2.0
KVM-RT™ User's Guide

0898604-2.0
April 2025

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
October 2021	1.3	RedHawk Linux 8.4
March 2023	1.4	RedHawk Linux 8.4
December 2023	1.5	RedHawk Linux 9.2
September 2024	1.6	RedHawk Linux 9.2
April 2025	2.0	RedHawk Linux 9.2

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の構成する方法を取り上げます。
- 4章 全てのKVM-RT CLIツールを要約します。
- 5章 **`kvmrt`**グラフィカル・ユーザー・インターフェース構成ツールを説明します。
- 6章 時刻の同期を説明します。
- 7章 仮想環境でのI/Oの利用を説明します。
- 8章 KVM-RTのゲストVMを解析およびデバッグする方法を説明します。
- 付録A Supermicro M12SWA-TFとASUS Pro WS WRX90E-SAGE SEの事前認証プラットフォームのNUMAノード・マッピングを取り上げます。
- 付録B シングル・ルートI/O仮想化(SR-IOV)の設定手順を提供します。
- 付録C PCIパススルーで必要となる可能性のある起動パラメータを提示します。

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 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

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、単一ホスト・システム上のゲストのRedHawk仮想マシンに対してRedHawkのリアルタイム・デターミニズムを拡げるためにクイック・エミュレータ(QEMU: Quick Emulator)とカーネル・ベース仮想マシン(KVM: Kernel-Based Virtual Machine)およびRedHawk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。

RedHawk KVM-RTは、ユーザーに次の機能を提供するためにRedHawk Linuxと共にQEMUとKVMの機能を活用しています：

- 過去のオペレーティング・システムの仮想化
- 物理的なシステム・ハードウェアの設置面積を縮小
- 安全に隔離された環境でリアルタイムおよび非リアルタイムの仮想マシンを実行
- VMがほぼベアメタル構成での実行を可能にするI/O実装ソリューションを提供

当初より提供されているCLIツールに加え、**`kvmrt`**というGUIツールが2.0リリースに追加されました。両方ともKVM-RT攻勢を生成、管理、制御するために使用することが可能ですが、CLIツールはGUIでは提供されないいくつかの拡張機能を提供します。**`kvmrt`** GUIとCUIの両ツールは同期して動作し、互いに連動して使用することが可能です。例えば、CLIコマンド**`kvmrt-import`**は**`libvirt`**からVMをインポートするために使用可能である一方、GUIツールはKVM-RT構成にインポートされた各VMを構成するために使用することも可能です。

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が使用されている間、KVM-RTはホスト・システム上でRedHawkカーネルが起動されていることを要求します。更なるシステム構成が必要となる可能性があります。

例えば、PCIパススルーは起動パラメータの付加が必要になる可能性があります。詳細については付録Cの「PCIパススルー起動パラメータ」を参照して下さい。

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検査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
```

2
はじめに

本章ではKVM-RTで仮想マシ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ンと連動します。仮想マシ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：

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

仮想マシ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。

仮想マシンを生成するために仮想マシン・マネージャーを使用

仮想マシン・マネージャーはlibvirtフレームワーク内の仮想マシンを生成、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**`virt-manager(1)`**のmanページを参照して下さい。

仮想マシ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の仮想マシンを生成し、それを仮想マシン・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
- 仮想マシンにイメージを展開
- 仮想マシン・マネージャーに仮想マシンをエクスポート

仮想マシン・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。

仮想マシンを**KVM-RT**にインポート

仮想マシンが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「仮想マシンの構成」を参照して下さい。

仮想マシンの起動とシャットダウン

KVM-RT用に***kvmrt***という名称の**systemd**サービスが存在します。これは、自動開始設定が構成されたVMはシステム起動中に自動的にブートされ、システムがシャットダウン中に実行中のVMはシャットダウンされることを有効にすることが可能です。本サービスはデフォルトでは有効となっていません。有効にするとサービスは次回の起動で自動的に開始されます。直ぐに開始したい場合、次のようにサービスを有効にし、それを開始する必要があります：

```
systemctl enable kvmrt
```

```
systemctl start kvmrt
```

--cleanオプション付きで**kvmrt-boot**を呼び出すため、実行中のVMがある場合はサービス開始が失敗することに注意して下さい。

以下のKVM-RTツールはVMの起動、シャットダウン、ステータスを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：

```
$ sudo kvmrt-boot
```

実行中の全てのVMをシャットダウン：

```
$ sudo kvmrt-shutdown
```

全てのVMのステータスを問合せ：

```
$ sudo kvmrt-stat
```

これら全てのコマンドに個々のVMを指定することが可能です。実行例：

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

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

デフォルトでVMは同時に停止されることに注意して下さい。**-v** (レポート表示)オプションを使用した場合、異なるVMからの出力が混同されないようにシャットダウンは順に実行されます。

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でも**--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における*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
仮想マシンの構成

libvirtフレームワーク内に構成された仮想マシンは、仮想マシ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、構成ファイルを使用する全ての**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を記述します。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。構成の実例として2つのゲストVMを付け加えますが、2番目は未使用(無効)です：

```
[fde74e84-0e1b-404e-90e7-72101e79c48a]
name = RedHawk-8.4-RT
title = Real-Time RedHawk 8.4
description = Configured for real-time
nr_vcpus = auto
cpu_topology = auto
cpuset = n1,n2
rt = True
rt_memory = auto
numatune = auto
hide_kvm = False
autostart = True
disabled = False
pcidevs = 0000:21:00.0 0000:22:00.0 0000:22:00.1
comments = remember to change autostart to true after testing

[aeec46cc-0638-4949-ac04-146b233194a9]
name = RedHawk-8.4
title = RedHawk8.4
description = RedHawk8.4VM
nr_vcpus = 2
```

```
cpu_topology = auto
cpuset =
rt = False
rt_memory = auto
numatune = auto
hide_kvm = False
autostart = True
disabled = True
pcidevs =
comments = This VM is not used anymore; kept for reference
```

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

{ CPUSET }:CPUのリストまたはCPUの範囲をカンマ区切り(例： 0,1,16-19)、同様にNUMAノードは「n」、コアは「c」、ダイは「d」、パッケージは「p」の接頭辞付き整数で指定することが可能です。更に反転設定を付与するために「~」を前に置いた文字列にすることが可能です。(例： ~n0)

{ PCISET }: スペース区切りの完全なPCIバス・アドレスのリスト。各デバイスは構文「domain:bus:device.function」で記載されます。(例：`0000:21:00.00`)

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は可能です。

`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 } | **`auto`**

本属性はVM内の仮想CPUの数を定義します。**auto**に設定した場合、仮想CPUの数は「**cpuset**」に定義された物理CPUの数から1を引いた値に自動的に設定されます。**rt**がtrueである場合、ハイパースレッドのシブリングはダウンされるので**cpuset**の計算には含められません。

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

ゲストの仮想マシンがWindowsオペレーティング・システムを実行する場合、*cpu_topology*属性がKVM-RTで正しく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。本設定を**auto**に変更するのが最適です。KVM-RT Release Notesの既知の問題項にある「Windowsオペレーティング・システムが動作するVM」のラベルの付いた項目を参照して下さい。

cpuset = { CPUSET }

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を定義します。CPUSETは前述の他のフィールド・タイプで定義しています。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。

デフォルトの値は"" (CPUバイアスなし)です。

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。

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。*cpuset*が空であった場合、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。

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** (KVMを非表示にしない)です。

rt = { bool }

本属性はリアルタイム用にVMを構成します。

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。

デフォルトの値は**false** (非リアルタイム)です。

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**false** (自動起動しない)です。

disabled = { bool }

trueが設定されている場合、VMはKVM-RTから隠蔽されます。これは使用しないVM構成を確保する方法を提供します。

デフォルトの値は**false** (VMは有効化)です。

pcidevs= { PCISET }

KVM-RTがVMにPCIパススルーするスペース区切りの完全なPCIデバイス・バス・アドレスのリスト。デバイスのバス・アドレスは**pci(1)**コマンドの出力の中からデバイスを検索することで取得することが可能です。

同じIOMMUグループ内の全てのデバイスは同じVMにパススルーされる必要があります。デフォルトで**pci**コマンドは一般的なデバイスのみをリスト表示し、IOMMUグループ内の他のデバイスは除外する可能性があります。全デバイスを見るには**pci**コマンドに**-a**オプションを追加、または**kvmrt-edit-config**を失敗させて除外されたデバイスのデバイス・バス・アドレスを出力してください。

デフォルト値は""です。

comments = { string }

ユーザー・コメントの場所となります。複数行のコメントの場合、スペースまたはTABを使って追加行を字下げして下さい。

デフォルト値は""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 -r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。

非リアルタイムVMにおいて、*cpuset*内の全てのCPUはQEMU/KVMスレッドのいずれかに割り当てられます。ホストCPUの供給不足(*cpuset*内のCPUが*nr_vcpus* + 1未満)は結果的にホストCPUに1個以上のvCPUが固定されます。*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。

リアルタイムVMにおいて、*cpuset*内のホストCPUは1番小さな番号のCPUから順にvCPUへ割り当てられます。残りのCPU(少なくとも1個以上が必要)は非vCPUスレッドで使用されます。ホストCPUの供給不足はリアルタイムVMでは認められておらず、*cpuset*を空にすることも認められていません。

KVM-RTによるRedHawkリアルタイム機能の使用の理解

構成ファイル内で*rt*構成属性が有効化されている場合、次のRedHawkリアルタイム・システムの機能が実行されます：

- *cpuset*の全てのCPUがシールドされます。**shield(1)**を参照して下さい。
- ハイパースレッドのシブリングが停止されます。**cpu(1)**および後述の「KVM-RTによるスレッド化CPUの使用」を参照して下さい。
- メモリ・ロックが有効化されます。**run(1)**の**-L**オプションを参照して下さい。
- ホスト上のIRQアフィニティが変更されている可能性があります。以下参照して下さい。

PCIパススルー・デバイスを使用する各リアルタイムVMにおいて全ての関連するvfio IRQのアフィニティは、リアルタイムで使用される最後のホストCPUにIRQをバインドするように変更されます。いつVMが実行されようともこれはバックグラウンドで常に行われます。構成内のいずれのリアルタイムVMで使用されるホストのCPUいずれもバインドされないことを保証するために全てのホスト上の他のデバイスIRQは必要に応じてアフィニティが変更されます。

*rt*構成属性が有効化されている場合には*numatune*も有効にすることを推奨します。*numatune*が有効化されると指定されたNUMAノードはリアルタイムVMへのメモリ割り当てに使用されます。**NUMA(7)**を参照して下さい。

KVM-RTによるスレッド化CPUの使用

Intelのハイパースレッド、またはAMDのSMTのようなスレッドCPUアーキテクチャを持つホスト・システムにおいて、リアルタイムVMが使用されている場合にKVM-RTはマルチ・スレッド化CPUコアに対して特別な処理を提供します。

CPUコア・リソース(キャッシュ等)の競合を回避するために1つのスレッド化シブリングCPUだけが使用されることをリアルタイムは要求します。これを確実にするため、リアルタイムVMに割り当てられた各CPUコアの1つのスレッド化シブリングCPUを除いて全てをKVM-RTはシャットダウンします。これはVMの*cpuset*を割り当てる時にいくつかの考慮を必要とします。

リアルタイムVMには*cpuset*で指定されたCPUに関連する全てのスレッド化シブリングCPUの所有権が与えられます。これはVMが消費するCPUは*cpuset*で指定された以上は使用しないことになります。スレッド化コア毎に1つのCPUだけがリアルタイムで使用され、他はシャットダウンされます。

非リアルタイムVMをホストするスレッド化コアに対しては特別な処理は提供されません。

リアルタイム仮想マシンの構成

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**hwtopo**または**cpustat**コマンドを使用して下さい。**hwtopo**はNUMAノード、CPUコア、論理CPUのレイアウトを表示します。次の例は複数のNUMAノードを持つマルチスレッド・アーキテクチャのコマンド出力を示します：

```
$ 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)
...
```

最適な性能のためにリアルタイムVMを構成する場合は次のルールを遵守する必要があります。いずれかのルールに違反した場合はKVM-RTツールは適切なエラーまたは警告を表示します。エラーは継続するために是正する必要がありますが、警告は構成が最適ではない可能性があることのヒントとなります。

- 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。
- 他のいずれの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。

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。
- *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。

4
ツール

ccur-rttoolsパッケージ内に含まれるRedHawkのリアルタイム・ツールは、RedHawk LinuxとKVM-RTの両方に標準装備されています。両方の一連のツール(RedHawkのリアルタイム・ツールとKVM-RTツール)について本章で説明します。

RedHawkのリアルタイム・ツール

これらのコマンドは成し得る最良のKVM-RT構成を達成するためにシステムを調査および変更することに使用が可能です。

とても簡素な説明のみを以下に記述します。詳細については各ツールで提供されるmanページもしくは*--help*オプションを使用して下さい。これらのコマンド全てが記載されている**rttools(7)**のmanページもあります。

コマンド・ライン・インターフェース

cpus
CPUの状態を表示または変更します。

irqs
システム上のIRQに関する情報を表示します。

tasks
システム上のタスク(プロセス・スレッド)に関する情報を表示します。

hwtopo
システムのハードウェア・トポロジーを表示します。

pci
PCIデバイス情報を表示します。

irq-affinity
IRQのCPUアフィニティを表示または変更します。

task-affinity
タスク(プロセス・スレッド)のCPUアフィニティを表示または変更します。

cpustat
ハードウェア・トポロジー、CPUの状態、IRQ、タスク実行、CPUアフィニティを1つに結合して表示します。

cpi
IRQの割込み回数をCPU毎に表示します。

グラフィカル・ユーザー・インターフェース

hwtopo-gui
システムのハードウェア・トポロジーをGUIにて表示します。

cpustat-gui

ハードウェア・トポロジー、CPUの状態、IRQ、タスク実行、CPUアフィニティを1つに結合してGUIにて表示します。

interview

CPU毎の割込み回数をリアルタイムでGUIにて表示します。

KVM-RTツール

これらのコマンドはKVM-RT固有です。各コマンドに関連するmanページと*--help*オプションがあります。これらのツール全てが記載されている**kvmrt(7)**のmanページもあります。

殆どのKVM-RTツールはデフォルトで**/etc/kvmrt.cfg**ファイルを使用しますが、別の構成ファイルを**-f**オプションを介して指定することが可能です。

開始コマンド

kvmrt-validate-host:

現在のシステム構成がKVM-RTホストとして有効であるかどうかを検証します。そうではない場合に行う変更の提案を提供します。

kvmrt-import:

KVM-RT構成ファイルに**libvirt**仮想マシンをインポートします。デフォルトで現在のシステム上の全てのlibvirt VMがインポートされますが、代わりに個々のVMを指定することも可能です。**-u**オプションを使用しない限り、既にKVM-RT構成ファイルに記載されているVMはいずれもスキップされます。

構成コマンド

kvmrt-edit-config:

ユーザーがKVM-RT構成ファイル**/etc/kvmrt-edit-config**の編集、検証、同期するのを許可します。これはデフォルトの構成ファイルですが、他を**-f**オプションで指定することが可能です。

kvmrt-show-config:

KVM-RT構成にある仮想マシンの構成を表示します。表示する情報を制御するためにオプションが利用可能です

kvmrt-sync-config:

libvirtのVM構成の**XML**ファイルとKVM-RT構成ファイルを同期します。デフォルトでKVM-RT構成ファイル内の全てのVMが同期されますが、代わりに個々のVMを指定することも可能です。

オプションで状態を問い合わせるだけも可能です。

kvmrt-validate-config:

構成内の全てのVMが個々に検証されるだけでなく結合されたVMの衝突も評価されます。デフォルトで、無効化されていない構成内のVMだけが評価されますが、無効にするために**-all**オプションを使用することが可能です。

起動/シャットダウン・コマンド

kvmrt-boot:

構成を検証した後、KVM-RT構成内の仮想マシンを起動します。デフォルトで「autostart」構成パラメータが有効である構成内の全てのVMが起動されますが、**--all**オプションが構成内の全てのVMを起動するために使用することが可能です。代わりに個々のVMを指定することも可能です。

いずれかのVMが実行中の場合、リアルタイムに対しては必要に応じて単に再調整して起動エラーは無視されます。オプション**--clean**が指定された場合、既に実行中の可能性のある仮想マシンがなければ許容される起動エラーはありません。

kvmrt-shutdown:

仮想マシンをシャットダウンしそれらのVMで使用されていたいずれのリアルタイム・ポリシーも削除します。デフォルトで構成内の全てのVMが同時にシャットダウンされますが、代わりに個々のVMを指定することも可能です。**-v**詳細オプションを使用するとシャットダウンからの出力が不明瞭とならないようにシャットダウンがシリアル化されます。**--force**オプションが利用可能です。

kvmrt-stat:

KVM-RT構成内の仮想マシンの状態を表示します。デフォルトで全ての有効化されたVMが表示されますが、**--all**オプションは無効化されたVMも表示します。代わりに個々のVMを指定することも可能です。

5
KVM-RT GUI

kvmrtのGUIはKVM-RT構成を生成し管理するのが容易なグラフィカル・ユーザー・インターフェースを提供します。これは**kvmrt**で起動し、対応するmanページは**kvmrt-gui(1)**です。

GUIとCLIツールの両方にKVM-RT構成を生成し管理する機能一式を含んでいます。両方が同期して動作し、一緒に使用することも可能です。例えば、GUIツールがKVM-RT構成にインポートされた各VMを構成するために使用されている間、ユーザーが**libvirt**からVMをインポートするためにCLIコマンド**kvmrt-import**を使用することも可能です。

NOTE

仮想マシンはKVM-RT構成ツール外で生成されます。2-1ページの「仮想マシンの構築」を参照してください。

NOTE

2-1ページの「はじめに」の章と3-1ページの「仮想マシンの構成」の章で説明している全ての概念を用いてKVM-RT構成を生成する前に理解する必要があります。GUIを使用する場合はこれらの章に記載されたCLIコマンドの説明は無視することが可能です。

kvmrtのGUIで使用される2種類のウィンドウは次のとおりです：

1. ダッシュボード： VMを制御および監視するために永続的に使用します。
2. 構成ダイアログ： 構成の変更を本ダイアログで行います。

ダッシュボード

KVM-RTダッシュボードは**kvmrt**が起動している間は存在し続け、KVM-RT構成内のVMを監視し制御するために使用されます。次項でダッシュボードの様々な構成部品を説明します。

一番初めの定義された構成ファイルが存在しない場合、ダッシュボードは次に示されたように見えます：

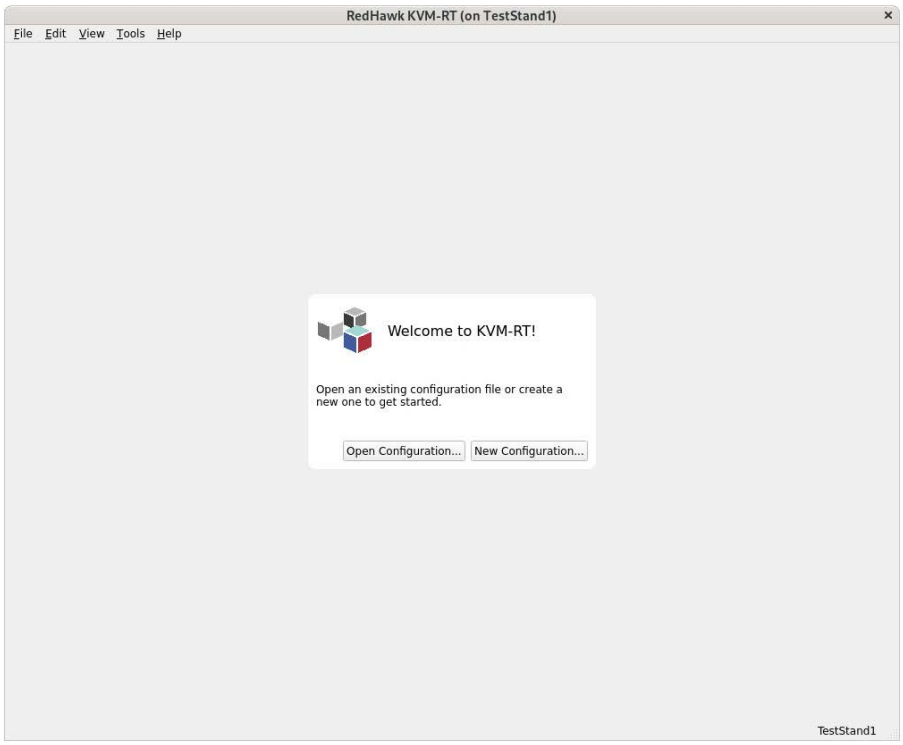


図5-1 Welcome to KVM-RT!ページ

Welcomeページで新しい構成ファイルを既定(**`/etc/kvmrt.cfg`**)もしくは任意で生成することが可能です。続いて構成ファイルを編集し**`libvirt`**からVMをインポートするよう指示されます。VMを構築するためにRedHawk Architectを使用している場合、エクスポートおよび**`kvmrt`**を起動するArchitectのオプションもあります。

KVM-RT構成に含める**`libvirt`**のVMを確認したら、Edit KVM-RT Configurationダイアログページが表示されます。この時に各VMの構成を編集するオプションが得られます。

NOTE

ここで構成を編集しないことを望む場合であっても、選択した構成ファイルにロードされた構成を書き込むApplyボタンを使用する必要があります。そうしない場合、構成情報はロードされますがGUIを終了すると失われます。

VMがインポートされるとダッシュボードは次に示されたもののようになります：

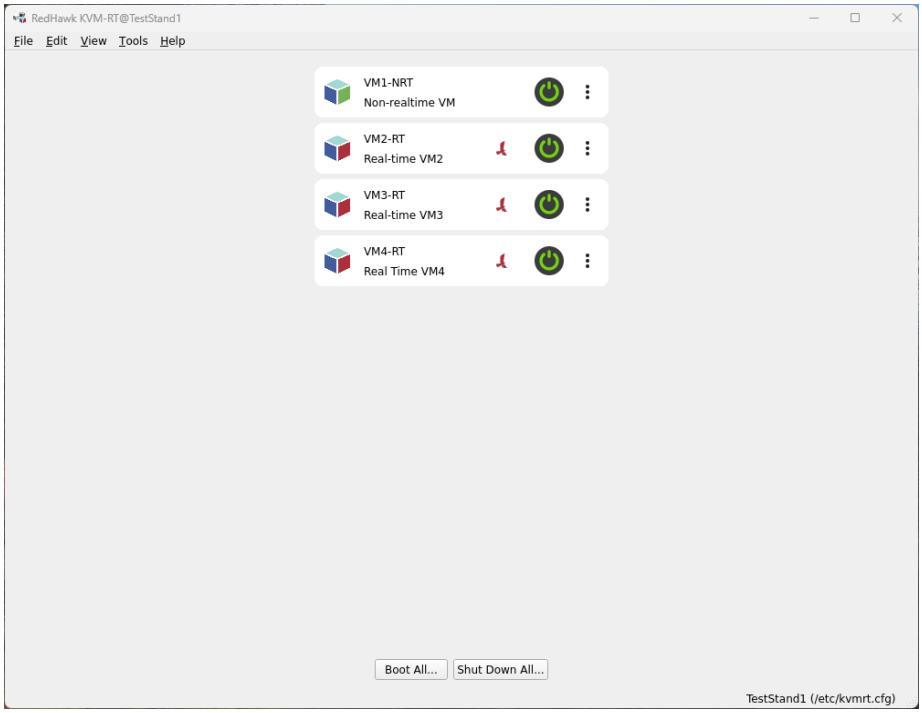


図5-2 VMが4つ構成されたダッシュボードを起動

VMは構成ファイル内で指定されたTitleとDescriptionで確認します。

VMは起動されると表示され、シャットダウンすると電源アイコンが緑からグレーに変わります。

赤いRedHawkアイコンと赤いキューブの側面は、これらがリアルタイムVMであることを示す視覚的な合図です。

ホスト・システムの名称(本例では「TestStand1」)および現在ロードされた構成ファイルはページの右端にあります。

ダッシュボードのツールバー・メニュー

以下はトップ・バー・メニューを介して利用可能な機能の簡単な説明になります。不明瞭な機能のみさらに詳しく説明します。

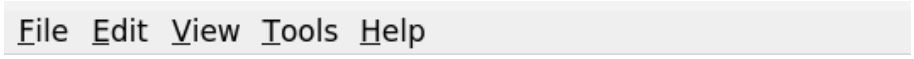


図5-3 ダッシュボードのツール・バー

File

Fileメニューから次の機能が利用可能です：

- New Configuration: 新しい構成ファイルを生成します。
- Open Configuration: 構成ファイルをロードします。
- Close Configuration: ロードされた構成ファイルをクローズします。
- Clone Configuration: 現在ロードされている構成ファイルをコピーします。本機能は構成フィルのバックアップする、またはKVM-RT構成に一時的に変更を行うのに便利です。

Edit

Editメニューから次が可能です：

- Edit Configuration: -KVM-RT構成を変更し、現在ロードされている構成ファイルをAPPLYするために使用することが可能です。

NOTE

変更を認識しAPPLYボタンを有効にさせるにはその領域でTABまたはEnterを入力する必要があります。

複数の**`kvmrt`** GUIのインスタンスを実行することは可能ですが、構成ファイルがロードされたインスタンスが１つのみになるまでどれも構成ファイルへの書き込みは出来ません。

- Lock Configuration: -構成ファイルをロックすると権限のないユーザーのKVM-RT構成の閲覧や編集から防げます。ユーザーは他の機能を実行し続けることが可能です。
- Resynchronize Configuration: - KVM-RT構成ファイルの変更を**`libvirt`**構成へ転送します。これはツールで自動的に行われますが、一部の特殊な状況では同期していない通知が表示される可能性があります。同期を回復するには本機能を使用して下さい。

View

Viewメニューの機能は次のとおりです。これらの機能全てが各VMの電源アイコンの横にある縦3つの点のメニューを介して個々のVMで利用することも可能であることに気を付けて下さい。

- Open All VM consoles: -起動されているかどうかにかかわらず各VMのターミナル・コンソールを開きます。
- Show All VM Runtime Details: -起動されているかどうかにかかわらずKVM-RT構成内の全てのVMのランタイム詳細ページを起動します。

- Close All VM Runtime Details: - KVM-RT構成内の全てのVMのランタイム詳細ページをクローズします。

Tools

次のツールはToolsメニュー・バーから表示させることが可能です。表示するデータのフィルタリングは各ツールのViewメニューを介して得ることが可能です：

- Host Hardware Topology Viewer: -ホスト・システムのハードウェア・トポロジーを表示します。詳細は**`hwtopo-gui(1)`**を参照して下さい。
- Host CPU Status Viewer: -結合されたハードウェア・トポロジー、CPUの状態、IRQ、タスク実行、CPUアフィニティを表示します。詳細は**`cpustat-gui(1)`**を参照して下さい。
- Host Interrupt Count Viewer: -CPU毎の割り込みカウントを表示します。**`interview(1)`**を参照して下さい。

Help

Helpメニューは次の選択肢を提供します：

- User's Guide: -本書を表示します。
- About KVM-RT: -リリースおよび著作権の情報を表示します。

ダッシュボードのVM毎ドロップ・ダウン・メニュー

下図のように電源アイコンの右にある縦3つの点から呼び出されるメニューもあります。リスト表示された機能は個々のVMに適用されます。

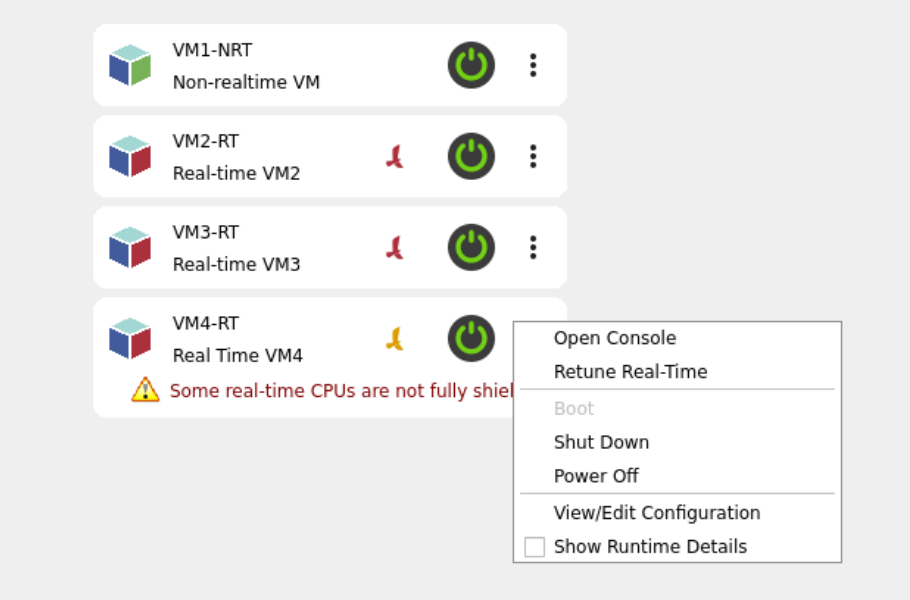


図5-4 警告を表示するVMのドロップ・ダウン・メニューの抜粋

殆どのオプションは説明不要もしくはツール・メニュー・バーの項で説明されています。ツール・メニュー・バーでは構成された全てのVMに適用されている一方、本メニューでは1つのVMにのみ適用されていることに注意して下さい。ここに初めて表示されるもののみ説明します。

Retune Real-Timeメニュー・オプションは最初のVMのチューニングが上図に示すVM4-RTのように乱れている場合に役立ちます。これはシステム管理者がシステムのシールドを変更してVM4-RTに影響を与えた可能性があります。上図で示されたような赤い文字の警告が表示され、赤いRedHawkアイコンがオレンジ色に変わります。

アクティブな構成の仕様に戻るには、オレンジ色のRedHawkアイコンもしくはドロップ・ダウン・メニュー上のRetune Real-Timeをクリックして下さい。システムが再チューニングされるとオレンジ色のRedHawkアイコンが赤色に戻ります。

ダッシュボード・ボタン： Boot All/Shutdown All

ダッシュボード・ボタンのBoot All...とShutdown All...はダッシュボード・ページの下側中央の方に置かれています。

Boot All...ボタンをクリックするとKVM-RT構成内で自動開始するよう構成された全てのVMが起動します。プロンプト・ダイアログは自動開始構成の上書きおよびVMの起動の許可を表示します。

プロンプト・ダイアログはクリーンな起動を選択することも可能です。クリーン起動は最初の起動時にホスト・システムを再チューニングします。クリーン起動において、既に実行中の構成内の仮想マシンが存在せず起動エラーが許容されない場合、起動シーケンスは即時停止されることに注意して下さい。

Shutdown All...ボタンは実行中の全てのVMをシャットダウンします。確認ダイアログがポップアップ表示され、正常にシャットダウンしないVMに対しては強制的に電源オフするオプションも提示します。

確認ダイアログ

ダッシュボードのEditメニューからEdit Configurationをクリックすると次の構成ダイアログがポップアップ表示されます：

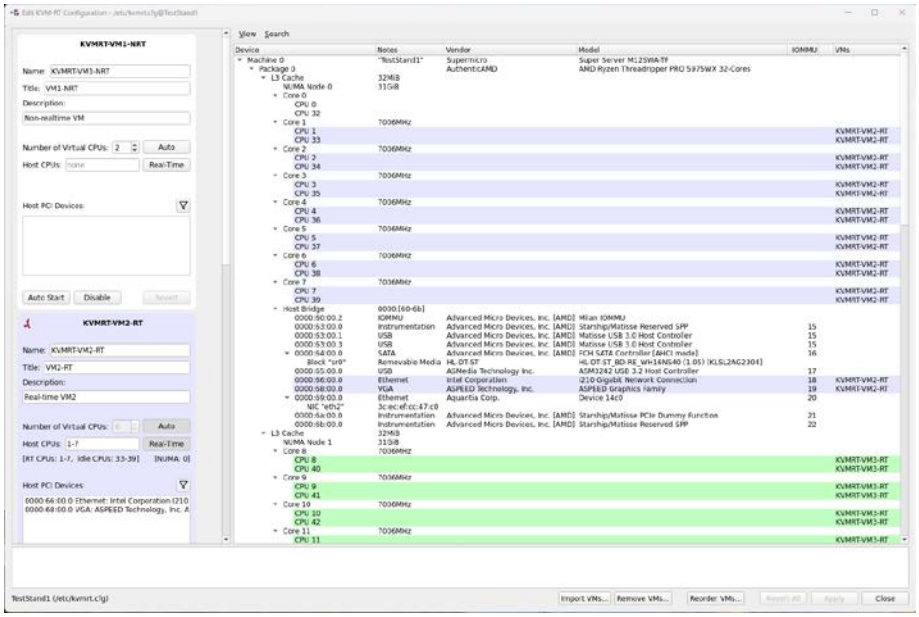


図5-5 構成ダイアログ

次項で構成ダイアログの様々な枠について考察します。

VM構成枠

VM構成セクションの抜粋を以下に示します。構成された2つのVM(1つは非リアルタイム、1つはリアルタイム)が示されています。

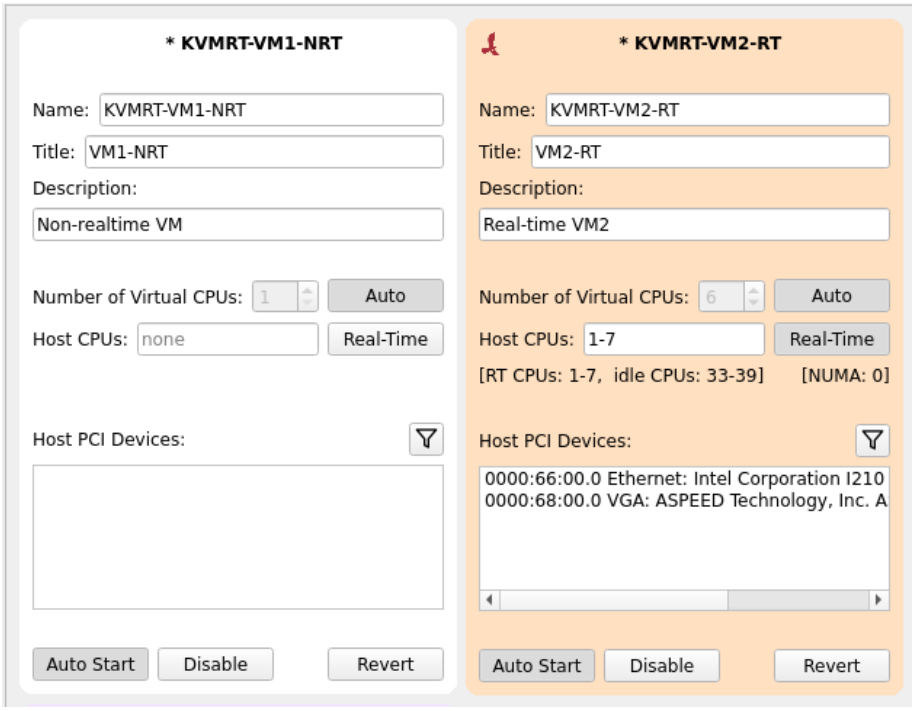


図5-6 2つのVM構成の抜粋

ページ・タイトルの左側にあるアスタリスクは、変更が適用されていないためGUIを終了し変更を放棄することを選択した場合に失うことを示しています。左隅のRedHawkアイコンはリアルタイムVMであることを示しています。

Nameフィールドはlibvirt構成内で一意である必要があり、VMがシャットダウンしている時のみ修正することが可能です。TitleとDescriptionは任意ですが一意的にVMを特定できる必要があります。TitleとDescriptionがダッシュボードに表示されます。

リアルタイムVMに関しては理想のNumber of Virtual CPUsはHost CPUsよりも1つ小さい数です。理想の数以外を要望する場合を除き、自動的に計算させるAutoボタンを有効にして下さい。

非リアルタイムVMに関してはHost CPUsを指定することは可能ですが、未指定のままである場合は割り当てられていないCPU上で動作します。本フィールドはリアルタイムVMに関しては指定することが必須となります。

リアルタイムVMに対してはReal-Timeボタンを有効にする必要があります。

Host PCI Devicesボックスは(パススルーが)割り当てられたPCIデバイスを表示します。

Host PCI Devicesフィールドの右側にある漏斗  のシンボルは右枠の表示をフィルタリングします。これは1つのVMに割り付けられたリソースのみを表示します。システム上の全てのリソース表示に戻るには、右枠にあるViewメニューのExpand Allオプションを使用して下さい。

Auto Startボタンを有効にするとBoot All...ボタンの呼び出す度、また**systemd**サービスの**kvmrt**が有効である場合はホスト・システムの再起動中にVMが起動します。**kvmrt systemd**サービスの詳細については2-2ページの「仮想マシンの起動とシャットダウン」を参照して下さい。

Disableボタンは構成を非アクティブの状態を維持するために使用することが可能です。無効なVMはダッシュボード・ページに表示されません。

最後にApplyボタンを使って構成を書き出した後に行った変更を廃棄するためにRevertボタンを使用することが可能です。

リソース・リスト枠

この枠はホスト・システムのリソースを一覧表示します。Viewメニューを介して表示する情報を制御することが可能です。

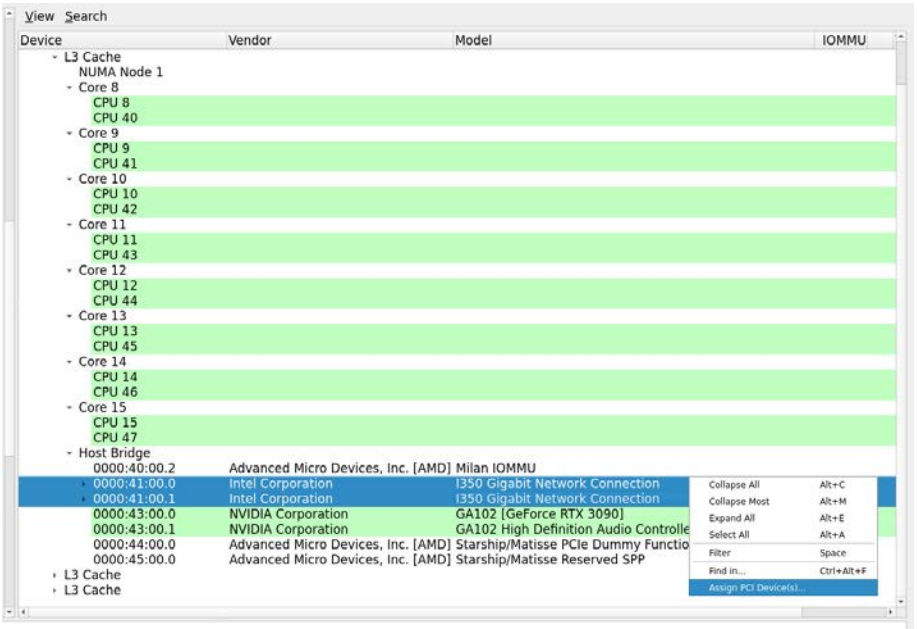


図5-7 VMへのデバイスの割り付け表示の抜粋

VMにリソースを割り当てるには、リソース場を右クリックして上図に示すメニューを呼び出します。続いてAssign PCI Devicesオプションをクリックして下さい。

複数のリソースが同じIOMMUグループ内にある場合、全てを追加する必要があります。これらは1つずつ追加(警告あり)、もしくは上図のようにSHIFTキーで全てのリソースを反転させて1度に全てのエントリを割り当てるために使用することも可能です。

上部のViewメニューは表示するリソースのフィルタリングを提供します。

上部のSearchメニューは表示されるリソースの文字列検索が可能です。

診断ボックス

構成ダイアログの下部にあるボックスは構成内のエラーおよび警告を表示します。構成への変更はENTERまたはTABキーを使用した場合のみ認識されることに注意して下さい。

以下はいくつかの構成上の問題がある構成です：

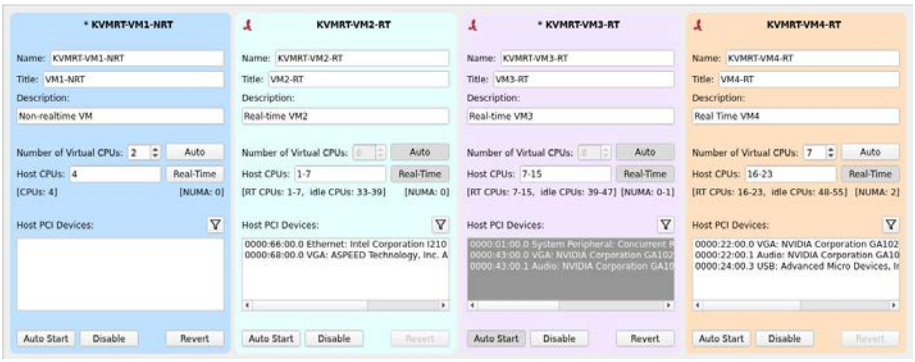


図5-8 構成ミスの例

上の例での構成ミスは次のとおりです：

- 非リアルタイム仮想マシンVM1-NRは、リアルタイム仮想マシンVM2-RTで予約されたCPU 4を使用しています。
- リアルタイム仮想マシンVM2-RTとVM3-RTは両方ともCPU 7を割り当てています。これらはNode 0も共有しています。
- RCIMデバイスはNode 1の配下にあるVM3-RTに割り当てられている一方、RCIMはNode 4の配下にあります。

以下は上記の構成ミスに関する警告とエラーを表示する診断ボックスの抜粋です：

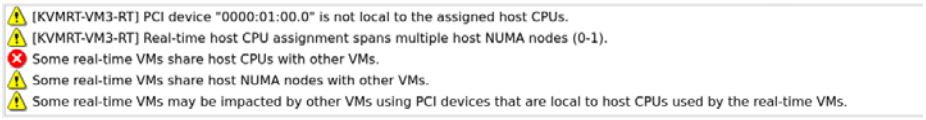
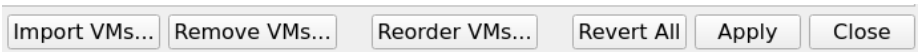


図5-9 警告とエラーを表示する診断ボックス

エラーは黒で強調表示されVM構成枠とここには示されていないリソース・リスト枠の両方で表示されることに注意して下さい。

様々なボタン

構成ダイアログの右下で以下のボタンを見ることが出来ます：



Import VMs: **libvirt**から新しいVMをインポートするために使用します。VMはKVM-RT構成内で動作するよう構成する必要があります。

Remove VMs: KVM-RT構成からVMを削除するために使用します。これは**libvirt**からは削除しないことに注意して下さい。

Reorder VMs: VMを表示および起動/シャットダウンする順番を再整理します。

Revert VMs: 最後に変更を適用してからの全てのVMへの全ての変更を放棄します。

Apply: 変更を構成ファイルに書き出します。

6
仮想マシンの時刻同期

chronyはNTPの万能な時刻同期実装です。これは幅広い条件でも正常に機能するように設計されており、仮想マシンで実行することが可能です。仮想マシン上のchronyシステムを構成及び開始する方法について具体的な手順がここに含まれています。ホスト・システムは既に時刻同期が構成されているものと想定します。詳細については**chronyd(1), chrony.conf(5)**およびオンライン・ドキュメントを参照して下さい。

複雑なアプリケーションは、2つ以上のVMまたはホストとの間で同期される時刻に依存する可能性があります。また、これはリアルタイムVMの性能の問題を解析、またはシステムの問題をデバッグするためにRedHawkのトレース機能を使用する場合、仮想ゲストの時刻はホストと同期することが必須となります。

chrony実行の手順

仮想ゲストの時刻クロックを同期するための様々なテクニックがありますが、**ptp_kvm**モジュールを介してchronyと同期するkvm_clockを推奨します。

ptp_kvmを使用するために**chronyd**を構成する過程は、ベース・ディストリビューションにより若干異なります。

ベース・ディストリビューションとしてUbuntuを使用している場合、次の設定を使用して下さい：

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

Rocky互換ディストリビューションを使用している場合、次の設定を使用して下さい：

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

次の手順は仮想ゲストでchronyを構成するのに役に立つはずです。適切な上記のディストリビューションの設定を以下の変数設定に置換して下さい。

1. インストールがまだの場合、chronyをインストールして下さい。

   Rocky互換システム：

   ```
   dnf install chrony
   ```

   Ubuntuシステム：

   ```
   apt install chrony
   ```

2. 起動時にptp_kvmモジュールをロードして下さい。

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

3. 構成ファイル内に「refclock」「server」「pool」「peer」を言及するいずれの行があるかどうかを確認してください。もしある場合、ファイルを編集しコメント・アウト(先頭に#記号を置く)して下さい。

   ```
   egrep 'refclock|server|pool|peer' $conf
   [ "$?" = 0 ] && vi $conf
   ```

4. 「refclock」を構成して下さい。

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

5. /etc/sysconfig/networkファイル内のPEERNTPがある全ての行をコメントアウト(先頭に#を置く)し、PEERNTP=noを付け足して下さい。

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

6. 適切な$driftファイルを削除して下さい。

   ```
   rm –f $drift
   ```

7. 適切なchronydサービスを有効化しますが、開始しないで下さい。

   ```
   systemctl enable $service
   ```

8. 新しい構成でクリーン・スタートするため再起動して下さい。

   ```
   reboot
   ```

7
仮想環境でのI/Oデバイスの利用

本章では、Linuxクイック・エミュレータ(QEMU: Quick Emulator)とカーネル・ベース仮想マシン(KVM: Kernel-Based Virtual Machine)の前後関係にある仮想化の重要な側面について取り上げます。一般的なデバイスを構成した場合に性能や低レイテンシーについて的確な情報に基づく決断が行われることが可能となるように仮想化内部の基本的な技術の更なる理解をユーザーに提供することを目的としています。

本章は、準仮想化、デバイス・エミュレーション、PCIパススルーを含む異なる仮想化の技術を取り上げ、これらは異なる性能のレベルおよびホスト・ハードウェアの相互作用を提供します。これはデバイスの種類を理解する根幹を提供します。

3種類の一般的なデバイス(ネットワーク、ストレージ、グラフィックス)についても各々仮想および物理ハードウェアのソリューションに分離可能ないくつかの実装を含めて説明されています。

概要

Linuxでは、QEMU/KVMがハイパーバイザーを構成します。QEMUはCPU、メモリ、ネットワーク・カード、ディスク・コントローラ、ディスプレイ・アダプタその他を含むハードウェア・エミュレーションを提供します。

KVMはIntel VT-xやAMD-vのようなハードウェア仮想化機能をユーザー空間に公開することで仮想化機能の中核を提供します。これは仮想マシンを管理するlibvirtのようなライブラリを当てにしています。KVMはCPUの仮想化も処理しており、仮想マシンがホストの物理CPU上で直接命令を実行することを許可します。

仮想化技術

仮想マシンにI/Oデバイスを利用可能にさせるための3つの一般的な技術(デバイス・エミュレーション、準仮想化、PCIパススルー)があります。

デバイス・エミュレーション

完全なデバイス・エミュレーションはゲスト・オペレーティング・システムが使用するハードウェア・デバイスを装うことで機能します。これはゲストOSが実際のハードウェアと接触していると信じている状態で実行することを可能にします。

デバイスの割込みが模擬されQEMUがそれらを傍受して処理することを除き、ゲストOSは標準的なI/O操作を利用してI/Oデバイスに接触します。この動作を処理するために頻繁にホストOSのCPUに転送、またはエミュレータにより直接操作されます。

デバイス・エミュレーションでは、割り込みが通常は物理システム上を対象とするハードウェア・デバイスの代わりにソフトウェアにより処理されるため、幾分パフォーマンスのオーバーヘッドを招きます。準仮想化がオプションではない、もしくはドライバーが利用可能ではない古いゲスト・オペレーティング・システムでこの方式は役立つ可能性があります。

準仮想化

準仮想化はゲストOSが仮想環境で実行していることを認識するように変更することで仮想マシンの性能が向上します。基本的なハイパーバイザーを認識することにより、ゲストOSは特定の動作に対してエミュレーション層を迂回およびハイパーバイザーと直接対話することが可能です。

virtioはQEMU/KVMが提供する準仮想化サポートとのインターフェースです。これは規格化されたデバイス用APIを提供します。ゲストOSは様々なデバイスのためのvirtio-pciのようなドライバへのサポートが必要になります。例として、ネットワーク(VirtioNet)、ストレージ・コントローラ(Virtio Block、Virtio SCSI)、グラフィクス(Virtio GPU)。

virtioドライバはゲストOS内のユーザー・プロセスからのI/O要求受け入れ、それらのI/O要求をQEMU内の適切なvirtioデバイスへ転送、完了要求をvirtioデバイスから回収の責任を負います。

virtioデバイスはI/O要求を対応するvirtioドライバから受け取り、それらのI/Oリクエストをホストの物理ハードウェアへ解放し、結果をvirtioドライバで利用できるようにします。

現在のvirtio内で最適化を利用すると著しい仮想マシンの性能向上がある一方、依然として多少のオーバーヘッドはあります。準仮想化を利用する全ての仮想マシンはホストのリソースがやはり競合し、それらは完全な専用物理ハードウェアとPCIパススルーを利用するVMを比較した場合にやはりある程度の予期可能な遅延となる可能性があります。

参考文献：

https://blogs.oracle.com/linux/post/introduction-to-virtio

PCIパススルー

PCIパススルーは仮想マシンI/O操作を実行中にホスト・カーネルの関与を減らすことで最高の性能を提供します。PCIパススルーはシステムによってはいくつかのカーネル起動パラメータ構成を必要とする可能性があります。付録Cの「PCIパススルー起動パラメータ」を参照して下さい。

PCIパススルーのいくつかの恩恵にはホストのメモリや他のVMからの安全な分離に加え、ベアメタルに近い性能を提供する直接ハードウェア・アクセスを含んでいます。

PCIパススルーの欠点の1つは完全なリソース割り当てとなります。本ソリューションはリソース制限に起因し十分な調整ができず、ホストや他のゲストが特定のVMに割り当てたデバイスを使用することができません。これは欠点でありますが、PCIパススルーは前述の手法を上回る最高のリアルタイム性能を提供します。

PCIパススルーがどのように動作するのかを理解するには、ホストOSが仮想マシンに割り当てる物理デバイス用に環境を準備するメカニズムについて説明する必要があります。これらのメカニズムにはIOMMUとVFIOを含んでいます。

IOMMU

入出力メモリ管理ユニット(IOMMU: Input-Output Memory Management Unit)は、デバイス・メモリ・アクセスの安全かつ高効率なマッピングを提供します。これはVMが割り当てられたデバイスのメモリのみにアクセス可能となるようなメモリ分離を提供します。IOMMUはデバイスで生成された割込みを適切にVMのCPUにルーティングされることを確保するために割込みの再マッピングも提供します。

参考文献：

https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/7/html/virtualization_deployment_and_administration_guide/sect-iommu-deep-dive#sect-iommu-deep-dive

VFIO

仮想機能I/O (VFIO: Virtual Function I/O)ドライバは、IOMMUが保護した安全な環境のユーザー空間にデバイス・アクセスを公開するためのIOMMU/デバイスに依存しないフレームワークです。これは物理ディスク・デバイスに直接接触およびホストのカーネルを回避するためにVMがデバイス・ドライバを利用することを許可します。

必要なメモリ・マッピングを設定しホストからの分離を提供することでVMにPCIデバイスを安全に割り当てるため、VFIOはIOMMUに依存します。IOMMU分離は常に個別のデバイス・レベルではありません。デバイス、相互接続、IOMMUトポロジーのプロパティは分離をデバイスのグループに変えることが可能です。これはVFIOはIOMMUグループ・レベルで動作し、同じIOMMUグループ内の全てのデバイスは同じVMにパススルーされる必要があることに注意する必要があります。

参考文献：

https://docs.kernel.org/driver-api/vfio.html

ネットワーク

仮想マシンは仮想および物理ネットワーク・デバイスを使用することが可能です。仮想ネットワーク・デバイスはNATもしくはMacVTapとして構成することが可能です。物理ネットワーク・デバイスはホストからパススルーされます。

仮想ネットワーク

仮想ネットワーク・デバイスは、virtio, e1000e, rt18139などの仮想デバイス・モデルのNATもしくはMacVTapとして構成することが可能です。

NAT

デフォルトで、libvirtの設定は仮想マシンのネットワーク構成として転送付きネットワーク・アドレス変換(NAT: Network Address Translation)を使用します。ホスト・システムは、いずれの物理インターフェースの追加なしで分離されたブリッジ・デバイス「virbr0」を所有します。

libvirtはゲストとの間のトラフィックを管理するためにiptablesルールを追加します。ゲストからの送出接続は仮想ネットワークをパススルーし、ホストIPアドレスを介して宛先に転送されます。外部接続はゲストとの通信が開始できませんが、ゲストはvirbr0を使ってホストだけでなく他のゲストから着信する通信も受け入れることが可能です。

ゲストがブリッジの外側からアクセス可能にするため、明示的なポート転送付き静的宛先ネットワーク・アドレス変換(DNAT: Destination Network Address Translation)をホスト上に構成する必要があります。

参考文献：

https://wiki.libvirt.org/Networking.html
https://wiki.libvirt.org/VirtualNetworking.html

MacVTap

MacVTapは仮想化ブリッジ・ネットワーク用に簡素化されたソリューションを提供します。これは、**`brctl(8), nmcli(1)`**もしくはLinuxのEthernetブリッジを生成する他の手法を介して生成されるネットワーク・ブリッジと混同させるものではありません。

MacVTapはMacvlanドライバとタップ・デバイスを混ぜ合わせています。これはKVM/QEMUで直接使用されるキャラクタ・デバイスと共に物理インターフェースの上位に生成されたインスタンスです。使用するゲスト用に同じEthernetセグメント上の独自のMACアドレスを提供することに加え、既存のホスト・ネットワーク・インターフェースを拡張したものです。

ゲストはホストが接続されている同じスイッチ上に表示されます。これは着信接続がゲストに到達できないNATネットワークを勝る長所を提供します。MacVTapを利用すると着信接続はゲストに到達することが可能です。

MacVTapはエンドポイント間通信を定義する3つのモードのうちの1つを設定することが可能です：

Virtual Ethernet Port Aggregator (VEPA)
Bridge
Private

最新のlibvirtの設定はブリッジ・モードを使用しており、ここで説明されるモードとなります。

ブリッジは全てのエンドポイントが互いに直接接続されることを許可しています。MacVTapを使用するゲストはホストが使用する同じネットワーク上で通信することが可能である一方、外部ネットワークを通る経路なしで同じMacVTapブリッジ接続を使用する他のゲストと直接フレームを交換することも可能です。外部ネットワークの速度は物理ネットワークの設定に制限されますが、ゲスト間の転送速度はNATネットワークが提供するものと同等です。

MacVTapの制限の1つは、KVMホストといずれのKVMゲストとの間で容易にネットワーク接続を有効化できないことです。

1つの可能な回避策はKVMホスト内に複数のネットワーク・インターフェースを持ってゲストと通信する他のホスト・インターフェースを構成することです。

参考文献：

https://virt.kernelnewbies.org/MacVTap
https://developers.redhat.com/blog/2018/10/22/introduction-to-linux-interfaces-forvirtual-networking#macvlan
https://docs.kernel.org/networking/tuntap.html
https://wiki.libvirt.org/TroubleshootMacvtapHostFail.html

仮想ネットワーク・デバイス・モデル

仮想ネットワーク・デバイス・モデルはゲスト内部のPCIデバイスとして存在する仮想化ネットワーク・カードです。これらのデバイス・モデルは前述のとおりNATもしくはMacVTapのバックエンド設定を利用します。

これらの一般的なデバイス・モデルはMacVTapおよびNATの両方のバックエンド設定で利用可能です：

virtio
e1000e
rtl8139 (ハイパーバイザーのデフォルト)

virtioはvirtioをサポートするゲスト・オペレーティング・システムで動作する準仮想化ドライバです。ゲストは仮想マシン内で実行していることを知っており、ハイパーバイザーの直接呼出しを許可します。本手法では転送速度が大幅に向上し完全にエミュレートされたネットワーク・デバイスは必要ありません。

残りのデバイス・モデル(e1000eとrtl8139)は、それぞれIntelおよびRealtekのネットワーク・カードをエミュレートします。ゲストOSがサポート可能なネットワーク・カードの制限にもよりますが、これらのオプションのいずれかはvirtioの代わりに必要となる可能性があります。

各デバイス・モデルに関するゲスト内に表示されるPCIデバイスの例：

virtio:
Ethernetコントローラー： Red Hat, Inc. Virtio 1.0 ネットワーク デバイス (rev 01)

e1000e:
Ethernetコントローラー： Intel Corporation 82574L ギガビット ネットワーク接続

rtl8139:
Ethernetコントローラー： Realtek Semiconductor Co., Ltd.
RTL-8100/8101L/8139 PCIファスト イーサネット アダプタ (rev 20)

デバイス・モデルの性能比較

仮想ネットワーク・デバイスを構成する場合、性能の違いに注意することが重要です。

各デバイス・モデルはNATとMacVTapの両構成で同じブリッジ上の2つのゲスト間の通信を使って比較しました。サンプリングは測定間隔が1秒の5分間(60秒/分 x 5分 = 300サンプル)で **`iperf3`** TCPテストを使って行われました。

テストはゲストOSでストレスありとなしの両方で完了しています。次はサンプリング中に測定した平均のギガビット/秒になります。

デバイス・モデル	ストレスなし (Gbps)	ストレスあり (Gbps)
virtio	10.3	4.44
e1000e	2.12	1.12
rt18139	0.55	0.44

デバイス・モデルを直接比較する場合、MacVTapとNAT構成間の速度は殆ど同等であるため結果は見分けられません。

ホスト外部と通信するMacVTapが構成されたゲストのテスト結果は、ユーザーのネットワーク制限次第です。virbr0デバイスの外部にデータ送信するNATが構成されているゲストに対しても同じと言うことが可能です。

これらの測定はいずれのリアルタイム機能も利用していないシステム上で行われたことに注意してください。ネットワークの割込みとiperfタスクの両方をシールドするRedHawkゲストと共にホスト上でKVM-RTを使用するとネットワークの性能と一貫性の劇的な向上を得られます。

参考文献：

https://wiki.libvirt.org/Virtio.html

物理ネットワーク

物理ネットワーク・カードは仮想マシンに専用のネットワーク・デバイスを供給するために使用することが可能です。シングル・ルートI/O仮想化(SR-IOV: Single Root I/O Virtualization)テクノロジーは物理ネットワーク・カードをPCIバス上に(仮想機能と呼ばれる)複数、単一デバイスとして表示させることが可能です。あるいは、ネットワーク・カード全体を完全なPCIパススルーを介して1つの仮想マシンに専念させることが可能です。

SR-IOV

シングル・ルートI/O仮想化(SR-IOV)はシングル・ルート機能を複数、単独、物理デバイスとして表示することを許可します。ネットワーク・カード上の1つのEthernetポートをそれぞれ独自の構成空間、バス・アドレス、IOMMUグループを持つマルチ機能として表示されます。マルチ機能はPCIデバイス情報を表示するコマンドの出力で個別のデバイスとして表示されます。

NOTE

KVM-RTの認証済みプラットフォームはSR-IOVをサポートします。非認定システムのユーザーは、そのシステムが本機能をサポートしておりBIOSでそれが有効化されていることが確かであるかどうかを対応するメーカーの資料を確認する必要があります。

NOTE

ネットワーク・カードもSR-IOV対応である必要があります。SR-IOV機能を確認するにはメーカーの仕様を参照して下さい。

SR-IOVが有効なデバイスは2つのPCI機能、物理機能(PF)と仮想機能(VF)を使用します。

物理機能

物理機能(PF)はSR-IOV機能を含んでいる完全なPCIeデバイスです。これはSR-IOVの機能性を構成および管理もしながら通常のPCIデバイスとして動作します。

仮想機能

仮想機能(VF)は物理機能から派生されます。仮想機能はI/Oを処理する簡素なPCIe機能です。1つの物理Ethernetポートは最大でデバイスの制限まで複数の仮想機能にマッピングすることが可能です。

前項で説明した仮想ネットワーク方式とは異なり、仮想機能をゲストに利用される前にいくつかの設定が必要となります。必要であれば付録Bの「SR-IOVの設定」を参照して下さい。

仮想機能が設定されたら、続いて他のPCIeデバイスと同じようにVFに関連付けられたPCIeデバイスをゲストにパススルーして下さい。

同じ物理機能の仮想機能を利用する同じホスト上のゲストは、スイッチへの転送およびカードへの返答がないためネットワーク制限を超える転送速度の恩恵を受けます。

仮想機能を使用するゲストはホストの対応する物理機能と通信することはできません。ゲストーホスト間の通信を有効にするには別のインターフェースを構成する必要があります。

参考文献：

https://www.intel.com/content/www/us/en/developer/articles/technical/configure-sr-iovnetwork-virtual-functions-in-linux-kvm.html
https://access.redhat.com/documentation/enus/red_hat_enterprise_linux/7/html/virtualization_deployment_and_administration_guide/sect-pci_devices-pci_passthrough

SR-IOV機能 vs MacVTap機能

仮想機能(SR-IOV)とMacVTapの類似点は性能において比較が必要なことです。

同一ゲストと10Gb Ethernetカードを持つ2つのKVM-RTホストが、他のネットワークから分離された専用のEthernet接続を通して互いに通信するように設定されました。

ゲストは最初に10Gbネットワークの1番目のインターフェースを使ってvirtioデバイス・モデルとブリッジ・モードのMacVTapを通して通信しました。

データが収集された後、仮想機能は同じインターフェースと仮想機能を使用するためにPCIパススルーを利用するゲストを構成しました。

往復のネットワーク試験中、MacVTapは各インターバルでGBのデータを転送した場合にSR-IOVよりも平均で34.6%以上のカーネルCPU時間、各インターバルでMBのデータを転送した場合に平均で69.3%以上のカーネルCPU時間が表示されました。

仮想ネットワーク方式と比較した場合、CPU使用がより低いことに加え、SR-IOVはスループットとスケーラビリティにおける利益も提供します。

ネットワークPCIパススルー

完全なネットワーク・カードのパススルーは、ホストと他のゲストの両方からその利用を分離している間はネットワーク・カードの全ての機能性をゲストへ提供する最高の方法です。

本方式はカードの他のインターフェース(ポート)がゲストの使用に対して構成されることを許可します。本方式はネットワーク性能がリアルタイム・アプリケーションに危機的であるどの事例に対しても最良です。

ホスト・マザーボードのPCIeスロットの制限を考えると本メソッドはゲストとネットワーク・インターフェースの最大数に対して最小の拡張性です。

ストレージ

ストレージ・オプションはqcow2やrawイメージのような仮想ディスク・フォーマット、同時に物理ディスク・パーティションの割り当てや物理ディスク・コントローラのPCIパススルーを含む物理ディスク構成を含んでいます。

仮想ディスク

仮想マシンは物理ディスクとして仮想ディスクイメージを扱いますが、I/Oは実際にはハイパーバイザー(QEMU/VM)を介してホスト・システムにより管理されます。

VMがI/O要求を仮想ディスクに行う場合、ハイパーバイザーはその要求を傍受し、ホストのファイル・システムに要求を転送する前に使用されているディスク・フォーマットに適切な方法で変換します。

変換は使用中の仮想ディスク・フォーマットによって変わります。

qcow2イメージでは、追加のメタデータ、スナップショット、圧縮の管理が伴うために幾分のオーバーヘッドがあります。

しかしながら、rawイメージはディスク・セクタが1対1のマッピングとなっています。これはハイパーバイザーが対応するrawイメージ・ファイルのオフセットに直接要求を変換することを許可します。

ホストのファイル・システムに転送されたら、I/O要求は標準ファイル操作として出現します。ファイル操作の複雑さはqcow2についてはイメージの構造や仮想ディスクの状態によって変わります。rawイメージに関する処理は所定のオフセットでの素直な読み取りまたは書き込みとなります。

ホストのカーネルはI/O操作をスケジュールし、ディスクの割込みは通常のホスト・システムであるように生成されます。

完了するとハイパーバイザーは結果をVMに送信します。VMは仮想ディスク上のI/O操作が完了したかのようにそれを解釈します。

qcow2イメージ

QEMU Copy-on-Write 2 (qcow2)イメージはいくつかの機能を提供します：

- 低密度割り当て。ホスト上のディスク空間は動的に割り当てられ、保存された範囲のデータに必要な空き領域のみを占有することが可能です。
- 圧縮。ホストのディスク使用量を削減するためにイメージを圧縮することが可能です。
- スナップショット。VMの状態を保存し、後で元に戻すことが可能です。

前述のようにqcow2が提供する利益は性能に影響する幾分のオーバーヘッドも取り込みます。

rawイメージ

rawイメージはシンプルで、生成する際に割り当てられた領域全体を占有する体系化されていないディスク・イメージです。このシンプルさのため、qcow2イメージと比較した場合にrawイメージはより良い読み取り/書き込み性能を提供します。

仮想ディスクの欠点

仮想ディスク・イメージは、物理ディスクをVMに割り当てるのと比較した場合に移動が容易で必要なリソースは少ない一方、VMとホストのディスク性能に影響を及ぼす欠点があります。

仮想ディスク・イメージはホストのファイル・システム上に置かれます。これはVMのディスクI/O操作がハイパーバイザーを介して通信しホスト・カーネルによって完了する必要があります。

VMのためのスケジューリングI/O操作とホストのためのI/O操作の間で競合があります。これはVMとホストの両方において読み取りと書き込みの操作にマイナスの影響を与える可能性があります。複数のVMを管理する場合は(各々ホストが代わりにI/O操作をスケジュール必要があるために)性能低下がより明白になります。

仮想ディスク・イメージでは、ダーティ・ページを物理ホスト・ディスクに書き込むためのフラッシュはホストに依存します。ホストが同期の実行を決定した場合に仮想ディスクの書き込み性能は低下します。

仮想ディスク・イメージを備えるVMは高速キャッシュ読み取り速度の恩恵を受けることが可能である一方、メモリからページを退去するタイミングでホストが支配します。これは読み取り性能が著しく低下することが示されています。

参考文献：

https://docs.redhat.com/en/documentation/red_hat_virtualization/4.3/html/technical_reference/qcow2
https://github.com/qemu/qemu/blob/master/docs/interop/qcow2.txt
https://www.qemu.org/docs/master/system/images.html
https://docs.kernel.org/admin-guide/mm/concepts.html#page-cache

物理ディスク

物理ハードウェア・ストレージはパーティション割り当ておよびPCIパススルーを通して仮想マシンに割り当てることが可能です。

パーティションの割り当て

ホスト・ディスク上の物理ディスク・パーティションがVMに割り当てられる場合、I/O要求はホストのディスク・パーティションに直接マッピングされ、仮想ディスク・イメージのファイル・フォーマット用にI/O要求を変換するハイパーバイザーの必要性はもはやありません。

VMがI/O要求を行う場合、ハイパーバイザーはリクエストを傍受し割り付けられたホスト・ディスク・パーティションに直接リクエストを渡します。

仮想ディスク・イメージと同様、I/O要求は標準ファイル操作として常にホスト・カーネルにより処理されますが、中間のファイル・システム・レイヤーはありません。

完了したらディスク割込みが生成され、操作が完了したことをホストへ信号を送信します。続いてハイパーバイザーはVMに信号を送信します。

パーティション割り当ては、複雑さやオーバーヘッドの低減、ディスク性能の改善のような仮想ディスク・イメージと比較して多くの恩恵を提供します。PCIパススルーと比較した場合、欠点は存在します。パーティション割り当てはI/O操作を実行するためにホスト・カーネルのサポートが常に影響し、VMとホストの両方でI/Oのスケジューリングの競合により顕著な性能への影響があります。前項で説明したホスト・キャッシュのフラッシュやメモリからのページの強制撤去の影響は、ここでもディスク・パーティション割り当てに当てはまります。

パーティション割り当てはパススルーとは見做されないことに注意してください。パーティション割り当ての特徴は、VMからの信号を傍受するハイパーバイザーの代わりに物理ディスク間のI/O操作を処理するために常にホスト・カーネルとディスク・ドライバに依存することです。

ストレージPCIパススルー

PCIパススルーはVM I/O操作の実行中においてホスト・カーネルの関与を減らすことにより最高の性能を提供します。

IOMMUが有効なシステムでは、ディスク・コントローラをゲストVMに直接渡してゲストVMのディスク・ドライバで制御することが可能です。この操作はVMへの完全に直接的なデバイス・アクセスを提供するためにホストのディスク・ドライバがアンロードされ、代わりにVFIOドライバをロードしていることで可能となります。

PCIパススルーの欠点の一つは完全なリソース割り当てです。ホスト・システムのトポロジーによっては、ディスク・コントローラのパススルーはホストがそのコントローラ下にあるディスクへのアクセスが出来なくなる可能性があります。これは、複数のディスクがそのコントローラに関連するIOMMUグループの一部である可能性があり、パススルーをVMに全ディスク割り当てたSATAコントローラでより明白です。ディスク・デバイスをVMが直性制御することを可能にし、VMのI/O操作の処理からホストを解放することを含むPCIパススルーの性能の優位性は、完全なリソース割り当てのコストを上回ります。

グラフィックス

仮想環境内の仮想グラフィクスの実装はVGA, QXL, Virtio GPUを含んでいます。VNCとSPICEは仮想グラフィクス・ディスプレイを供給するために使用されるディスプレイ・プロトコルです。完全なGPU PCIパススルーはホスト上の物理グラフィクス・カードへの完全なアクセスを提供し、最高のグラフィック性能と低レイテンシーをもたらします。

VGA

VGAは、QXLやVirtio GPUのようなより高度な仮想グラフィック・アダプタ用の特定のドライバを持たない可能性のある特に古いOSバージョンを含め幅広い互換性を提供します。

QEMUはbochs-drmドライバを使用するBochs VESA BIOS Extensions (VBE)を備えた簡素なVGAカードをエミュレートします。

エミュレートされたVGAグラフィック・アダプタは低解像度グラフィック・モードをサポートし、VBEを使ってより高い解像度および色深度を提供することが可能です。

本デバイスは簡素なフレームバッファを使用します。ハイパーバイザー(QEMU)はこのフレームバッファを読み取り、スクリーン上にそれを表示またはVNCのようなディスプレイ・プロトコルを介してリモート・クライアントに送信します。

エミュレートされたVGAはハードウェア・アクセラレーションをサポートせず、全てのグラフィック・レンダリングは高解像度表示または視覚的に激しいアプリケーションを制限する可能性のあるGPUで処理されます。

全体的に本書で説明している他の方式と比較した場合に性能は低くなりますが、より幅広い互換性を提供します。例えば、QXLとVirtio GPUの両方でホストGPUを使用せずとも一定の2Dアクセラレーションを提供する最適化を備えていますが、グラフィカル・デバイスのための準仮想化ドライバをゲストOSがサポートし使用することが必要となります。

QXL

QXLはSPICEプロトコル(後述のSPICEを参照)を利用した2Dアクセラレーションをサポートする準仮想化グラフィカル・アダプタです。

ホスト・マシンの能力を活用しVM内のグラフィクス操作効率を向上することで、従来のエミュレートされたグラフィクスよりも優れた性能を提供するよう設計されました。このグラフィカル・アダプタはVM内で準仮想グラフィクス「qxl」ドライバを利用し、QEMUを介してバックエンドのQXLデバイスと通信します。ゲストのグラフィカル操作をQXLデバイスが効率よく処理することが可能なコマンドに変換するためにゲストOSはこのqxlドライバが必要です。

QXLはVMの中で動作し、描画コマンド、画面更新、およびその他のグラフィクス関連タスクを操作します。エミュレートされたグラフィクス(VGA)とは違い、QXLは仮想環境の中で動作するよう設計されており、ホスト・システムとSPICEサーバーにタスクをオフロードすることが可能です。これはホストCPUとメモリ使用量に関して更に効率を良くします。また、最大3840x2160の単一でのモニタ解像度をサポートします。

Virtio GPU

Virtio GPUは最新の準仮想化グラフィクス・ドライバです。幅広いVirtioフレームワークの一部であるVirtio GPUは、ゲストOSに対して簡易化されたデバイス・インターフェースを見えるようにすることで効率の良い仮想化を提供します。これはエミュレートされたハードウェア全体に通常関連するオーバーヘッドを縮小します。Virtio GPUは完全なデバイス・エミュレーションやQXLのような古い準仮想化ソリューションと比較して優れた性能および柔軟性を提供します。

ゲストOSは準仮想化ドライバのサポートが必要になります。ドライバは軽量かつ仮想環境向けに最適化されている一方、古いオペレーティング・システムではサポートされていない可能性があります。このような状況では完全なエミュレーションが必要となります。

Virtio GPUは2Dアクセラレーションをサポートします。OpenGLを介した3Dアクセラレーションのサポートは現時点では安定していません。より完全なグラフィカル機能に関してはPCIパススルーを推奨します。

参考文献：

https://www.kraxel.org/blog/2019/09/display-devices-in-qemu/

ディスプレイ・プロトコル

本項ではVNC (Virtual Network Computing)プロトコルとSPICE (Simple Protocol for Independent Computing Environment)プロトコルはについて解説し比較しています。

VNC

VNC (Virtual Network Computing)は、VMホスト上で動作中のサーバーとユーザーのシステム上で動作中のクライアントとの間で画面の更新、キーボードやマウスのイベントを転送することによりGUIへのリモート・アクセスを提供するクロス・プラットフォーム・プロトコルです。

これはSPICEのようなより高度なプロトコルと比較して簡素なリモート・フレームバッファ(RFM: Remote Framebuffer)プロトコルがベースとなっています。

VNCは仮想マシン内で動作する古いオペレーティング・システムに対して有益なSPICEよりも幅広い互換性を提供しますが、SPICEと同様には機能しません。

参考文献：

https://web.mit.edu/cdsdev/src/howitworks.html

SPICE

SPICE (Simple Protocol for Independent Computing Environment)プロトコルは仮想化デスクトップへのリモート・アクセスを処理するために設計され、高性能グラフィクスおよびVMークライアント間の入力リダイレクションを可能にしています。

SPICEはゲスト・フレームバッファを管理してクライアントへのグラフィック更新を効率よくストリーミングします。これはフレームバッファへの変更を監視し、クライアントへは更新された領域のみを送信します。また、クライアントで使用されるグラフィック領域を頻繁にキャッシュに格納して、それらの再送信の必要性を削減しています。

VNCのようにSPICEはリモート・デスクトップを配信するために連携して動作するいくつかのコンポーネントを使ってクライアントーサーバー・プロトコルとして動作します。サーバーは(ハイパーバイザー内の)ホスト・マシン上で実行し、グラフィック、入力、およびマルチメディア・データ・ストリームを管理します。クライアントはサーバーに接続し、ゲストVMのディスプレイを受信すると同時にマウス、キーボード等から入力イベントも処理します。

SPICEはクリップボード共有、動的な解像度変更等の追加機能を提供するためにゲストの中で動作するソフトウェア・コンポーネントも備えています。一部の機能は使用しているグラフィクス・アダプタ(QXL vs Virtio GPU)によっては実行できない可能性があります。

SPICEは特にマルチメディアの重たい作業負荷やデスクトップとのやり取り時に通常は優れた性能を提供します。これは幾分クライアント側のキャッシングや高度な圧縮メソッドのような機能によるものです。

参考文献：

https://www.spice-space.org/documentation.html

グラフィクスPCIパススルー

GPUパススルーは、ホスト上の物理グラフィクス・カードの完全なアクセスを仮想マシンに提供します。IOMMUは安全なメモリ分離とアクセスを提供する一方、VFIOはゲストOS内のグラフィクス・ドライバが直接デバイスにアクセスすることを許可します。

結果、仮想マシンはまるで物理システムであるかのように全ての物理GPUの機能を利用できるようになります。使用するGPUにもよりますが、これはマルチ・ディスプレイ、3Dアクセラレーション、CUDAなどを含んでいます。

PCIパススルーを使うことで、ホストや他の仮想マシンに使用されるのを防ぐことに注意して下さい。必ずしもスケーリングが良いわけではないものの本メソッドは最高のグラフィック性能と低レイテンシーを提供します。

8
解析およびデバッグ

本章では、仮想化された環境の性能の問題を解析またはシステムの問題をデバッグするために使用可能なシステム・ツールを取り上げます。

新しいマルチ・マージ・トレース機能はRedHawkオペレーティング・システムの最新リリースに含まれています。これはタイム・スタンプで整理された1つのビューに複数のシステムのトレース・ダンプを統合することを可能にします。この新しい機能は、リアルタイム・アプリケーションの性能に影響を及ぼす可能性のあるVM-ホスト間の相互作用を頻繁に引き起こす仮想化環境のデバッグでは不可欠です。

マルチ・マージ・トレース機能を活用するには、トレースする全てのゲストVMは時刻クロック(TOD: Time Of Day)を使って同期する必要があります。トレースするために各ゲストVM上でchronyを開始するには6-1ページの「chrony実行の手順」項を参照して下さい。

NOTE

タイム・スタンプ・カウンター(TSC)は同期させることが出来ないため、複数のシステムのトレースを行う場合はTODタイム・スタンプ型のみを使用する必要があります。トレース・ツールでTODタイム・スタンプ・クロック・オプションを必ず選択して下さい。

本章では次の情報を提示します：

- RedHawkでサポートされるKVMトレース・イベント。
- **`xtrace`**と総称するRedHawkトレース・ツールの簡単な説明。これらのツールは簡素なコマンド・ライン・インターフェースを使用します。xtraceおよび新しいマルチ・マージ機能を使ったホストと1つのゲストVMをトレースする実例を含みます。
- KVM-RTゲスト・サービスという名前の新しいサービス。KVM-RTゲスト・サービスは、ゲストのユーザー空間アプリケーションにホスト・ハイパーバイザーにより公開された機能をアクセスする機会を与える新しいアプリケーション・プログラマー・インターフェース群です。

NightTraceはConcurrent Real-Timeが提供するオプション製品です。NightTraceはNightStarファミリーの一部で対話式デバッグや性能解析ツール、トレース・データ収集デーモン、データ値の記録やユーザーまたはカーネルから採取したデータの解析をユーザー・アプリケーションで可能にする2つのアプリケーション・プログラミング・インターフェース(API)で構成されます。

KVM-RTでNightTraceを使用する方法の情報については、NightTrace User's Guideの「Kernel Tracing with KVM-RT」項を参照して下さい。

KVMトレース・イベント

以下はRedHawkオペレーティング・システムでサポートされるKVMのトレース可能なイベントです。

`KVM_ENTER_VM_PID`

これはホスト・カーネルからゲストVMに実行/制御が転送される毎に引き起こされる一般的な包括的イベントです。ホストからゲストへの移行直前にホスト・システム上のKVMモジュールにより生成されます。

`KVM_EXIT_VM_PID`

これはゲストVMからホスト・カーネルに実行/制御が転送される毎に引き起こされる一般的な包括的イベントです。ゲストからホストへの移行直後にホスト・システム上のKVMモジュールにより生成されます。

`KVM_EXIT_HANDLER`

本イベントは、MSRの読み取りおよび書き込みが読まれるまたは書き込まれるMSRの種類とデータと一緒に発生した場合のようなVMの終了に関する追加情報を提供するために記録されます。本ベントはKVM_EXIT_VM_PIDの直後に記録されます。

`KVM_GUEST_HC_START`

本イベントはホストへのハイパーコールを行う直前にゲストVMにより記録されます。

`KVM_GUEST_HC_END`

本イベントは制御がハイパーコールから戻った直後にゲストVMにより記録されます。

`KVM_HOST_HC_ENTER`

本イベントは実行が一般的なハイパーコール・ハンドラーに達する直前にホスト・システムにより記録されます。

`KVM_HOST_HC_EXIT`

本イベントは実行が一般的なハイパーコール・ハンドラーを終了した直後にホスト・システムにより記録されます。

xtraceを使ったカーネル・トレース

`xtrace`はダンプのトレースおよび解析で使用されるコマンド・ライン・インターフェースです。

xtraceはRedHawkオペレーティング・システムの**ccur-xtrace**パッケージに付属しており、**xtrace** -<function>という名前のいくつかのツールを含んでいます。本パッケージで提供される全てのコマンドとライブラリを参照するには、RedHawkシステムで次を実行して下さい：

```
rpm –ql ccur-xtrace
```

以下は後述する実例で直接呼ばれるツールです。簡単な説明といくつかのオプションのみを以下言及します。詳細および他のオプションを参照するには**--help**オプションを使用して下さい：

xtrace-run:

シェル・コマンドの実行中にxtraceデータをキャプチャします。本コマンドはコマンド・ラインで指定する必要があります。コマンド終了時に**xtrace-run**は停止します。**-o**オプションはxtraceデータが保存される出力ディレクトリの名称を指定します。**-m**上書きオプションはトレースが長時間行われxtraceデータが巨大になる場合に使用することが可能です。

xtrace-multi-merge:

コマンド・ラインで指定されたxtraceデータ・ディレクトリを1つのマルチ・マージ・ディレクトリに統合します。これらは**xtrace-run**が起動された時に生成されたディレクトリです。コマンド・ラインではホスト用に1つ、トレースするゲストVMごとに1つのディレクトリを指定します。**-o**オプションは生成するマルチ・マージ・ディレクトリのディレクトリ名称を指定します。時刻クロック(TOD)だけが同期可能であることに注意して下さい。

xtrace-view:

ユーザーが理解可能な書式でxtraceデータを総合し表示します。xtraceデータ・ディレクトリを指定する必要があります。

xtrace-ctl:

1つまたは複数のCPU上のカーネルxtraceモジュールの制御を提供します。非対話型モードでは、FLUSH, PAUSE, RESUMEのようなコマンドをコマンド・ラインで指定します。

実例：xtraceを使ったマルチ・マージ・トレース

本例ではホスト・システムとゲストVMでトレース・ダンプを同時にキャプチャし、その後2つのトレース・ダンプを1つに統合します。この例はユーザー・アプリケーションが最初の5分以内に失敗することが分かっていることを前提とします。

NOTE

ゲストVMをトレースする前に時刻同期を構成し、実行している必要があります。トレースする各々のVMでchronyを開始するには、6-1ページの「chrony実行の手順」項を参照して下さい。

1. バックグラウンドでホスト・システムをトレースし、ユーザー・アプリケーションが失敗するのにかかる時間よりも長い時間スリープします：

   rm –rf *xtrace-host*
   xtrace-run –m overwrite –t tod **–o** *xtrace-host* **sleep** 600 &

2. トレースをホストからリモートで開始します。ユーザー・アプリケーションがゲストVMで失敗した時、トレース・バッファがフラッシュされます：

   ssh *guest_vm* "**rm -rf** *xtrace-vm*;
   xtrace-run -m overwrite **-t** tod **-o** *xtrace-vm* \
       bash -c '(*userapp* || **xtrace-ctl flush**)'"

3. トレース・バッファをフラッシュし、ホストでトレースを停止します：

   xtrace-ctl flush stop

4. トレース・データ・ディレクトリをゲストVMからホスト・システムにコピーします：

   scp -r *guest-vm*:*xtrace-vm* **.**

5. ゲストとホストのトレース・ディレクトリを1つに統合します：

   xtrace-multi-merge -o *xtrace-merged* *xtrace-host* *xtrace-vm*

6. 統合されたトレースをタイム・スタンプに従って並べ替えて表示します：

   xtrace-view *xtrace-merged*

表示されるフィールドは**xtrace-view**へのオプションで制御されます。次の出力例のフィールドは、タイム・スタンプ(TOD)、ホスト名、CPU、イベントです。

CPUは各ホストに対するローカルなので以下の引用では、「vm1 0」はゲストVMのホスト名が「vm1」の仮想CPU 0を意味します。

```
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ゲスト・サービス

仮想化環境は、VMで実行しているハード・リアルタイム・アプリケーションの性能に有害な影響を及ぼしかねない複雑なVMとホスト間の相互作用を引き起こす可能性があります。これらの相互作用の一部は不定期かつ/または再現しにくい可能性があります。これらのケースでは標準的なトレースのアプローチは十分ではありません。

KVM-RTゲスト・サービスは、ゲストのユーザー空間アプリケーションにホスト・ハイパーバイザーにより公開された機能をアクセスする機会を与える新しいアプリケーション・プログラマー・インターフェース群です。

複雑なものを再現する1つの方法は、各アプリケーションの中に含まれている暗示する専門知識を活用することです。アプリケーションが特定の時間であるべき状態やタイミングまたは状態の違反が発生した時の状態をアプリケーションは知っています。そのような背景において、KVM-RTゲスト・サービスはアプリケーション開発者に次のような能力を提供します：

1. ゲストVMで実行中のアプリケーションから直接ホスト上の主要なロギング/トレース機能(例えば、syslog, NightTrace, xtrace)に関連のあるイベント/データを記録します。

2. ホスト上のxtraceバッファをフラッシュします。これはゲストとホストの両方のバッファをほぼ同時にフラッシュするためにゲスト上のxtraceバッファのローカル・フラッシュを組み合わせることが可能です。

3. イベントの順番を成立させるため、ホストのクロックのコンテキストで明示的に事前定義された一連のイベントを記録します。例えば、以下はホスト上の2つの異なるゲストで記録されました。

   VM1上：

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

   VM2上：

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

   ホストでは、ホストのクロックのコンテキストで一連のイベントを見ることが可能です：

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

コマンド・ライン・インターフェース**`kvmrt-gs`**およびライブラリ**`libccur_kvmrt_gs`**で提供されるKVM-RTゲスト・サービスの機能は、次項で簡単に説明します。

また、トレース可能なKVM-RTゲスト・サービスのイベントおよびホストとゲストVMで有効化すべきカーネル起動パラメータが後述されています。

KVM-RTゲスト・サービス・ライブラリ・インターフェース

次の機能がライブラリ**libccur_kvmrt_gs**を介して提供されます。オプションや利用法に関する詳細については**libccur_kvmrt_gs(3)**のmanページを参照して下さい。

manページは以下に記載されたいずれかの機能の名称を使って呼び出すことが可能です。例：
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`

KVMRT_GSインターフェースが存在し、有効化され、許可されていればtrueを返します。同様にkvmrt_gs_<function>_availableは、個々のKVMRT_GSの関数が存在し、有効化され、許可されていればtrueを返します。

インターフェースの可用性はいずれの関数の可用性を意味することではないことに注意して下さい。更に、利用可能な関数の呼び出しは様々な理由により常に失敗する可能性があります。

`kvmrt_gs_ping`

*cookie*を使ってハイパーバイザーにpingします。本関数の目的は、ゲスト側とホスト側から簡単にトレースし組み合わせることが可能な方法でハイパーバイザー上にてVMEXITイベントを明示的に発生させるためにコピーまたは割り当てのない簡単で軽量なメカニズムをゲストに提供することです。本インターフェースはxtraceが利用可能な場合に対応するxtraceイベントを生成します。

`kvmrt_gs_log_msg`

ハイパーバイザー側の標準カーネル・ロギング・メカニズムを介して短いASCIIテキスト・メッセージを記録します。*msg*は標準的なNULLで終了するC言語文字列へのポインターです。ハイパーバイザーといずれの中間層も文字列の最大長を制限しますので、さもなければメッセージが切り詰められる可能性があります。以下のkvmrt_gs_xtrace_log_dataも参照して下さい。

`kvmrt_gs_xtrace_flush`

ホストOSのFLUSH xtraceイベントを発生させます。

*scope*はFLUSHに影響を受けるCPUを制御します：

```
KVMRT_GS_XTRACE_CPU_CURRENT
KVMRT_GS_XTRACE_CPU_VM
KVMRT_GS_XTRACE_CPU_ALL
```

現在のCPU、現在のVMを処理している全てのCPU、ホスト・システムでアクティブな全てのCPUにそれぞれFLUSHを発行します。

`kvmrt_gs_xtrace_log_data`

ゲスト側とホスト側の2つが一致するxtraceイベントとして、***size***バイトを含む任意のバイナリの***data***バッファに記録します。ハイパーバイザーといずれの中間層も最大サイズを制限しますので、さもなければ記録されたデータが切り詰められる可能性があります。上述のkvmrt_gs_log_msgも参照して下さい。

KVM-RTゲスト・サービス・コマンド・ライン・インターフェース

次のコマンドが**kvmrt-gs**コマンド・ライン・インターフェースを介して提供されます。オプションと使用方法の詳細については**kvmrt-gs(1)**のmanページを参照して下さい。

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

`available`

KVM-RTゲスト・サービスが利用可能な場合はSUCCESSを返します。

`ping_available`

「ping」コマンドが利用可能な場合はSUCCESSを返します。

`ping` ***`COOKIE`***

ユーザーが選定した任意の整数(unsigned long int) ***`COOKIE`***を使ってハイパーバイザーにpingを実行します。

`log_msg_available`

「log_msg」コマンドが利用可能な場合はSUCCESSを返します。

`log_msg` ***`MESSAGE`***

ハイパーバイザー上のメッセージを記録します。***`MESSAGE`***は通常の引用符付きASCII文字列または16進数でエンコードされたバイト列のどちらかが可能です。

`xtrace_flush_available`

「xtrace_flush」コマンドが利用可能な場合はSUCCESSを返します。

`xtrace_flush` ***`SCOPE`***

ホストOS上のxtraceバッファをフラッシュします。***`SCOPE`***は次のいずれかが可能です：{0: 現在のCPU；1: 全てのVM CPU；2: 全てのホストCPU}

`xtrace_log_data_available`

「xtrace_log_data」コマンドが利用可能な場合はSUCCESSを返します。

`xtrace_log_data` ***`DATA`***

バイナリ・データの状態でxtraceイベントを記録します。***`DATA`***は通常の引用符付きASCII文字列または16進数でエンコードされたバイト列のどちらかが可能です。

KVM-RTゲスト・サービス・トレース・イベント

KVM-RTゲスト・サービスは様々なトレース・イベントを記録します。全てのイベント・タイプがペアとして生じ、ゲスト側では*_GUESTが記録され、ホスト側では*_HOSTが記録されます。

ダブル・ロギングのような目的の背景は、ホストとゲストVMのクロックが同期していない可能性がある、または相互関係が変動した場合にトレース・ログ内で予測可能な基準点を提供することです。

`KVMRT_GS_PING_GUEST`
`KVMRT_GS_PING_HOST`

これらはKVM-RTゲスト・サービスの「ping」機能により生成されます。詳細については**`kvmrt_gs_ping(3)`**を参照して下さい。

`KVMRT_GS_FLUSH_GUEST`
`KVMRT_GS_FLUSH_HOST`

これらはKVM-RTゲスト・サービスの「xtrace_flush」機能により生成されます。詳細については**`kvmrt_gs_xtrace_flush(3)`**を参照して下さい。

`KVMRT_GS_LOG_DATA_GUEST`
`KVMRT_GS_LOG_DATA_HOST`

これらはKVM-RTゲスト・サービスの「xtrace_log_data」機能により生成され、XTRACE_EV_CUSTOMに類似しています。詳細については**`kvmrt_gs_xtrace_log_data(3)`**を参照して下さい。

KVM-RTゲスト・サービス・カーネル起動パラメータ

KVM-RTゲスト・サービスは、起動時に次のカーネル・パラメータが有効化されている必要があることを要求します。1つはホスト・システムに、その他はゲストVMに特化している事に注意して下さい。

`kvm.kvmrt_gs_hc_host_enabled=`

[KVM, x86] KVMホストでKVM-RTゲスト・サービスのハイパーコールを有効にします。これを1(有効)に設定するとKVMRT_GSハイパーコールおよび関連するGS機能をゲストに提供することをホストに許可します。これはKVMモジュール用のホスト側パラメータです。デフォルトは0(無効)となります。

`kvmrt_gs_hc_guest_enabled=`

[KVM_GUEST, x86] KVMゲストでKVM-RTゲスト・サービスのハイパーコールを有効にします。これを1(有効)に設定するとKVMRT_GSハイパーコールおよびその機能がホストより提供された場合、検出し使用することをゲスト・カーネルに許可します。これはゲスト側のカーネル・パラメータです。デフォルトは0(無効)となります。

`kvmrt_gs_syscall_enabled=`

[KVM_GUEST, x86] KVMゲストでKVM-RTゲスト・サービスの間接システムコール(syscall)を有効にします。これを1(有効)に設定するとKVMRT_GS間接システムコールおよびその機能をゲストで実行中のユーザー空間アプリケーションに提供することをゲスト・カーネルに許可します。

これはゲスト側のカーネル・パラメータです。デフォルトは0(無効)となります。

付録A
事前認証されたシステムのNUMAマッピング

本付録では、事前認証されたプラットフォームのためのデバイス・スロットとI/OポートのNUMAノード・マッピングについて説明します。

KVM-RT製品はSupermicro M12SWA-TFとASUS Pro WS WRX90E-SAGE SEマザーボードを考慮しています。

Supermicro M12SWA-TFボードはAMD Ryzon Threadripper PRO 5975WXと5965WXモデルの両方をサポートしています。

これらのシステムはKVM-RT製品ラインとして事前承認されています。KVM-RT製品に関する詳細については、KVM-RT Release Notesの「KVM-RT製品」項を参照して下さい。

KVM-RTでのNUMAマッピングの重要性

システムのトポロジーを利用すると更に効率の良いKVM-RT構成をもたらします。これはデバイスの割込みを異なるNUMAノードに再割り当てする必要性を減らし、リアルタイムVMのレイテンシーが低下します。

NUMAノード・マッピングはホスト、リアルタイムと非リアルタイムのゲストVMのためにデバイスの配置を検討するのに重要です。リアルタイムVMと同じNUMAノード上にデバイスを配置したら、これらのデバイスが使用される時にデータへのより速いアクセスとより低いレイテンシーを確かにします。他方、リアルタイムVMで使用されないデバイスがリアルタイムVMと同じNUMAノードに割り当てられると干渉のリスクが増え、リアルタイムVMのレイテンシーが増加します。

最適な構成を実現する第一歩は、最初にVMが使用するデバイスを調査し次にVMに使用されるデバイスを配置可能なNUMAノードを選択することによりリアルタイムVM用に最善のNUMAノードを選択することです。

例えば、Supermicro M12SWA-Tシステムの略図を参照して下さい。2つのPCIeスロットを要求するリアルタイムVMはNUMAノード1またはNUMAノード2に配置することが可能です。一方1つのPCIeスロットと2つのUSBポートを要求するものはNUMAノード3に配置するのが最善となります。

デバイスのスロットとポートが全てのNUMAノード間で均等に分離されていないので妥協する必要の可能性があることに注意して下さい。更に、リアルタイムVMにとって理想の選択ではないNUMAノード0に優先権があります。リアルタイムVM用に選択されるNUMAノードにマップされるものを越えてより多くのPCIeスロットとより多くのデバイスが必要である場合、他のNUMAノードにより現在処理されるIRQはリアルタイムVMに選択されるNUMAノードに再割り当てされる必要があります。

NUMAノード・マッピングの略図

KVM-RTハードウェア構成を最適化することを目指し、略図はサポートしている各ボードの各々のNUMAノードにマッピングするデバイス・スロットやI/Oポートを提供しています。

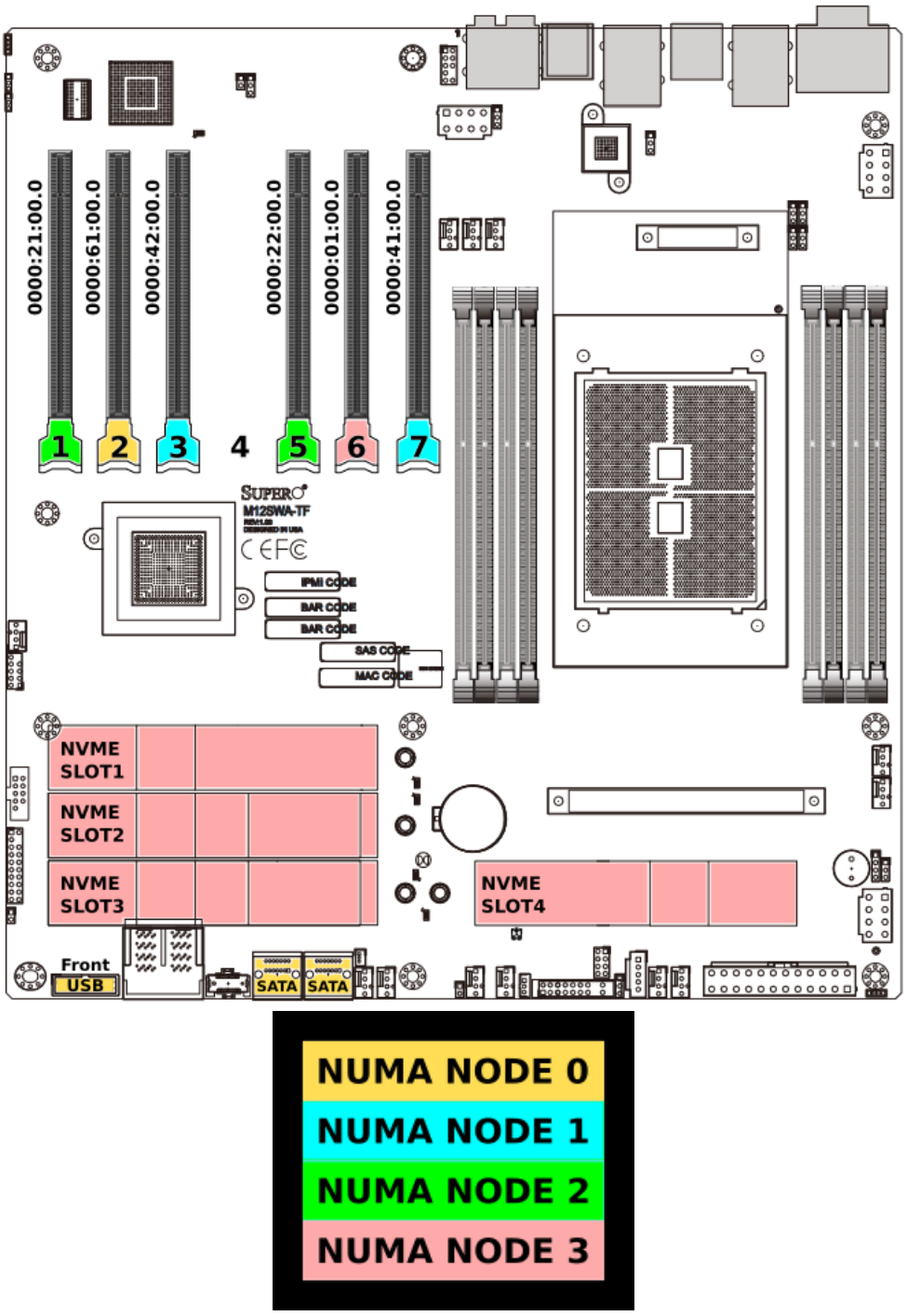
一部のオンボード・デバイスとポートのバス・アドレスは略図には含まれていません。これらのアドレスは使用中のデバイスやポートによって変化するためです。

デバイスの略図はNUMAノードにマッピングしている色分けされたデバイス・スロットを含んでいます。各PCIeスロットの横に表示されるバス・アドレスは再起動しても持続します。これはVMにPCIeパススルー用にデバイスを選択する際にそれらを識別するのを容易にします。

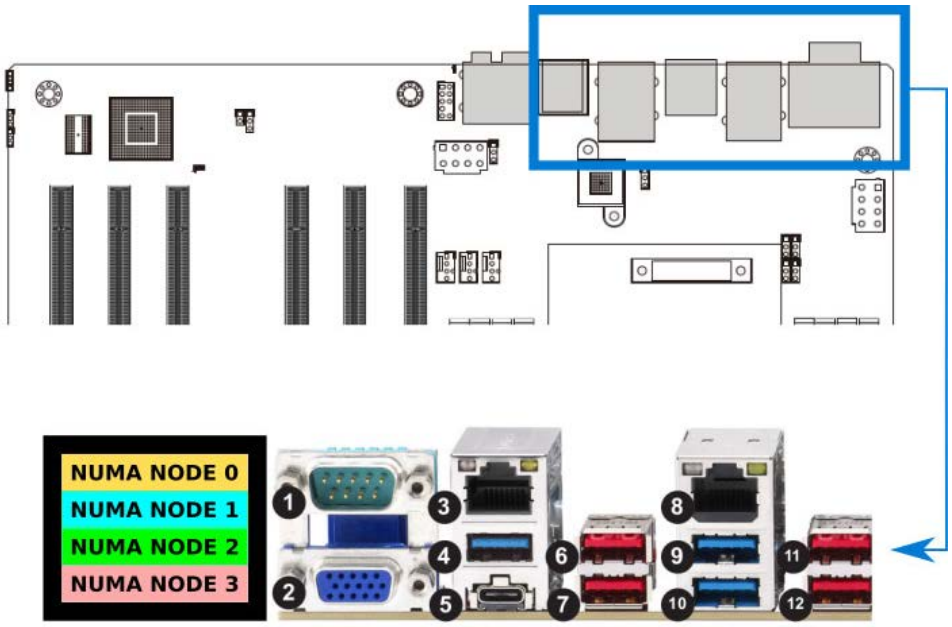
I/Oポートの略図は色分けされたNUMAノードのエントリの表を含んでいます。左側の数字は略図の中に示されているポート番号に対応しています。この表はどのポートまたはデバイス・ブリッジをVMにパススルーさせるかを決定する際に役立つ各ポートの詳細も含んでいます。

Supermicro M12SWA-FT

以下に示すマッピングは、BIOSリビジョン2.1が動作しているSupermicro M12SWA-TFのみ関連します。



図A-1 Supermicro M12SWA-TFデバイスに関するNUMAノード・マッピング

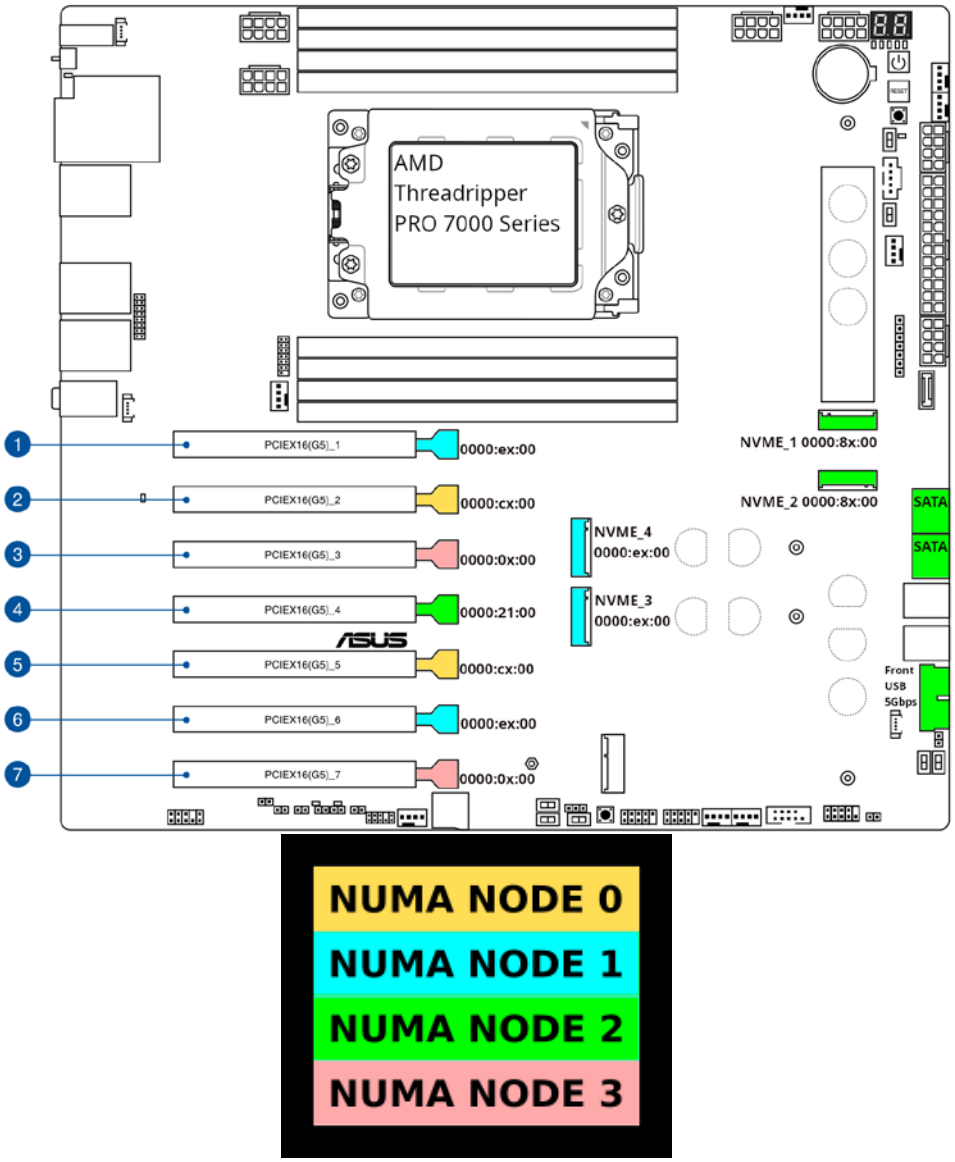


	Rear Panel I/O Ports	NUMA Node
1	COM1	0
2	VGA Port	0
3	1Gb LAN Port (i210)	0
4	USB3.2 Gen1 Type A, 5Gb/s	2
5	USB3.2 Gen2x2 Type C, 20Gb/s	0
6	USB3.2 Gen2 Type A, 10Gb/s	3
7	USB3.2 Gen2 Type A, 10Gb/s	3
8	10Gb LAN port (AQC113C)	0
9	USB3.2 Gen1 Type A, 5Gb/s	0
10	USB3.2 Gen1 Type A, 5Gb/s	0
11	USB3.2 Gen2 Type A, 10Gb/s	0
12	USB3.2 Gen2 Type A, 10Gb/s	0

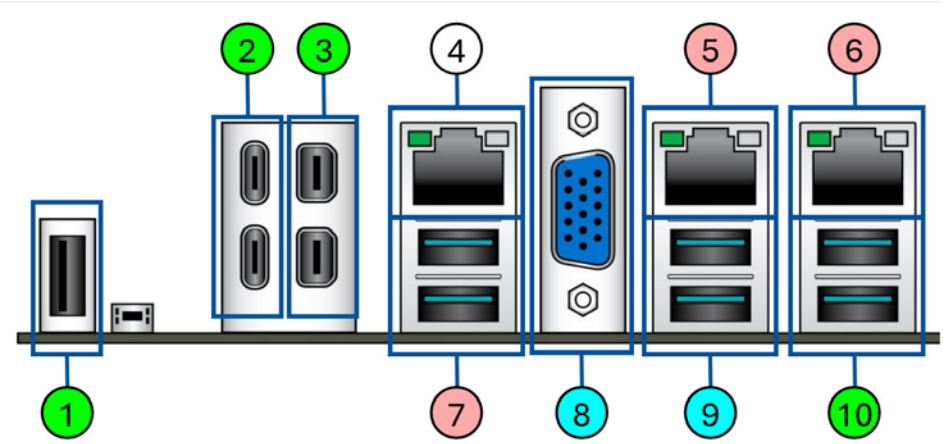
図A-2 Supermicro M12SWA-TFのI/Oポートに関するNUMAノード・マッピング

ASUS Pro WS WRX90E-SAGE SE

以下に示すマッピングは、BIOSリビジョン0803が動作しているASUS WS WRX90E-SAGE SEのみ関連します。



図A-3 ASUS Pro WS WRX90E-SAGE SEデバイスに関するNUMAノード・マッピング



	Rear Panel I/O Ports	NUMA Node
1	USB 2.0 Type A	2
2	USB 40Gbps with ASMedia ASM4242 controller	2
3	Mini DisplayPort IN	2
4	Realtek 1Gb Ethernet (MGMT Port)	-
5	Intel 10Gb Ethernet	3
6	Intel 10Gb Ethernet	3
7	USB 10Gbps Type A	3
8	VGA Port	1
9	USB 10Gbps Type A	1
10	USB 10Gbps Type A	2

図A-4 ASUS Pro WS WRX90E-SAGE SEデバイスのI/Oポートに関するNUMAノード・マッピング

付録B
SR-IOVの設定

本項はSR-IOVの設定を扱います。本項内の例ではIntel Ethernet Converged Network Adapter X550-T2を利用します。

一部の設定は仮想機能をゲストが利用する前に必要となります。SR-IOV機能を持つカードがホスト上にインストールされたら次のようにSR-IOVの設定を開始して下さい

1. SR-IOV機能を持つカードがホストにインストールされていることを確認し、**pci**コマンド出力で対象のカードを特定して下さい：

   # pci | grep -i ether

```
0000:21:00.0 Ethernet: Intel Corporation Ethernet Controller 10G X550T
0000:21:00.1 Ethernet: Intel Corporation Ethernet Controller 10G X550T
```

2. デバイスがSR-IOV機能を持っていること、同様に利用できるVFの総計と使用中のVFの数を確認して下さい。

   # lspci -vvv -s 0000:21:00.0

```
[...]
Capabilities: [160 v1] Single Root I/O Virtualization (SR-IOV)
    IOVCap: Migration-, Interrupt Message Number: 000
    IOVCtl: Enable+ Migration- Interrupt- MSE+ ARIHierarchy+
    IOVSta: Migration-
    Initial VFs: 64, Total VFs: 64, Number of VFs: 0, Function Dependency Link: 00
    VF offset: 128, stride: 2, Device ID: 1565
    Supported Page Size: 00000553, System Page Size: 00000001
    Region 0: Memory at 00000000b0d00000 (64-bit, non-prefetchable)
    Region 3: Memory at 00000000b0c00000 (64-bit, non-prefetchable)
    VF Migration: offset: 00000000, BIR: 0
```

   このデバイスは最大64個の仮想機能(Virtual Function: VF)を構成することが可能で現在構成されているのは0であることに注意して下さい。

3. システムが動作中にVFを生成して下さい。物理機能(Physical Function: PF)に関連付けるインターフェースを決定して下さい：

   # lshw -class net -businfo

```
Bus info               Device          Class   Description
=============================================================
pci@0000:21:00.0       enp33s0f0       network Ethernet Controller 10G X550T
pci@0000:21:00.1       enp33s0f1       network Ethernet Controller 10G X550T
```

   本例では、PCIアドレス「0000:21:00.0」でインターフェース「enp33s0f0」のX550Tカードの1番目のポートに関心を寄せています。

要望するインターフェースに対して構成する仮想機能の数を更新して下さい。実行例：

```
# echo 2 > /sys/class/net/enp33s0f0/device/sriov_numvfs
```

構成されたVFを表示して下さい：

```
# lshw -class net -businfo
```

```
Bus info                Device          Class   Description
=========================================================
pci@0000:21:00.0        enp33s0f0       network Ethernet Controller 10G X550T
pci@0000:21:00.1        enp33s0f1       network Ethernet Controller 10G X550T
pci@0000:21:10.0        enp33s0f0v0     network X550 Virtual Function
pci@0000:21:10.2        enp33s0f0v1     network X550 Virtual Function
```

上で指定したインターフェースに対して2つのVF(enp33s0f0v0とenp33s0f0v1)が生成されました。

VFは独自のIOMMUグループに分離されていますので、異なるゲストに対して同じPFを共有するVFを個々にPCIパススルーを行っていることに留意することは重要です。

```
# pci -G
```

```
[...]
    IOMMU Group 45:
      0000:21:00.0 Ethernet: Intel Corporation Ethernet Controller 10G X550T
    IOMMU Group 46:
      0000:21:00.1 Ethernet: Intel Corporation Ethernet Controller 10G X550T
    IOMMU Group 56:
      0000:21:10.0 Ethernet: Intel Corporation X550 Virtual Function
    IOMMU Group 57:
      0000:21:10.2 Ethernet: Intel Corporation X550 Virtual Function
```

また、どのように仮想機能が別のドライバを利用するのかに注意して下さい。

```
# pci -v 0000:21:00.0
```

```
0000:21:00.0 Ethernet: Intel Corporation Ethernet Controller 10G X550T
       (numa=2, iommu=45, driver=ixgbe, irq=48, msi_irqs=265-269)
       NIC "enp33s0f0" (ec:e7:a7:07:79:44)
```

```
# pci -v 0000:21:10.0
```

```
0000:21:10.0 Ethernet: Intel Corporation X550 Virtual Function
       (numa=2, iommu=56, driver=ixgbevf, irq=0, msi_irqs=270-272)
       NIC "enp33s0f0v0" (76:da:1f:8f:a3:c5)
```

4. 再起動後も仮想機能を持続させます。システム時同時に仮想機能が構成されることを確実にするには2つの手法があります。

a. 起動行パラメータ

前述したように物理機能のドライバを特定し、そのドライバを利用する各ネットワーク・カードに対しインターフェースごとに構成するには仮想機能の最大数で起動行パラメータを追加して下さい。例：

ixgbe.max_vfs=2

/etc/default/grubのGRUB_CMDLINE_LINUXの最後に起動オプションを追加し、ベース・ディストリビューションに適した方法でgrubファイルを更新して下さい。

Ubuntuシステム：

```
# update-grub
```

RHEL互換システム：

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

b. modprobe構成ファイル

物理機能のドライバ名を使って**/etc/modprobe.d**の中に構成ファイルを生成し、そのドライバを利用する各ネットワーク・カードに対しインターフェースごとに構成するには仮想機能の最大数をオプションに追加して下さい。

例：

```
# echo 'options ixgbe max_vfs=2' >>
/etc/modprobe.d/ixgbe.conf
```

5. 仮想機能が構成されたら、他のPCIデバイスと一緒にVFに関連付けられたPCIデバイスをゲストにパススルーします。

付録C
PCIパススルー起動パラメータ

本項で記載されているパラメータは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。カーネル起動パラメータを追加および削除するには**`blscfg(1)`**コマンドを使用し、変更を有効にするにはシステムを再起動して下さい。

RedHawkリリース8.Xで必須

RedHawkリリース8.Xが実行中のシステムでは次を設定することが必要になります：

`intel_iommu=on`

RedHawk 8.Xが動作するIntelベースのシステムはパススルーを有効にするためにカーネルで*intel_iommu=on*を有効にする必要があります。

ホスト性能向上用オプション

ホストの性能を向上させるため、次も有効にすることが可能です：

`iommu=pt`

本オプションはパススルーで使用されるデバイスに対してのみIOMMUを有効にして、優れたホスト性能を提供します。

NOTE

本オプションは全てのハードウェアでサポートされているわけではありません。パススルーが失敗する場合、本オプションは削除して下さい。

グラフィック・カードで必須

殆どのPCI-e (PCI Express)カードは追加のカーネル起動パラメータを設定する必要もなくPCIパススルーで使用することが可能です。グラフィック・カードは例外です。グラフィック・カードはホストのドライバに要求される前の起動初期にVFIOドライバにより要求される必要があります。

次の起動パラメータはグラフィック・カードをパススルーするために使用することが可能です。**`lspci -nnk`**コマンドはデバイスに関する必要な情報を得るために利用可能であることに注目して下さい。

PCIベンダーとデバイスIDは行末の角括弧([])の中にリスト表示されます。
BUS:SLOT.FUNCTIONの情報は最初に記載されます。グラフィック・カードに対し複数のデバイスがリスト表示される場合、全てのデバイスを含める必要があります。

デバイスを指定するには次の2つの起動パラメータのうち1つを使用することが可能です。最初の起動パラメータが一番簡単に使用できますが、2番目のものはシステムに同じベンダーとデバイスIDを持つカードが複数ある場合に必要となります。

1. *vfio-pci.ids*=[vendor:device,…]

   本パラメータはVFIOドライバに割り当てるPCIデバイスのカンマ区切りのリストを設定することが可能です。各デバイスはvender:deviceで指定します。

   NOTE

   本起動パラメータを使用時にシステムに同じベンダーとデバイスIDを持つ複数のカードがある場合、ホストはいずれのデバイスも適切に初期化および使用することが出来なくなります。

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

   本パラメータはVFIOドライバに割り当てるPCIデバイスのカンマ区切りのリストを設定します。各デバイスはBUS:SLOT.FUNCTIONで指定します。

   本起動パラメータはシステムに同じベンダーとデバイスIDを持つ複数のカードがあって、ホストが1枚以上のカードを利用できるようにしたい場合に使用する必要があります。

   NOTE

   システムのカードの物理的な配置に変更がある場合、BUS:SLOT.FUNCTIONの設定は変わっている可能性がありますので再評価する必要があります。変更した場合、カーネル起動パラメータの設定を更新してシステムを再起動する必要があります。