、VMへのCPU割り当てはCPU #9(最初のソケットの最後のコア)から始めてカウント・ダウン、CPU #19(2番目のソケットの最後のコア)から始めてカウント・ダウンする必要があります。最低限でも、CPU #0(最初のソケットの最初のコア)とCPU #10(2番目のソケットの最初のコア)はKVM-RTホストが使用するために未割当てのままとする必要があります。本推奨事項の説明については4章の「移動できないIRQ」のラベルの付いた項目を参照して下さい。
- *numatune*は**auto**に設定して下さい。
- 他のいずれのVMの*numatune*が対象のVMで使用するNUMAノードを含むような構成にしないで下さい。
- 全てのVMで構成されるリアルタイム・ポリシーを表示するには**kvmrt-show-config**コマンドを使用して下さい。
- 現在実行中の全てのVMスレッドのCPUバイアス状況を表示するには**kvmrt-stat -t**コマンドを使用して下さい。

スレッド化CPUを使用するKVM-RTの理解

IntelのHyper-Threadingなどのスレッド化CPUアーキテクチャを搭載するホスト・システムでは、リアルタイムVMが使用される場合はKVM-RTはマルチ・スレッドCPUコアに特別な処理を行います。

リアルタイムはCPUコア・リソース(例えば、キャッシュ等)の競合を回避するためにスレッド化されたシブリングCPUを1つだけ使用することを要求します。これを確実にするため、KVM-RTはリアルタイムVMに割付けた各CPUコアにおいてスレッド化されたシブリングCPUの1つを除いて全て停止します。これはVMの*cpuset*を指定する時にいくつかの考慮を必要とします。

リアルタイムVMは、*cpuset*に指定されたCPUに関連する全てのスレッド化されたシブリングCPUの所有権が与えられます。

これは結果としてVMはCPUを消費しますが*cpuset*に指定されているよりも多くは使用しないことになる可能性があります。スレッド化コアあたり1つのCPUだけがリアルタイムで使用され、他は停止されることになります。

非リアルタイムVMに提供するスレッド化コアに対しては特別な処理は行われません。

4
既知の問題

次の分野に特別な配慮が行われる必要があります：

KVM-RT Version 1.2での既知の問題

起動中にVMが無応答

時間管理のジャンプが発生する可能性のあるVMの世界では、TSCは問題がある可能性があります。「hide_kvm」オプションが実装されている場合、安全なkvm_clock TSCはもはや使用不可能です。RedHawk 6.xなどの古いRedHawkカーネルは、より不安定なTSC(および他の時間管理ソース)を使って起動することに苦労しています。TSCに切り替えようとしている時、時々VMは起動処理で無応答になります。VMが起動しない場合は、**kvmrt-edit-config**を使用して「hide_kvm = False」を設定して下さい。

グラフィック負荷の高いプログラム

グラフィック負荷の高いVMが専用のGPUハードウェアを使用しない場合、エミュレートされたグラフィックまたはグラフィック負荷の高いプログラムは、VMが他の仮想マシーンのリアルタイム性能に影響を及ぼす可能性があります。CPUのオンボード・グラフィックは、ホストおよびVMのためのVRAMメモリのアクセスに関してはチップ上のメモリ・コントローラーに依存しています。メモリ・コントローラーがグラフィックのアクセスで過負荷となった場合、リアルタイムVMは性能への打撃に苦しむ可能性があります。

Windowsオペレーティング・システムを実行するVM

デフォルトのlibvirt CPUトポロジー設定を使用した場合、Windows仮想マシーンの性能が劇的に低下するWindowsオペレーティング・システムの「ソケット単位」のライセンスが存在します。ユーザーは、単一ソケット上の多数コア/スレッドとなるようにKVM-RT構成内の*cpu_topology*設定を調整する必要があります。Windowsを実行するVMに対して*cpu_topology*パラメータを**auto**に設定することを推奨します。CPUトポロジーの設定が調整されていない場合、Windows VMはシングルCPUであるかのように動作しシステム性能は減速します。

CPU毎IRQ

一部のデバイス・ドライバーはCPU毎IRQを使用します。これらのIRQはリアルタイムVMの性能に影響を及ぼします。リアルタイム性能に影響を与える可能性のあるスレッド化CPUアーキテクチャ内のシブリングCPUの停止を阻害する可能性もあります。一部のCPU毎IRQは**shield**システム・サービスを使って移動させることが可能です。本サービスの変更は、構成ファイル**/etc/sysconfig/shield**を編集して行うことが可能で、その変更は**systemctl(1)**を使用して有効な状態にすることが可能です。

移動できないIRQ

一部のアーキテクチャでは、IRQは各ソケット内の最初のコアの最初のCPUにバインドされ、他のCPUへ移動することができません。スレッド化CPUアーキテクチャではコアは1つ以上のCPUを搭載しているので、「最初のCPU」という言葉が使用されていることに注意して下さい。これらのIRQはこれらのCPUで実行するリアルタイムVMの性能に影響を及ぼす可能性がありますので、各ソケットの最初のコアの最初のCPUはKVM-RTホスト用に予約しリアルタイムVMへは割付けないことを推奨します。