仮想マシン実装技術解説
ホスト型仮想マシンモニタKVMとその応用への期待

KVM : An Implementation of Virtual Machine Monitor - Overview and Prospective Applications

尾崎亮太 独立行政法人情報通信研究機構
中尾彰宏 東京大学

x86/x86-64 アーキテクチャ上の Linux にホスト型仮想マシン環境を構築する Kernel-based Virtual Machine（KVM）について述べる．また，KVM の応用例として，オーバレイネットワーク・テストベッド PlanetLab への適用案を紹介する．

特徴

KVM[5] は，x86 アーキテクチャ仮想化支援プロセッサ拡張機能（以下 x86 仮想化拡張）を利用し完全仮想化環境の仮想マシンを提供するホスト型仮想マシンモニタとして Linux 上に実装された．Linux カーネル，x86 仮想化拡張，PC エミュレータなどを積極的に活用し，短期間で実用に耐える機能性を得ることができた．最近では，準仮想化環境を取り入れ，課題であった低い I/O 性能を改善した．仮想マシンの実装としては比較的新しくまだ成熟度は低いが，導入の容易さや将来性の高さから近年注目されつつある．

《 ホスト型 》

ホスト型構成では，仮想マシンは物理マシン上に動く OS（以下ホスト OS）の上に構築され，その仮想マシン上で動く OS（以下ゲスト OS）およびアプリケーションをホスト OS 上の 1 つのプロセスとして扱う（以下，ゲスト OS とその上で動くアプリケーションを合わせてゲストと呼ぶ）．そのためホスト OS が提供するプロセス管理コマンド，たとえば `kill`，`renice`，`ionice`，`taskset` などでゲストを制御することができる．また特権ユーザでなくとも仮想マシンを起動することができる．さらにホスト OS の機能を活用することができるという利点も存在する．たとえば Linux カーネルのスケジューリング，メモリ管理，電力管理機能などを利用できる．

逆に欠点としては，仮想マシンモニタの規模が比較的大きく，障害の要因となりやすいことが挙げられる．仮想マシンモニタの一部となるホスト OS の障害がゲストへ与える影響は大きいため，仮想マシンモニタの障害は可能な限り避けなければならない．ホスト型構成より仮想マシンモニタが小さいハイパーバイザ構成に比べると，この点でシステムの信頼性が低いと思われる．

《 完全仮想化 》

KVM は後述する x86 仮想化拡張と PC エミュレータ QEMU[4] を併用することで完全仮想化仮想マシンを提供する．完全仮想化仮想マシン上では，物理マシン上で動くように実装された OS やアプリケーションが改変なしに仮想マシン内で動作する．そのため，レガシな OS を動作させたり，仮想化導入コストを低減することが可能である．

《 準仮想化の導入 》

完全仮想化により提供される利点は大きいが，仮想化環境構築のためのオーバヘッドは小さくない．特に I/O 処理におけるオーバヘッドは無視できるものではない．

そこで KVM は，x86 仮想化拡張を用いつつ準仮想化を導入する Hybrid-virtualization（以下複合仮想化）により性能を改善する方針を採用した．準仮想化は，物理マシンの仮想化支援機能がない環境において，ゲスト OS を仮想マシンに適合するように改良することで仮想化環境を構築し，さらにゲストの高い性能を実現する手法である[2]．複合仮想化は，x86 仮想化拡張による完全仮想化でオーバヘッドが大きくなるゲストの処理のみを改良し，性能向上を図る手法である[6]．準仮想化実装の 1 つである Xen とは異なり，KVM はゲスト OS のデバイスドライバのみを入れ替えるだけで準仮想化環境を提供することができる．

《 各種機能 》

KVM は以下のような機能を持つ．

- 32/64bit ホスト上の 32/64bit ゲスト（ただし 32bit ホスト上の 64bit ゲストを除く）
- SMP ホスト，SMP ゲスト（ゲストは最大 4CPU）
- ライブマイグレーション（同一サブネットワーク内）
- ゲストメモリスワッピング
- QEMU の機能の活用

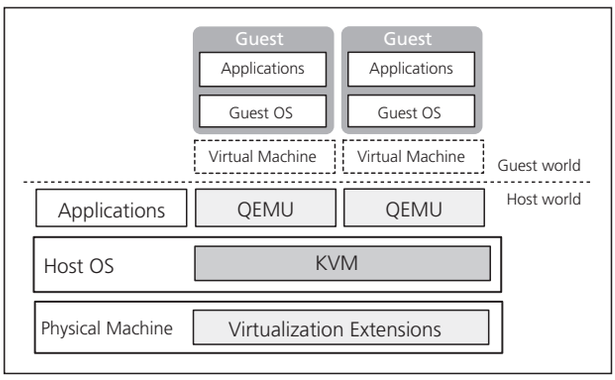

図 -1　KVM 構成

- 多様な周辺デバイスのエミュレーション
- gdb によるゲストのデバッグ
- VNC を用いた遠隔ゲスト操作
- ゲストのサスペンド・レジューム

構成要素

KVM は，**図 -1** のように x86 仮想化拡張，PC エミュレータ，そして Linux などの複数のコンポーネントを利用して仮想マシンを構築している．

《 KVM本体 》

KVM 本体は Linux のローダブルカーネルモジュールとして実現されている．仮想マシン構築のための最低限の機能のみ実装することでコードの肥大化や保守管理の複雑化を避けている．KVM 本体は，後述する x86 仮想化拡張の制御，ゲストメモリの管理，仮想マシンを構成するプロセッサと割込み制御ハードウェア（PIC, IOAPIC，Local APIC），そして KVM を用いて仮想マシンを構築する PC エミュレータ用 API を提供する．その他の仮想マシンを構成する機能，たとえば周辺デバイスのエミュレーションを含むほとんどの I/O 処理は PC エミュレータに委譲する．

《 x86 仮想化拡張 》

従来の x86 アーキテクチャ互換のプロセッサには，仮想マシンの構築を補助する機能が欠けていた．それだけでなく，ソフトウェア的に仮想マシンを構築することを困難にする，仮想化に影響を与える特権レベルでトラップできない一般命令が存在した．そのため，動的バイナリ変換技術や準仮想化技術などを用いる必要があった．しかし，近年の仮想化技術に対する要求から，Intel や AMD といった x86 プロセッサベンダは x86 プロセッサにも仮想化を支援する機能を実装した．

以下本稿では，x86 仮想化拡張の実装の 1 つである Intel VT-x[9]（以下 VT-x）の用語を説明に用いる．VT-x と AMD-V[1] では提供する機能に大きな差はないため，本稿の解説は AMD-V の場合にも適用できるものと思われる．

ここでは VT-x が提供する以下の 3 つの機能について述べる．その他 VT-x，AMD-V の x86 仮想化拡張の詳細については本連載の過去の記事を参照されたい．

- VMX operations：x86 アーキテクチャに存在した特権レベル（ring 0 ～ 3）と直交した動作モードとして `root operation` と `non-root operation` が導入された．仮想マシンモニタは root operation，ゲストは non-root operation で動作する．
- VMX operation 間のハードウェア的な遷移：仮想マシンモニタが横取りすべきゲストの処理やイベントをトラップし，仮想マシンモニタへ動作を移行させる．これを `VM exit` と呼び，逆にゲストに移行することを `VM enter` と呼ぶ．たとえば，特権命令の発行，I/O ポートアクセス，外部割込みなどが VM exit の対象となる．また仮想マシンモニタがゲストに仮想的な割込みを柔軟に挿入するための機能も提供する．
- VM exit 原因の報告：ゲストの処理内容を適切に把握するためにゲストが処理内容を仮想マシンモニタに提供する．たとえば，プロセッサが提供する報告から制御レジスタやモデル固有レジスタへのアクセス，I/O 命令の発行，TLB フラッシュ命令の発行などを仮想マシンモニタは判別することが可能となる．

現在普及している VT-x プロセッサは，仮想化に必要な機能をすべて備えているわけではなく，メモリ仮想化，Memory-mapped I/O（MMIO）のハンドリングは仮想マシンモニタによるソフトウェア的な対処を必要とする．

《 PC エミュレータ 》

完全仮想化を行うためには，プロセッサやメモリだけでなく周辺デバイスおよびそれとプロセッサ間で行われる I/O も仮想的に実現しなければならない．

KVM は PC エミュレータとして QEMU を用いて仮想的な周辺機器および I/O を実現している．QEMU はプロセッサ命令をエミュレートすることで単体でも完全仮想化環境を実現する PC エミュレータである．さらに異なるプロセッサアーキテクチャ命令を相互に変換することができ，たとえば，x86 アーキテクチャマシン上で SPARC アーキテクチャ向けの OS を動かすことができる．

仮想マシンの実現

仮想マシンを構築するということは，OS が物理マシンで管理する資源をすべて仮想化するということである．ここでは，主要な資源であるプロセッサ，メモリ，I/O

	root operation	*non-root operation*
ring 3	QEMU	Guest user processes
ring 0	KVM	Guest kernel

表 -1　KVM 実行モード（VT-x の場合）

の仮想化を KVM がどのように行っているかについて述べる．

《 プロセッサ仮想化 》

仮想マシン構築におけるプロセッサ仮想化には，ゲスト間およびホスト・ゲスト間のプロセッサ資源の分配やゲストの特権命令などのハンドリングが必要となる．KVM はこのほとんどを x86 仮想化拡張を活用し解決しているため，その手法の詳細は x86 仮想化拡張について説明した過去の連載記事に譲る．

KVM 仮想マシン環境では，ゲストは VT-x の non-root operation，QEMU は root operation の ring 3，KVM 自身は root operation の ring 0 で実行される（**表 -1**）．x86 仮想化拡張では，root operation の ring 0，つまり KVM のみがゲスト・仮想マシンモニタ間の移行およびそれを制御できる．つまり，ゲストが VM exit を起こした場合は必ず KVM へ移行し，逆にゲストへ移行することは KVM からしかできず，QEMU とゲスト間は直接は移行できない（**図 -2**）．たとえば，周辺デバイスをエミュレートする QEMU がゲストへ仮想割込みを挿入したい場合は，その要求を KVM へ依頼して，KVM がその処理を行う．

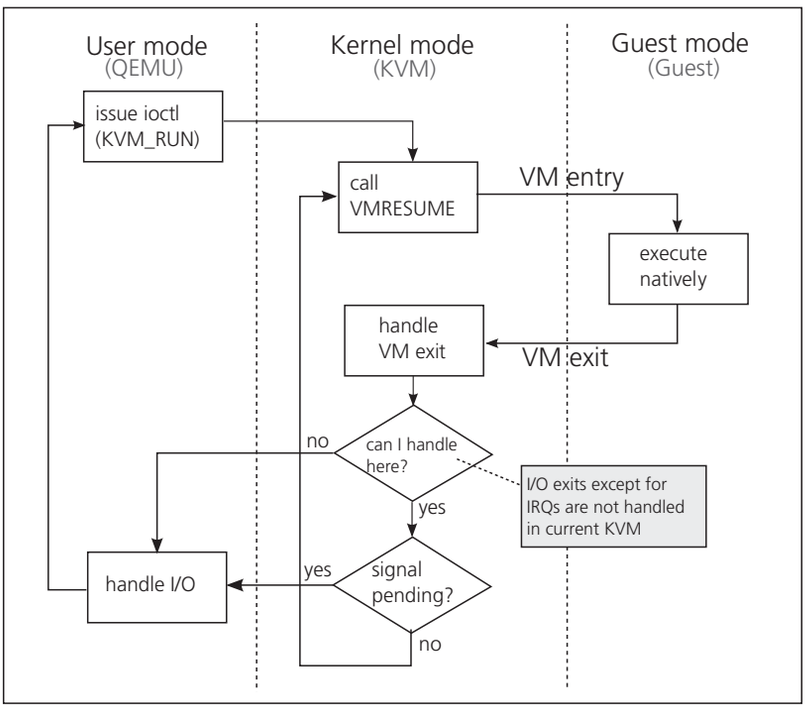

図 -2　ゲスト実行ループ

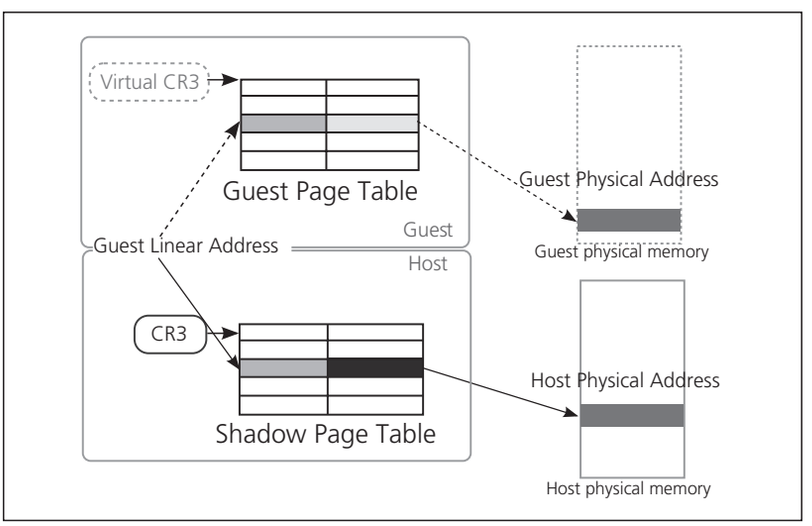

図 -3　シャドウページテーブル

《 メモリ仮想化 》

x86 プロセッサはメモリ仮想化機能（MMU）を提供しており個々のユーザプロセスに個別の仮想的なメモリ空間を提供している．MMU はその仮想アドレスを物理アドレスにハードウェアで変換する．しかし，同機能だけでは仮想マシンを構築する目的には不足である．仮想マシンモニタには，(1) ゲスト仮想アドレスからゲスト物理アドレスへの変換に加え，(2) ゲスト物理アドレスからホスト物理アドレスへの変換も必要となる．現在普及している x86 プロセッサの仮想化機能は (1) (2) の変換を 2 つ両方行うことはできないため，仮想マシンモニタがソフトウェア的に解決しなければならない．

KVM はシャドウページテーブルを用いてこれを解決している（**図 -3**）．シャドウページテーブルは KVM が管理するゲストごとに用意されるページテーブルであり，ゲスト仮想アドレスからホスト物理アドレスへの変換を行う．ゲスト OS は自身のページテーブルを持っているが，実際に MMU が参照する（CR3 レジスタが指す）のはこのシャドウページテーブルである．KVM は，ゲスト OS が CR3 やページテーブルを更新しようとするとき VM exit が起きるようにあらかじめ設定しておき，ゲストページテーブルの更新に合わせてシャドウページテーブルを更新する．シャドウページテーブルは最初空になっており，ゲストがページフォルトを起こすたびに充足していくようになっている．

KVM の初期の実装では，簡素化のために，ゲストのページテーブル更新のトリガとしてゲストが発行する

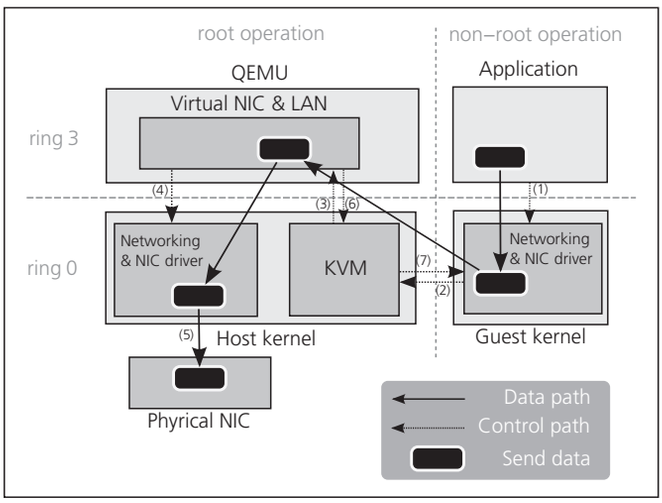

図 -4　I/O パス（外部ホストへのデータ送信の場合）

TLB フラッシュ命令を用いていた．ページテーブルを更新した際には MMU 内の TLB を無効化しなければならないため，TLB フラッシュ命令が発行されればゲストページテーブルが更新されたと判断できる．しかし，この実装ではどのページテーブルエントリが無効になったのかを検出できず，すべてのシャドウページテーブルをフラッシュしなければならないため，ゲスト切替のオーバヘッドが大きくなっていた．

現在の KVM ではどのページテーブルが無効化されたかを正しくハンドリングできるように改良されている．これを実現するため，ゲストページテーブルのページを書き込み禁止にして，ゲストの処理内容を解析した後，それをエミュレートし，ゲストページテーブルとシャドウページテーブルが完全に同期するようにした．これにより，シャドウページテーブルをコンテクスト切替をまたいで保持可能にしている．

《 I/O 仮想化 》

現在普及している x86 仮想化拡張および周辺デバイスは，I/O 仮想化の機能は不十分でソフトウェアによる支援が必要となる．KVM は QEMU を用いて物理デバイスをソフトウェア的にエミュレートする仮想デバイスをゲストに提供する．

OS のドライバが周辺デバイスを制御する際には Programmed I/O（PIO） や MMIO，Direct Memory Access（DMA）などのメモリアクセスが利用される．完全仮想化を実現する仮想マシンモニタは，これらをすべてエミュレートしなければならない．PIO は，x86 仮想化拡張が命令をトラップしてくれるが，MMIO は通常のメモリアクセスのため，MMIO 領域をアクセス禁止にして VM exit を誘発させなければ KVM がハンドリングすることはできない．PIO，MMIO どちらの場合も，KVM は命令解析を行い，ゲストの処理内容を明らかにしなければならない．KVM は解析結果から I/O 要求を再構築して，そのエミュレートを QEMU に依頼する．QEMU は，KVM の API を用いてゲスト物理メモリをすべて自身の仮想メモリにマップしているため，ゲスト物理メモリ全体にアクセスすることができる．そのため QEMU は DMA を単なるメモリコピーでエミュレートすることができる．

QEMU は必要な場合は物理デバイスを利用して仮想デバイスの処理を実現する．たとえば，ゲストがホスト外部へ通信するときには，一般的な OS のネットワーク API を使い物理ネットワークデバイスに対してデータを送受信する．ゲストアプリケーションが外部ホストへデータを送信する際の仮想化処理とデータの流れを**図 -4** に示す．（1）アプリケーションから渡されたデータを（2）ゲストカーネルのデバイスドライバが（仮想）ネットワークデバイスへ DMA 転送を依頼するときに，その I/O アクセスがプロセッサによりトラップされ（VM exit）処理が KVM へ移行する．（3）KVM はその I/O アクセスを解析し，仮想 I/O 処理を QEMU へ依頼する．（4）QEMU はマップされたゲストメモリ領域から DMA 転送されるデータをコピーし，ホストカーネルへ渡す（ホスト OS とのインタフェースとしては TAP が利用されることが多い）．（5）そしてホストカーネルのデバイスドライバは物理 NIC を通してそのデータを外部へ送信する．（6）一方 QEMU は，ホストカーネルへのデータ転送が完了した後，KVM に依頼して（6）DMA 完了を示す仮想割込みをゲストカーネルへ挿入し処理をゲストへ戻す（VM enter）．

この一連の処理を見ても，仮想化による処理量の増加が少なくないことが分かるが，実際には KVM および QEMU がトラップする I/O アクセスはもっと多い．たとえば，仮想 NIC の割込みステータスレジスタの参照や割込み制御レジスタを用いた割込み停止処理はすべてトラップされ，QEMU によってエミュレートされる．

現在の KVM では，周辺デバイスのエミュレーションを QEMU 内で行っているが，それによって仮想化のオーバヘッドが増加していることも事実である．周辺デバイスのエミュレーションを KVM 内で行い，仮想化処理とデータの流れを短縮することで性能の向上を図ることができる．しかしながら，QEMU の資産を活かすことができなくなる，カーネル内のコードの複雑さが増すといった問題もあり，性能とのトレードオフになる．

複合仮想化

仮想化支援ハードウェアと準仮想化技術を合わせて仮想マシンを構築する技術を複合仮想化という．前述のよ

うな I/O 処理の非効率さを避けるため，KVM は複合仮想化技術を取り入れた．

《 Virtio 》

KVM は `virtio` と呼ばれる準仮想化共通 API を用いて準仮想化デバイスとそのドライバを実装している．Virtio は，Rusty Russel が中心となって開発している Linux の軽量仮想マシンモニタである `lguest` の準仮想化ドライバの実装に用いられており，すでに Linux カーネルに取り込まれている．そのためゲストが Linux であれば KVM の準仮想化機能を容易に利用することができる☆1．この API を実装したゲストは，同じくこの API を実装した仮想マシンモニタの上で準仮想化の恩恵を受けることができる．

《 KVM virtio デバイス 》

Virtio を利用するには，ゲストの virtio ドライバと対になる virtio デバイスが仮想マシンモニタに必要となる．現在 KVM は，virtio ネットワーク，ブロック，PCI デバイスを実装している．ゲスト OS は，それらを一般的な PCI デバイスとそこに接続された周辺デバイスと同じように扱うことができる．

現在の KVM では，virtio デバイスを QEMU に実装している．Virtio ドライバとデバイスを実装するには，ゲスト・仮想マシンモニタ間のメモリ共有領域と，ゲスト・仮想マシンモニタ間の通知機能が必要となる．前述のとおり QEMU はゲストのメモリ領域を自身のメモリ空間内にマップしており，共有メモリ領域を作ることは容易である．ゲストから仮想マシンモニタへの通知には virtio PCI の I/O ポートを利用し，仮想マシンモニタからゲストへの通知は virtio PCI への割込みを利用している．

ここでは virtio ネットワークデバイスにおけるデータの送受信方法について述べる．データ送受信は，一般的なネットワークデバイスの DMA と同じようにデータバッファを指すディスクリプタリングを用いて行われる．ディスクリプタリングは送信用と受信用の 2 つ用意される．個々のディスクリプタは Linux のソケットバッファ構造体（`sk_buff`）に格納されたデータ領域を指す．ディスクリプタリングは，生産者・消費者モデルで virtio ドライバとデバイス間で管理され，たとえば送信用リングでは，virtio ドライバがバッファをディスクリプタに登録し virtio デバイスがそれを消費していく．

☆1 ただ残念なことに原稿執筆時の最新版である linux-2.6.24 の virtio ドライバは完全ではなく，Rusty の git ツリーを利用しなければ，virtio を試すことはできなかった．しかし，Linux メインツリーに取り込まれるのは時間の問題だと思われる．

Virtio デバイスと一般的な物理ネットワークデバイスとの違いは，物理的な制約がほぼないため理想的なデータ送受信処理を実現できることである．たとえば，ディスクリプタのサイズを多く取ることができたり，割込みの禁止，許可などを共有メモリ領域を介して行うことができる．そのため，ゲスト・仮想マシンモニタ間の遷移を最小限にしつつ，一度に渡すデータ量を最大にすることができる．

筆者らの実験環境では，virtio ネットワークの性能は仮想ネットワークデバイス（`rtl8139`）の場合に比べ 6 ～ 8 倍の性能を得ることができることを確認している[7]．しかし，まだ最適化の余地が残されており，さらなる性能向上も期待できる．

KVM の利点と欠点

KVM の利点としては，ホスト型仮想マシンモニタのため導入が容易で，ホストである Linux の機能を活用できることが挙げられる．ゲストは一般的なプロセスとほぼ同様に扱うことができ，既存のツールを利用できる．また KVM の発展はその他のシステムと共にあることも大きい．前述のとおり QEMU や Linux カーネルから恩恵を多く得ており，逆に KVM からそれらのコミュニティへのフィードバックも多い．このようなコミュニティを越えた連携が相互発展へつながり，KVM の将来性への期待となっていると筆者らは感じている．

逆に欠点としては，開発期間がまだそれほど長くないため，管理ツールなどが不足していることが挙げられる．特に，商用の仮想化ソフトウェアや Xen などと比べると，かなり少ないことは否めない．またサーバ統合などの利用を考えた場合，仮想ハードウェアの動的な追加・削除ができないことも問題である．しかし，これらの欠点は開発が続き，開発者が増えていくことで解消されていくものと思われる．

広域ネットワーク・テストベッドの基盤技術としての応用例

KVM の応用例として広域ネットワーク・テストベッド PlanetLab への適用を考える．現在 PlanetLab ではノード仮想化技術に Linux-VServer を用いているが，VServer を用いた仮想化では任意のゲスト OS を起動できない，通常はルーティングテーブルやファイアウォールの変更ができないなどさまざまな制限がある．KVM を Linux-VServer 内で動かすことにより，PlanetLab の資産を活かしつつ仮想マシンによる OS 非依存で柔軟なノード仮想化環境を提供することができる．

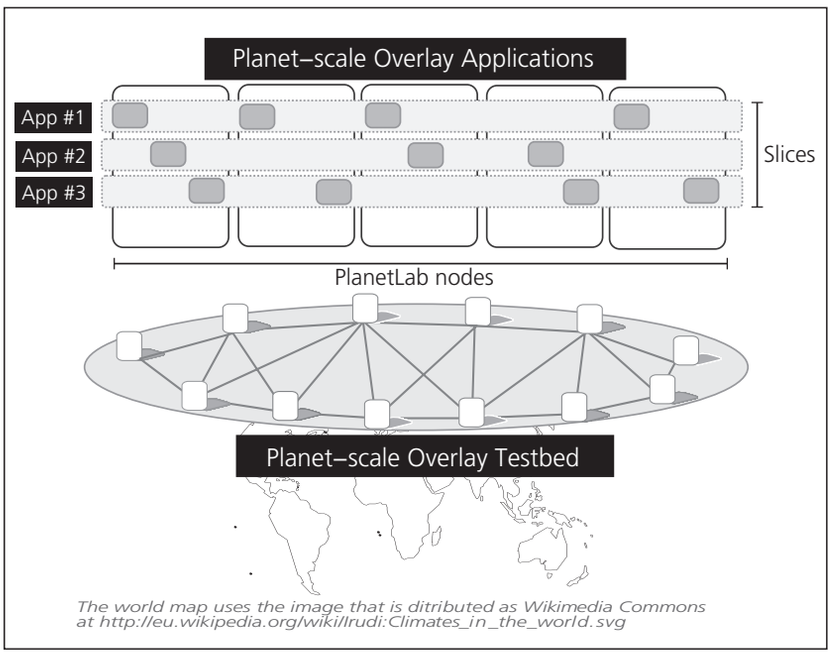

図 -5　PlanetLab

《 PlanetLabの概要 》

PlanetLab は，大学，研究機関から貸し出された，全世界に分散して存在するノード（PC サーバ）群を仮想ネットワーク上に結合し，参加者に実験環境を提供するテストベッドである（**図 -5**）．従来，広域分散ネットワークシステムの実験は，Emulab などのエミュレーション環境や LAN 内に閉じた実験室環境で行われることが多かった．PlanetLab は，それらの環境と違い，ノード群をインターネットを介して結合して構築された実験環境であるため，PlanetLab 上に展開された実験サービスはインターネットからアクセスが可能である．この特徴により，新しく広域分散ネットワークサービスを展開しようと考えた場合に，PlanetLab 上で，まずパイロットサービスを実行しユーザを獲得，革新的なネットワークサービスを少ない初期投資で早期展開することが可能である．

PlanetLab の各ノードの資源は，スライス（Slice）と呼ばれる単位で各々のネットワークサービスへ分配される．スライスは，複数のノードの資源の集合体と定義される．ネットワークサービス構築者は，割り当てられたスライスに属する資源（プロセッサ，メモリ，ストレージ，ネットワーク等）を用いてサービスを展開する．当然，特定のアプリケーション構築者に不適切に資源を多く割り当てたり，他のネットワークサービスのデータを盗用することを禁ずるために、PlanetLab の資源管理システムでは，スライスを適切に分配し，隔離する機能が必要となる．

PlanetLab は Linux-VServer[8] を改良した仮想化技術を用いてノード資源の仮想化を行う [3]（**図 -6**）．Linux-VServer の Virtual Private Server（VPS）と呼ばれるゲスト実行環境は，Linux カーネルを改良することで，さまざまな資源を分離し，個別のゲストごとに提供することを可能にする．たとえば，ファイルシステム，プロセス空間，IPC，ネットワークなどがホストやほかのゲストから切り離され個々のゲストに隔離された形で提供される．それに加えて，ホスト名やドメイン名，カーネルのバージョン，メモリやディスクの容量もホストとは異なる情報がゲストに渡される．PlanetLab 上の資源管理の単位であるスライスは，まさにこの VPS により実現されている．Linux-VServer をはじめとする Container 型仮想化システムの利点は，仮想化された物理マシン資源をゲストに提供する KVM や Xen などに比べて仮想化オーバヘッドが小さいことが挙げられる．PlanetLab では現在 850 以上の実験プロジェクトの各々がスライス上に展開されており，この程度のスケーラビリティを実現するためには Linux-VServer などのオーバヘッドのできるだけ小さい Container 型の仮想化技術が必須である．しかしながら，Container 型は後述のように，柔軟性の面で課題があるのも事実である．

《 PlanetLabの課題とKVMを用いた解決案 》

オーバヘッドの観点から仮想化性能が優れている Linux-VServer に代表される，Container 型の仮想化技術には，いくつかの問題点も残されている．たとえば，ゲスト環境はホスト環境のサブセットになるため，Linux-VServer の場合，ゲスト OS として必然的に Linux しか選択できないことになる．そのため，PlanetLab におけるネットワークサービスの開発者は，カスタムカーネルを利用できない，特定のカーネルバージョンでしか動かないソフトウェアを選択できないなど，実験上の制約を受けることがある．特に，ネットワークサービスの中でも，オーバレイルーティングやネットワークプロトコルの研究開発を行う場合，カーネルのカスタマイズを行ったり，ファイアウォールやルーティングテーブルの

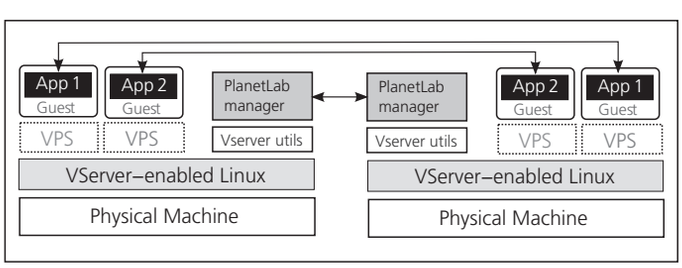

図 -6　PlanetLab ノード構成

変更をゲスト OS 上で自由に行うことができないなど不便な点が多い．

そこで，これらの課題を解決するために複数の仮想化技術を組み合わせて利用することが考えられる．たとえば，PlanetLab で Xen と Linux-VServer を組み合わせたり，KVM と Linux-VServer を組み合わせて利用することで，より柔軟な実験環境を構築することを考える．筆者らを含め複数の研究者が，Xen の完全仮想環境，準仮想化環境を用いた，カスタムカーネルを利用可能な PlanetLab 環境を構築した例がある．しかし Xen のようなハイパーバイザー型と VServer のような Container 型の仮想化技術の組合せはリソース管理の実装の観点からは容易ではない．一方，KVM では，ゲストは一般的なユーザプロセスとして扱われるため，Linux-VServer 内で動かすことが容易である．この特徴を考慮すれば，KVM と組み合わせることにより，Linux-VServer による資源分割を行いつつ，仮想マシンにより Linux 以外の OS も利用できる新しい PlanetLab 環境を構築することが可能になる．この構成のもう 1 つの利点は，PlanetLab 用に用意されたネットワーク・テストベッド管理システムを流用できることである．KVM には，前述のようにネットワーク I/O のパフォーマンスや開発ツールの点でいくつかの課題が残されているが，このようなホスト型の仮想化技術はホストの資源管理機構を再利用できるという利点があり，広域ネットワーク・テストベッドの資源管理基盤技術として筆者らが今後の発展に最も期待する仮想化技術の 1 つである．

まとめ

本稿では，x86 仮想化拡張を利用し仮想マシンを提供するホスト型仮想マシンモニタである KVM について概説した．KVM がどのように仮想マシンを構築し，準仮想化技術をどのように取り入れているか説明した．また PlanetLab の事例では既存のネットワーク・テストベッド環境に組み込んで仮想マシン環境を提供することで既存システムの柔軟性を向上させる KVM の活用法を紹介した．

《 KVMの今後 》

KVM はまだ未成熟で改善の余地が多く残されているが，多くの開発者たちの努力により日々急速に発展してきている．この原稿を執筆中にも多くの興味深いパッチが登場している．たとえば以下のような機能が KVM に実装されつつある．

- Intel Extended Page Table 対応
- AMD Nested Paging（Rapid Virtualization Indexing）対応
- IA-64 プロセッサ対応
- Virtio Balloon デバイス
- ゲスト間メモリ共有
- 準仮想化 MMU（`paravirt_ops` API 使用）
- KVM 内 PIT
- e1000 デバイスエミュレーション（QEMU）

本誌が発行されるころにはこれらの機能はすでに KVM に取り込まれていると思われる．

（編集者追記）

今回で仮想マシン道しるべの連載は終了となります．近年の仮想化技術は大変な進歩を遂げ，さまざまな領域で活用されておりますが，その基本的な仕組みや，個々の技術の特徴を知る上で，良いリファレンスになったのではないかと思います．仮想化技術にかかわる方々に，少しでも役立てていただければ，編集者一同，嬉しい限りです．

参考文献

1) AMD : AMD64 Virtualization Codenamed "Pacifica" Technology, Secure Virtual Machine Architecture Reference (2005).
2) Barham, P., et al. : Xen and the Art of Virtualization, Proc. *The 19th ACM Symposium on Operating Systems Principles*, pp.164-177 (2003).
3) Bavier, A. et al. : Operating System Support for Planetary-scale Network Services, Proc. *The 1st Conference on Symposium on Networked Systems Design and Implementation*, pp.253-266 (2004).
4) Bellard, F. : QEMU, a Fast and Portable Dynamic Translator, Proc. *USENIX 2005 Annual Technical Conference, FREENIX Track*, pp.41-46 (2005).
5) Kivity, A., et al. : kvm: the Linux Virtual Machine Monitor, Proc. *Ottawa Linux Symposium 2007*, pp.225-230 (2007).
6) Nakajima, J., et al. : Hybrid-Virtualization - Enhanced Virtualization for Linux, Proc. *Ottawa Linux Symposium 2007*, pp.87-96 (2007).
7) Ozaki, R., et al. : Analysis of Network I/O Performance in KVM, Proc. *IPSJ SIG Technical report*, 2008-OS-107, pp.111-118 (2008).
8) Soltesz, S. et al. : Container-based Operating System Virtualization : a Scalable, High-performance Alternative to Hypervisors, Proc. *the ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pp.275-287 (2007).
9) Uhlig, R., et al. : Intel Virtualization Technology, IEEE Computer, Vol.38, No.5, pp.48-56 (2005).

（平成 20 年 2 月 27 日受付）

尾崎亮太（正会員）
ozaki-r@nict.go.jp
1979 年生．2003 年電気通信大学情報工学専攻博士前期課程修了．2006 年総合研究大学院大学情報学専攻博士後期課程修了．博士（情報学）．国立情報学研究所特任研究員を経て，現在，情報通信研究機構専攻研究員．システムソフトウェア，オペレーティング・システム，仮想マシン，並列分散処理などに興味を持つ．

中尾彰宏（正会員）
nakao@iii.u-tokyo.ac.jp
1968 年生．東京大学理学部物理学科卒業．同大学院工学系研究科情報工学専攻修士修了．日本 IBM 東京基礎研究所を経て米国 Princeton 大学大学院情報科学科にて修士および博士修了．現在，東京大学大学院情報学環准教授．専門は，コンピュータネットワーク，オペレーティング・システム，システムソフトウェア，ネットワーク仮想化，オーバレイネットワークなど．